
MindSpore API 文档

发布 **2.6.0-rc1**

MindSpore

2025 年 05 月 08 日

1	mindspore	1
1.1	数据表达	1
1.2	运行环境	188
1.3	随机种子	205
1.4	随机状态管理	207
1.5	序列化	210
1.6	自动微分	224
1.7	并行优化	234
1.8	即时编译	237
1.9	工具	248
2	mindspore.ops	293
2.1	Tensor 操作函数	293
2.2	数学运算函数	406
2.3	神经网络层函数	561
2.4	Parameter 操作函数	707
2.5	微分函数	723
2.6	调试函数	726
2.7	稀疏函数	729
2.8	MC2 函数	778
3	mindspore.nn	783
3.1	基本构成单元	783
3.2	容器	820
3.3	封装层	825
3.4	卷积神经网络层	839
3.5	循环神经网络层	854
3.6	Transformer 层	862
3.7	嵌入层	870
3.8	非线性激活函数层	874
3.9	线性层	905
3.10	Dropout 层	908
3.11	归一化层	911
3.12	池化层	924
3.13	填充层	950
3.14	损失函数	963
3.15	优化器	996

3.16	动态学习率	1036
3.17	图像处理层	1050
3.18	公共层	1052
3.19	工具	1056
4	mindspore.mint	1057
4.1	Tensor	1057
4.2	随机采样	1098
4.3	数学运算	1103
4.4	mindspore.mint.nn	1214
4.5	mindspore.mint.nn.functional	1283
4.6	mindspore.mint.optim	1351
4.7	mindspore.mint.linalg	1358
4.8	mindspore.mint.special	1364
4.9	mindspore.mint.distributed	1368
5	mindspore.common.initializer	1403
6	mindspore.amp	1411
6.1	梯度缩放	1411
6.2	数据类型自动转换	1417
6.3	溢出检测	1422
7	mindspore.train	1425
7.1	模型	1425
7.2	回调函数	1433
7.3	评价指标	1456
7.4	工具	1481
8	mindspore.parallel	1485
8.1	并行基础配置	1485
8.2	模型加载与转换	1498
8.3	函数式并行切分	1505
8.4	其他	1510
9	mindspore.runtime	1517
9.1	执行	1517
9.2	内存	1520
9.3	流	1525
9.4	事件	1531
10	mindspore.device_context	1535
10.1	CPU 设备后端管理	1535
10.2	GPU 设备后端管理	1536
10.3	Ascend 设备后端管理	1540
11	mindspore.communication	1547
11.1	mindspore.communication.GlobalComm	1548
11.2	mindspore.communication.init	1548
11.3	mindspore.communication.release	1549
11.4	mindspore.communication.create_group	1549
11.5	mindspore.communication.destroy_group	1550
11.6	mindspore.communication.get_comm_name	1551
11.7	mindspore.communication.get_group_size	1552
11.8	mindspore.communication.get_group_rank_from_world_rank	1553

11.9	mindspore.communication.get_local_rank	1554
11.10	mindspore.communication.get_local_rank_size	1555
11.11	mindspore.communication.get_process_group_ranks	1556
11.12	mindspore.communication.get_rank	1556
11.13	mindspore.communication.get_world_rank_from_group_rank	1557
11.14	mindspore.communication.HCCL_WORLD_COMM_GROUP	1558
11.15	mindspore.communication.NCCL_WORLD_COMM_GROUP	1558
11.16	mindspore.communication.MCCL_WORLD_COMM_GROUP	1558
11.17	mindspore.communication.comm_func	1559
12	mindspore.dataset	1577
12.1	数据处理 Pipeline 介绍	1578
12.2	数据处理 Pipeline 快速上手	1579
12.3	自定义数据集加载	1579
12.4	标准格式数据集加载	1609
12.5	开源数据集加载	1620
12.6	其他数据集	1872
12.7	采样器	1881
12.8	配置	1893
12.9	工具	1910
13	mindspore.dataset.transforms	1925
13.1	通用	1925
13.2	视觉	1939
13.3	文本	2058
13.4	音频	2095
14	mindspore.mindrecord	2163
15	mindspore.numpy	2177
15.1	Array 生成	2177
15.2	Array 操作	2210
15.3	逻辑运算	2244
15.4	数学运算	2259
15.5	MindSpore Numpy 与 MindSpore 特性结合	2341
16	mindspore.scipy	2345
16.1	mindspore.scipy.linalg	2345
16.2	mindspore.scipy.optimize	2357
17	mindspore.multiprocessing	2361
17.1	介绍	2361
17.2	使用说明	2362
18	mindspore.utils	2365
19	mindspore.experimental	2367
19.1	实验性优化器	2367
19.2	实验性 EmbeddingService	2399
20	mindspore.ops.primitive	2407
20.1	算子原语	2407
20.2	神经网络层算子	2414
20.3	数学运算算子	2538
20.4	Tensor 操作算子	2622

20.5	Parameter 操作算子	2728
20.6	数据操作算子	2745
20.7	通信算子	2746
20.8	调试算子	2766
20.9	稀疏算子	2775
20.10	框架算子	2777
20.11	算子信息注册	2793
20.12	自定义算子	2803
20.13	光谱算子	2809
21	mindspore.boost	2813
22	mindspore.nn.probability	2829
22.1	Bijectors	2829
22.2	Distributions	2841
23	mindspore.rewrite	2919
24	mindspore.hal	2935
24.1	设备管理	2935
24.2	流管理	2940
24.3	事件管理	2944
24.4	内存管理	2946

1.1 数据表达

1.1.1 张量

<code>mindspore.Tensor</code>	张量，即存储多维数组（n-dimensional array）的数据结构。
<code>mindspore.tensor</code>	此接口用于在 <code>Cell.construct()</code> 或者 <code>@jit</code> 装饰的函数内，创建一个新的 <code>Tensor</code> 对象。
<code>mindspore.COOTensor</code>	用来表示某一 <code>Tensor</code> 在给定索引上非零元素的集合，其中索引（ <code>indices</code> ）指示了每一个非零元素的位置。
<code>mindspore.CSRTensor</code>	用来表示某一 <code>Tensor</code> 在给定索引上非零元素的集合，其中行索引由 <code>indptr</code> 表示，列索引由 <code>indices</code> 表示，非零值由 <code>values</code> 表示。
<code>mindspore.RowTensor</code>	用来表示一组指定索引的 <code>Tensor</code> 切片的稀疏表示。
<code>mindspore.SparseTensor</code>	用来表示某一张量在给定索引上非零元素的集合。
<code>mindspore.is_tensor</code>	判断输入对象是否为 <code>mindspore.Tensor</code> 。
<code>mindspore.from_numpy</code>	将 <code>Numpy</code> 数组转换为张量。

mindspore.Tensor

class `mindspore.Tensor` (`input_data=None`, `dtype=None`, `shape=None`, `init=None`, `const_arg=False`, `device=None`)

张量，即存储多维数组（n-dimensional array）的数据结构。

说明：

- 当使用 `init` 参数来初始化 `Tensor` 时，通常需要使用 `Tensor.init_data` 来加载 `Tensor` 的数据。
- CPU、GPU 的所有模式，以及 Atlas 训练系列产品的 图模式 (`mode=mindspore.GRAPH_MODE`) 尚不支持 in-place 操作。

- `input_data` 的默认值 `None` 只作为一个占位符，并不意味着可以创建一个 `NoneType` 的 `Tensor`。
- 当前对 `shape` 中包含 0 的 `Tensor` 支持的不完善。

警告：当转换 `Tensor` 类型时，推荐使用 `Tensor.astype()` 而非 `Tensor(sourceTensor, dtype=newDtype)`。

参数：

- **input_data** (`Union[Tensor, float, int, bool, tuple, list, numpy.ndarray]`) - 被存储的数据，可以是其他 `Tensor`，也可以是 Python 基本数据（如 `int`，`float`，`bool` 等），或是一个 NumPy 对象。默认值：`None`。
- **dtype** (`mindspore.dtype`) - 用于定义该 `Tensor` 的数据类型，必须是 `mindspore.dtype` 中定义的类型。如果该参数为 `None`，则数据类型与 `input_data` 一致，默认值：`None`。
- **shape** (`Union[tuple, list, int, mindspore.Symbol]`) - 用于定义该 `Tensor` 的形状。如果指定了 `input_data`，则无需设置该参数。如果 `shape` 中存在 `None` 或 `Symbol` 类型数据，表示创建一个动态形状（dynamic shape）的 `Tensor`，此时不需要设置 `input_data` 参数；如果 `shape` 中仅存在整数类型数据，表示创建一个静态形状（static shape）的 `Tensor`，此时必须设置 `input_data` 或 `init` 参数。默认值：`None`。
- **init** (`Initializer`) - 用于在并行模式中延迟 `Tensor` 的数据的初始化，如果指定该参数，则 `dtype` 和 `shape` 也必须被指定。默认值：`None`。
- **const_arg** (`bool`) - 指定该 `Tensor` 作为网络输入时是否为常量。默认值：`False`。
- **device** (`str`) - 该参数为保留参数，不需要用户配置。默认值：`None`。

输出：

`Tensor`。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.common.initializer import One
>>> # initialize a tensor with numpy.ndarray
>>> t1 = Tensor(np.zeros([1, 2, 3]), ms.float32)
>>> print(t1)
[[[0. 0. 0.]
 [0. 0. 0.]]]
>>> print(type(t1))
<class 'mindspore.common.tensor.Tensor'>
>>> print(t1.shape)
(1, 2, 3)
>>> print(t1.dtype)
Float32
>>>
>>> # initialize a tensor with a float scalar
>>> t2 = Tensor(0.1)
>>> print(t2)
0.1
>>> print(type(t2))
<class 'mindspore.common.tensor.Tensor'>
>>> print(t2.shape)
()
>>> print(t2.dtype)
```

(下页继续)

(续上页)

```
Float32
>>>
>>> # initialize a tensor with a tuple
>>> t3 = Tensor((1, 2))
>>> print(t3)
[1 2]
>>> print(type(t3))
<class 'mindspore.common.tensor.Tensor'>
>>> print(t3.shape)
(2,)
>>> print(t3.dtype)
Int64
...
>>> # initialize a tensor with init
>>> t4 = Tensor(shape = (1, 3), dtype=ms.float32, init=One())
>>> print(t4)
[[1. 1. 1.]]
>>> print(type(t4))
<class 'mindspore.common.tensor.Tensor'>
>>> print(t4.shape)
(1, 3)
>>> print(t4.dtype)
Float32
```

<code>mindspore.Tensor.abs</code>	详情请参考 <code>mindspore.ops.abs()</code> 。
<code>mindspore.Tensor.absolute</code>	<code>mindspore.Tensor.abs()</code> 的别名。
<code>mindspore.Tensor.acos</code>	详情请参考 <code>mindspore.ops.acos()</code> 。
<code>mindspore.Tensor.acosh</code>	详情请参考 <code>mindspore.ops.acosh()</code> 。
<code>mindspore.Tensor.add_</code>	<code>mindspore.Tensor.add()</code> 的 in-place 版本。
<code>mindspore.Tensor.add</code>	<code>self</code> 和 <code>other</code> 逐元素相加。
<code>mindspore.Tensor.addbmm</code>	详情请参考 <code>mindspore.ops.addbmm()</code> 。
<code>mindspore.Tensor.addcddiv</code>	详情请参考 <code>mindspore.ops.addcddiv()</code> 。
<code>mindspore.Tensor.addcmul</code>	详情请参考 <code>mindspore.ops.addcmul()</code> 。
<code>mindspore.Tensor.addmm</code>	详情请参考 <code>mindspore.ops.addmm()</code> 。
<code>mindspore.Tensor.addmm_</code>	<code>mindspore.Tensor.addmm()</code> 的 in-place 版本。
<code>mindspore.Tensor.addmv</code>	详情请参考 <code>mindspore.ops.addmv()</code> 。
<code>mindspore.Tensor.addr</code>	详情请参考 <code>mindspore.ops.addr()</code> 。
<code>mindspore.Tensor.adjoint</code>	详情请参考 <code>mindspore.ops.adjoint()</code> 。
<code>mindspore.Tensor.all</code>	检查指定维度上是否所有元素均为 <code>True</code> 。
<code>mindspore.Tensor.allclose</code>	返回一个布尔型标量, 表示 <code>self</code> 的每个元素均与 <code>other</code> 的对应元素在给定容忍度内“接近”。
<code>mindspore.Tensor.amax</code>	详情请参考 <code>mindspore.ops.amax()</code> 。
<code>mindspore.Tensor.amin</code>	详情请参考 <code>mindspore.ops.amin()</code> 。
<code>mindspore.Tensor.aminmax</code>	详情请参考 <code>mindspore.ops.aminmax()</code> 。
<code>mindspore.Tensor.any</code>	检查指定维度上是否含有 <code>True</code> 。
<code>mindspore.Tensor.angle</code>	详情请参考 <code>mindspore.ops.angle()</code> 。
<code>mindspore.Tensor.approximate_equal</code>	详情请参考 <code>mindspore.ops.approximate_equal()</code> , 当前接口参数 <code>other</code> 和参考接口中的参数 <code>y</code> 实际为同一参数。
<code>mindspore.Tensor.arccos</code>	<code>mindspore.Tensor.acos()</code> 的别名。
<code>mindspore.Tensor.arccosh</code>	<code>mindspore.Tensor.acosh()</code> 的别名。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.arcsin</code>	<code>mindspore.Tensor.asin()</code> 的别名。
<code>mindspore.Tensor.arcsinh</code>	<code>mindspore.Tensor.asinh()</code> 的别名。
<code>mindspore.Tensor.arctan</code>	<code>mindspore.Tensor.atan()</code> 的别名。
<code>mindspore.Tensor.arctan2</code>	<code>mindspore.Tensor.atan2()</code> 的别名。
<code>mindspore.Tensor.arctanh</code>	<code>mindspore.Tensor.atanh()</code> 的别名。
<code>mindspore.Tensor.argmax</code>	返回 <code>tensor</code> 在指定轴上的最大值索引。
<code>mindspore.Tensor.argmax_with_value</code>	返回 <code>tensor</code> 在指定轴上的最大值及其索引。
<code>mindspore.Tensor.argmin</code>	返回 <code>tensor</code> 在指定轴上的最小值索引。
<code>mindspore.Tensor.argmin_with_value</code>	返回 <code>tensor</code> 在指定轴上的最小值及其索引。
<code>mindspore.Tensor.argsort</code>	按指定顺序对 <code>self</code> 沿给定维度进行排序, 并返回排序后的索引。
<code>mindspore.Tensor.argwhere</code>	详情请参考 <code>mindspore.ops.argwhere()</code> 。
<code>mindspore.Tensor.asin</code>	详情请参考 <code>mindspore.ops.asin()</code> 。
<code>mindspore.Tensor.asinh</code>	详情请参考 <code>mindspore.ops.asinh()</code> 。
<code>mindspore.Tensor.asnumpy</code>	将张量转换为 NumPy 数组。
<code>mindspore.Tensor.assign_value</code>	将另一个 <code>Tensor</code> 的值赋给当前 <code>Tensor</code> 。
<code>mindspore.Tensor.astype</code>	将 <code>Tensor</code> 转为指定数据类型, 可指定是否返回副本。
<code>mindspore.Tensor.atan</code>	详情请参考 <code>mindspore.ops.atan()</code> 。
<code>mindspore.Tensor.atan2</code>	详情请参考 <code>mindspore.ops.atan2()</code> 。
<code>mindspore.Tensor.atanh</code>	详情请参考 <code>mindspore.ops.atanh()</code> 。
<code>mindspore.Tensor.baddbmm</code>	详情请参考 <code>mindspore.ops.baddbmm()</code> 。
<code>mindspore.Tensor.bernoulli</code>	详情请参考 <code>mindspore.mint.bernoulli()</code> 。
<code>mindspore.Tensor.bfloat16</code>	将输入 <code>Tensor</code> 转换为 <code>bfloat16</code> 类型。
<code>mindspore.Tensor.bincount</code>	详情请参考 <code>mindspore.ops.bincount()</code> 。
<code>mindspore.Tensor.bitwise_and</code>	逐元素执行两个 <code>Tensor</code> 的与运算。
<code>mindspore.Tensor.bitwise_not</code>	逐元素对当前 <code>Tensor</code> 进行位取反。
<code>mindspore.Tensor.bitwise_left_shift</code>	详情请参考 <code>mindspore.ops.bitwise_left_shift()</code> 。
<code>mindspore.Tensor.bitwise_or</code>	逐元素执行两个 <code>Tensor</code> 的或运算。
<code>mindspore.Tensor.bitwise_right_shift</code>	详情请参考 <code>mindspore.ops.bitwise_right_shift()</code> 。
<code>mindspore.Tensor.bitwise_xor</code>	逐元素执行两个 <code>Tensor</code> 的异或运算。
<code>mindspore.Tensor.bmm</code>	详情请参考 <code>mindspore.ops.bmm()</code> 。
<code>mindspore.Tensor.bool</code>	将输入 <code>Tensor</code> 转换为 <code>bool</code> 类型, 其中 <code>Tensor</code> 中的值为 0 时, 则为 <code>False</code> ; 非 0 时, 则为 <code>True</code> 。
<code>mindspore.Tensor.broadcast_to</code>	详情请参考 <code>mindspore.ops.broadcast_to()</code> 。
<code>mindspore.Tensor.byte</code>	将输入 <code>Tensor</code> 转换为 <code>uint8</code> 类型。
<code>mindspore.Tensor.cauchy</code>	使用柯西分布生成的数值填充当前 <code>Tensor</code> 。
<code>mindspore.Tensor.ceil</code>	详情请参考 <code>mindspore.ops.ceil()</code> 。
<code>mindspore.Tensor.cholesky</code>	详情请参考 <code>mindspore.ops.cholesky()</code> 。
<code>mindspore.Tensor.cholesky_solve</code>	详情请参考 <code>mindspore.ops.cholesky_solve()</code> 。
<code>mindspore.Tensor.choose</code>	根据原始 <code>Tensor</code> 数组和一个索引数组构造一个新的 <code>Tensor</code> 。
<code>mindspore.Tensor.chunk</code>	沿着指定轴 <code>dim</code> 将输入 <code>Tensor</code> 切分成 <code>chunks</code> 个 sub-tensor。
<code>mindspore.Tensor.clamp</code>	详情请参考 <code>mindspore.ops.clamp()</code> 。
<code>mindspore.Tensor.clamp_</code>	<code>mindspore.Tensor.clamp()</code> 的 in-place 版本。
<code>mindspore.Tensor.clip</code>	<code>mindspore.Tensor.clamp()</code> 的别名。
<code>mindspore.Tensor.clone</code>	返回一个当前 <code>Tensor</code> 的副本。
<code>mindspore.Tensor.col2im</code>	详情请参考 <code>mindspore.ops.col2im()</code> 。
<code>mindspore.Tensor.conj</code>	详情请参考 <code>mindspore.ops.conj()</code> 。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.contiguous</code>	将 Tensor 转成一个连续内存的 Tensor, 该 Tensor 包含与原 Tensor 相同的数据。
<code>mindspore.Tensor.copy</code>	复制一个 Tensor 并返回。
<code>mindspore.Tensor.copy_</code>	将 <code>src</code> 中的元素复制到 Tensor 中并返回这个 Tensor。
<code>mindspore.Tensor.copysign</code>	详情请参考 <code>mindspore.ops.copysign()</code> 。
<code>mindspore.Tensor.cos</code>	详情请参考 <code>mindspore.ops.cos()</code> 。
<code>mindspore.Tensor.cosh</code>	详情请参考 <code>mindspore.ops.cosh()</code> 。
<code>mindspore.Tensor.count_nonzero</code>	计算输入 Tensor 指定轴上的非零元素的数量。
<code>mindspore.Tensor.cov</code>	详情请参考 <code>mindspore.ops.cov()</code> 。
<code>mindspore.Tensor.cross</code>	详情请参考 <code>mindspore.ops.cross()</code> 。
<code>mindspore.Tensor.cummax</code>	详情请参考 <code>mindspore.ops.cummax()</code> 。
<code>mindspore.Tensor.cummin</code>	详情请参考 <code>mindspore.ops.cummin()</code> 。
<code>mindspore.Tensor.cumprod</code>	详情请参考 <code>mindspore.ops.cumprod()</code> 。
<code>mindspore.Tensor.cumsum</code>	计算输入 Tensor <code>self</code> 沿轴 <code>dim</code> 的累积和。
<code>mindspore.Tensor.deg2rad</code>	详情请参考 <code>mindspore.ops.deg2rad()</code> 。
<code>mindspore.Tensor.diag</code>	详情请参考 <code>mindspore.ops.diag()</code> 。
<code>mindspore.Tensor.diagflat</code>	详情请参考 <code>mindspore.ops.diagflat()</code> 。
<code>mindspore.Tensor.diagonal</code>	详情请参考 <code>mindspore.ops.diagonal()</code> 。
<code>mindspore.Tensor.diagonal_scatter</code>	详情请参考 <code>mindspore.ops.diagonal_scatter()</code> 。
<code>mindspore.Tensor.diff</code>	详情请参考 <code>mindspore.ops.diff()</code> 。
<code>mindspore.Tensor.digamma</code>	详情请参考 <code>mindspore.ops.digamma()</code> 。
<code>mindspore.Tensor.div</code>	详情请参考 <code>mindspore.ops.div()</code> 。
<code>mindspore.Tensor.div_</code>	<code>mindspore.Tensor.div()</code> 的 in-place 版本。
<code>mindspore.Tensor.divide</code>	<code>mindspore.Tensor.div()</code> 的别名。
<code>mindspore.Tensor.dot</code>	计算两个 1DTensor 的点积。
<code>mindspore.Tensor.double</code>	将输入 Tensor 转换为 <code>float64</code> 类型。
<code>mindspore.Tensor.dsplitt</code>	详情请参考 <code>mindspore.ops.dsplitt()</code> 。
<code>mindspore.Tensor.dtype</code>	返回 Tensor 的数据类型 (<code>mindspore.dtype</code>)。
<code>mindspore.Tensor.eigvals</code>	详情请参考 <code>mindspore.ops.eigvals()</code> 。
<code>mindspore.Tensor.eq</code>	详情请参考 <code>mindspore.ops.eq()</code> 。
<code>mindspore.Tensor.equal</code>	详情请参考 <code>mindspore.ops.equal()</code> 。
<code>mindspore.Tensor.erf</code>	详情请参考 <code>mindspore.ops.erf()</code> 。
<code>mindspore.Tensor.erfc</code>	详情请参考 <code>mindspore.ops.erfc()</code> 。
<code>mindspore.Tensor.erfinv</code>	详情请参考 <code>mindspore.ops.erfinv()</code> 。
<code>mindspore.Tensor.exp</code>	详情请参考 <code>mindspore.ops.exp()</code> 。
<code>mindspore.Tensor.exp_</code>	原地更新版本的 <code>mindspore.Tensor.exp()</code> 。
<code>mindspore.Tensor.expand</code>	详情请参考 <code>mindspore.ops.broadcast_to()</code> , 当前接口的参数 <code>size</code> 和参考接口中的参数 <code>shape</code> 实际为同一参数。
<code>mindspore.Tensor.expand_as</code>	将输入张量的 <code>shape</code> 扩展为另一个输入张量的 <code>shape</code> 。
<code>mindspore.Tensor.expand_dims</code>	详情请参考 <code>mindspore.ops.expand_dims()</code> 。
<code>mindspore.Tensor.expm1</code>	详情请参考 <code>mindspore.ops.expm1()</code> 。
<code>mindspore.Tensor.fill_</code>	用指定的值填充 <code>self</code> 。
<code>mindspore.Tensor.fill_diagonal</code>	将 <code>self</code> Tensor 的主对角线, 填充成指定的值, 并返回结果。
<code>mindspore.Tensor.flatten</code>	沿着从 <code>start_dim</code> 到 <code>end_dim</code> 的维度, 对输入 Tensor 进行展平。
<code>mindspore.Tensor.flip</code>	详情请参考 <code>mindspore.ops.flip()</code> 。
<code>mindspore.Tensor.fliplr</code>	详情请参考 <code>mindspore.ops.fliplr()</code> 。
<code>mindspore.Tensor.flipud</code>	详情请参考 <code>mindspore.ops.flipud()</code> 。
<code>mindspore.Tensor.float</code>	将输入 Tensor 转换为 <code>float32</code> 类型。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.float_power</code>	详情请参考 <code>mindspore.mint.float_power()</code> 。
<code>mindspore.Tensor.floor</code>	详情请参考 <code>mindspore.ops.floor()</code> 。
<code>mindspore.Tensor.floor_</code>	<code>mindspore.Tensor.floor()</code> 的 in-place 版本。
<code>mindspore.Tensor.floor_divide</code>	按元素将 <i>self</i> Tensor 除以输入 Tensor, 并向下取整。
<code>mindspore.Tensor.floor_divide_</code>	按元素将 <i>self</i> Tensor 除以 <i>other</i> Tensor, 并向下取整。
<code>mindspore.Tensor.flush_from_cache</code>	如果 Tensor 开启缓存作用, 则将缓存数据刷新到 host 侧。
<code>mindspore.Tensor.fmax</code>	详情请参考 <code>mindspore.ops.fmax()</code> 。
<code>mindspore.Tensor.fmod</code>	详情请参考 <code>mindspore.ops.fmod()</code> 。
<code>mindspore.Tensor.fold</code>	详情请参考 <code>mindspore.ops.fold()</code> 。
<code>mindspore.Tensor.frac</code>	详情请参考 <code>mindspore.ops.frac()</code> 。
<code>mindspore.Tensor.from_numpy</code>	将 Numpy 数组转换为张量。
<code>mindspore.Tensor.gather</code>	返回输入 Tensor 在指定 <i>index</i> 对应的元素组成的切片。
<code>mindspore.Tensor.gather_elements</code>	详情请参考 <code>mindspore.ops.gather_elements()</code> 。
<code>mindspore.Tensor.gather_nd</code>	详情请参考 <code>mindspore.ops.gather_nd()</code> 。
<code>mindspore.Tensor.gcd</code>	详情请参考 <code>mindspore.ops.gcd()</code> 。
<code>mindspore.Tensor.ge</code>	<code>mindspore.Tensor.greater_equal()</code> 的别名。
<code>mindspore.Tensor.geqrf</code>	详情请参考 <code>mindspore.ops.geqrf()</code> 。
<code>mindspore.Tensor.ger</code>	详情请参考 <code>mindspore.ops.ger()</code> 。
<code>mindspore.Tensor.greater</code>	详情请参考 <code>mindspore.ops.greater()</code> 。
<code>mindspore.Tensor.greater_equal</code>	详情请参考 <code>mindspore.ops.greater_equal()</code> 。
<code>mindspore.Tensor.gt</code>	详情请参考 <code>mindspore.ops.gt()</code> 。
<code>mindspore.Tensor.H</code>	返回共轭和转置的矩阵 (2-D Tensor) 的视图。
<code>mindspore.Tensor.half</code>	将输入 Tensor 转换为 <code>float16</code> 类型。
<code>mindspore.Tensor.hardshrink</code>	详情请参考 <code>mindspore.ops.hardshrink()</code> 。
<code>mindspore.Tensor.has_init</code>	判断 Tensor 是否已经初始化。
<code>mindspore.Tensor.heaviside</code>	详情请参考 <code>mindspore.ops.heaviside()</code> 。
<code>mindspore.Tensor.histc</code>	详情请参考 <code>mindspore.ops.histc()</code> 。
<code>mindspore.Tensor.hsplitt</code>	详情请参考 <code>mindspore.ops.hsplitt()</code> 。
<code>mindspore.Tensor.hypot</code>	详情请参考 <code>mindspore.ops.hypot()</code> 。
<code>mindspore.Tensor.i0</code>	详情请参考 <code>mindspore.ops.bessel_i0()</code> 。
<code>mindspore.Tensor.igamma</code>	详情请参考 <code>mindspore.ops.igamma()</code> 。
<code>mindspore.Tensor.igammac</code>	详情请参考 <code>mindspore.ops.igammac()</code> 。
<code>mindspore.Tensor.imag</code>	详情请参考 <code>mindspore.ops.imag()</code> 。
<code>mindspore.Tensor.index_add</code>	将 Tensor <i>y</i> 加到 Parameter 或 Tensor <i>self</i> 的指定 <i>axis</i> 轴和指定 <i>indices</i> 位置。
<code>mindspore.Tensor.index_add_</code>	根据 <i>index</i> 中的索引顺序, 将 <i>alpha</i> 乘以 <i>source</i> 的元素累加到 <i>self</i> 中。
<code>mindspore.Tensor.index_fill</code>	详情请参考 <code>mindspore.ops.index_fill()</code> 。
<code>mindspore.Tensor.index_put</code>	根据 <i>indices</i> 中的下标值, 使用 <i>values</i> 中的数值替换 Tensor <i>self</i> 中的相应元素的值。
<code>mindspore.Tensor.index_put_</code>	根据 <i>indices</i> 中的下标值, 使用 <i>values</i> 中的数值替换 Tensor <i>self</i> 中相应元素的值。
<code>mindspore.Tensor.index_select</code>	返回一个新的 Tensor, 该 Tensor 沿维度 <i>axis</i> 按 <i>index</i> 中给定的索引对 <i>self</i> 进行选择。
<code>mindspore.Tensor.init_data</code>	获取此 Tensor 的数据。
<code>mindspore.Tensor.inner</code>	详情请参考 <code>mindspore.ops.inner()</code> 。
<code>mindspore.Tensor.inplace_update</code>	详情请参考 <code>mindspore.ops.inplace_update()</code> 。
<code>mindspore.Tensor.int</code>	将输入 Tensor 转换为 <code>int32</code> 类型, 其中 Tensor 中的值为浮点数时, 则会丢弃小数部分。
<code>mindspore.Tensor.inv</code>	详情请参考 <code>mindspore.ops.inv()</code> 。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.inverse</code>	详情请参考 mindspore.ops.inverse() 。
<code>mindspore.Tensor.invert</code>	详情请参考 mindspore.ops.invert() 。
<code>mindspore.Tensor.isclose</code>	返回一个 <code>bool</code> 类型的 <code>Tensor</code> , 表示 <code>input</code> 的每个元素与 <code>other</code> 的对应元素在给定容忍度内是否“接近”。
<code>mindspore.Tensor.isfinite</code>	详情请参考 mindspore.ops.isfinite() 。
<code>mindspore.Tensor.is_complex</code>	详情请参考 mindspore.ops.is_complex() 。
<code>mindspore.Tensor.is_contiguous</code>	判断 <code>Tensor</code> 的内存是否连续。
<code>mindspore.Tensor.is_floating_point</code>	详情请参考 mindspore.ops.is_floating_point() 。
<code>mindspore.Tensor.isinf</code>	详情请参考 mindspore.ops.isinf() 。
<code>mindspore.Tensor.isnan</code>	详情请参考 mindspore.ops.isnan() 。
<code>mindspore.Tensor.isneginf</code>	详情请参考 mindspore.ops.isneginf() 。
<code>mindspore.Tensor.isposinf</code>	详情请参考 mindspore.ops.isposinf() 。
<code>mindspore.Tensor.isreal</code>	详情请参考 mindspore.ops.isreal() 。
<code>mindspore.Tensor.is_signed</code>	判断输入 <code>Tensor</code> 的 <code>dtype</code> 是否有符号数类型。
<code>mindspore.Tensor.item</code>	返回只包含一个元素的 <code>Tensor</code> 的元素值。
<code>mindspore.Tensor.itemset</code>	将标量插入到 <code>Tensor</code> (并将标量转换为 <code>Tensor</code> 的数据类型)。
<code>mindspore.Tensor.itemsize</code>	返回一个 <code>Tensor</code> 元素的长度 (以字节为单位)。
<code>mindspore.Tensor.lcm</code>	详情请参考 mindspore.ops.lcm() 。
<code>mindspore.Tensor.ldexp</code>	详情请参考 mindspore.ops.ldexp() 。
<code>mindspore.Tensor.le</code>	详情请参考 mindspore.ops.le() 。
<code>mindspore.Tensor.lerp</code>	详情请参考 mindspore.ops.lerp() 。
<code>mindspore.Tensor.less</code>	详情请参考 mindspore.ops.less() 。
<code>mindspore.Tensor.less_equal</code>	详情请参考 mindspore.ops.less_equal() 。
<code>mindspore.Tensor.log</code>	详情请参考 mindspore.ops.log() 。
<code>mindspore.Tensor.log10</code>	详情请参考 mindspore.ops.log10() 。
<code>mindspore.Tensor.log1p</code>	详情请参考 mindspore.ops.log1p() 。
<code>mindspore.Tensor.log2</code>	详情请参考 mindspore.ops.log2() 。
<code>mindspore.Tensor.logaddexp</code>	详情请参考 mindspore.ops.logaddexp() 。
<code>mindspore.Tensor.logaddexp2</code>	详情请参考 mindspore.ops.logaddexp2() 。
<code>mindspore.Tensor.logcumsumexp</code>	详情请参考 mindspore.ops.logcumsumexp() 。
<code>mindspore.Tensor.logdet</code>	详情请参考 mindspore.ops.logdet() 。
<code>mindspore.Tensor.logical_and</code>	详情请参考 mindspore.ops.logical_and() 。
<code>mindspore.Tensor.logical_not</code>	详情请参考 mindspore.ops.logical_not() 。
<code>mindspore.Tensor.logical_or</code>	详情请参考 mindspore.ops.logical_or() 。
<code>mindspore.Tensor.logical_xor</code>	逐元素计算两个 <code>Tensor</code> 的逻辑异或运算。
<code>mindspore.Tensor.logit</code>	详情请参考 mindspore.ops.logit() 。
<code>mindspore.Tensor.logsumexp</code>	详情请参考 mindspore.ops.logsumexp() 。
<code>mindspore.Tensor.log_normal</code>	使用给定均值 <code>mean</code> 和标准差 <code>std</code> 的对数正态分布的数值填充当前 <code>Tensor</code> 。
<code>mindspore.Tensor.long</code>	将输入 <code>Tensor</code> 转换为 <code>int64</code> 类型, 其中 <code>Tensor</code> 中的值为浮点数时, 则会丢弃小数部分。
<code>mindspore.Tensor.lt</code>	mindspore.Tensor.less() 的别名。
<code>mindspore.Tensor.lu_solve</code>	详情请参考 mindspore.ops.lu_solve() 。
<code>mindspore.Tensor.masked_fill</code>	详情请参考 mindspore.ops.masked_fill() 。
<code>mindspore.Tensor.masked_fill_</code>	mindspore.Tensor.masked_fill() 的 <code>in-place</code> 版本。
<code>mindspore.Tensor.masked_scatter</code>	根据 <code>mask</code> , 使用 <code>tensor</code> 中的值, 更新 <code>Tensor</code> 本身的值, 返回一个 <code>Tensor</code> 。
<code>mindspore.Tensor.masked_select</code>	详情请参考 mindspore.ops.masked_select() 。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.matmul</code>	计算两个数组的矩阵乘积。
<code>mindspore.Tensor.max</code>	返回输入 Tensor 的最大值。
<code>mindspore.Tensor.maximum</code>	详情请参考 <code>mindspore.ops.maximum()</code> 。
<code>mindspore.Tensor.mean</code>	默认情况下, 移除输入所有维度, 返回 <code>input</code> 中所有元素的平均值, 也可仅缩小指定维度 <code>dim</code> 大小至 1。
<code>mindspore.Tensor.median</code>	输出 Tensor 指定维度上的中值与索引。
<code>mindspore.Tensor.t</code>	详情请参考 <code>mindspore.ops.t()</code> 。
<code>mindspore.Tensor.mH</code>	访问此属性等价于调用 <code>self.adjoint()</code> 方法。
<code>mindspore.Tensor.min</code>	返回输入 Tensor 的最小值。
<code>mindspore.Tensor.minimum</code>	详情请参考 <code>mindspore.ops.minimum()</code> 。
<code>mindspore.Tensor.mm</code>	计算两个矩阵的乘积。
<code>mindspore.Tensor.moveaxis</code>	详情请参考 <code>mindspore.ops.moveaxis()</code> 。
<code>mindspore.Tensor.movedim</code>	详情请参考 <code>mindspore.ops.movedim()</code> 。
<code>mindspore.Tensor.move_to</code>	同步或异步的方式将 Tensor 拷贝到目标设备上, 默认同步方式。
<code>mindspore.Tensor.msort</code>	详情请参考 <code>mindspore.ops.msort()</code> 。
<code>mindspore.Tensor.mT</code>	返回将最后两个维度交换的 Tensor 。
<code>mindspore.Tensor.mul</code>	详情请参考 <code>mindspore.ops.mul()</code> 。
<code>mindspore.Tensor.mul_</code>	两个 Tensor 逐元素相乘。
<code>mindspore.Tensor.multinomial</code>	详情请参考 <code>mindspore.ops.multinomial()</code> 。
<code>mindspore.Tensor.multiply</code>	详情请参考 <code>mindspore.ops.mul()</code> 。
<code>mindspore.Tensor.mvlgamma</code>	详情请参考 <code>mindspore.ops.mvlgamma()</code> 。
<code>mindspore.Tensor.nan_to_num</code>	详情请参考 <code>mindspore.ops.nan_to_num()</code> 。
<code>mindspore.Tensor.nanmean</code>	详情请参考 <code>mindspore.ops.nanmean()</code> 。
<code>mindspore.Tensor.nanmedian</code>	详情请参考 <code>mindspore.ops.nanmedian()</code> 。
<code>mindspore.Tensor.nansum</code>	计算输入 Tensor 指定维度元素的和, 将非数字 (NaNs) 处理为零。
<code>mindspore.Tensor.narrow</code>	沿着指定的轴, 从指定的起始位置获取指定长度的 Tensor 。
<code>mindspore.Tensor.nbytes</code>	返回 Tensor 占用的总字节数。
<code>mindspore.Tensor.ndim</code>	返回 Tensor 维度的数量。
<code>mindspore.Tensor.ndimension</code>	<code>mindspore.Tensor.ndim</code> 的别名。
<code>mindspore.Tensor.ne</code>	<code>mindspore.Tensor.not_equal()</code> 的别名。
<code>mindspore.Tensor.neg</code>	详情请参考 <code>mindspore.ops.neg()</code> 。
<code>mindspore.Tensor.negative</code>	<code>mindspore.Tensor.neg()</code> 的别名。
<code>mindspore.Tensor.nelement</code>	<code>mindspore.Tensor.numel()</code> 的别名。
<code>mindspore.Tensor.new_ones</code>	返回一个大小为 <code>size</code> 的 Tensor , 填充值为 1。
<code>mindspore.Tensor.new_zeros</code>	返回一个大小为 <code>size</code> 的 Tensor , 填充值为 0。
<code>mindspore.Tensor.nextafter</code>	详情请参考 <code>mindspore.ops.nextafter()</code> 。
<code>mindspore.Tensor.nonzero</code>	返回所有非零元素下标位置。
<code>mindspore.Tensor.norm</code>	详情请参考 <code>mindspore.ops.norm()</code> 。
<code>mindspore.Tensor.normal_</code>	使用随机数原地更新 Tensor , 且随机数的采样服从由参数 <code>mean</code> 和 <code>std</code> 所构成的正态分布。
<code>mindspore.Tensor.not_equal</code>	详情请参考 <code>mindspore.ops.ne()</code> 。
<code>mindspore.Tensor.numel</code>	详情请参考 <code>mindspore.ops.numel()</code> 。
<code>mindspore.Tensor.numpy</code>	<code>mindspore.Tensor.asnumpy()</code> 的别名。
<code>mindspore.Tensor.orgqr</code>	详情请参考 <code>mindspore.ops.orgqr()</code> 。
<code>mindspore.Tensor.ormqr</code>	详情请参考 <code>mindspore.ops.ormqr()</code> , 其中参数 <code>input2</code> 和 <code>input3</code> 分别对应 <code>mindspore.ops.ormqr()</code> 的参数 <code>tau</code> 和 <code>other</code> 。
<code>mindspore.Tensor.outer</code>	详情请参考 <code>mindspore.ops.outer()</code> 。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.permute</code>	<code>Tensor.permute</code> 支持 <i>axis</i> 以不定数量的位置参数形式入参自动解包, 与 <code>mindspore.ops.permute()</code> 的入参有微小的差异。
<code>mindspore.Tensor.positive</code>	详情请参考 <code>mindspore.ops.positive()</code> 。
<code>mindspore.Tensor.pow</code>	详情请参考 <code>mindspore.ops.pow()</code> 。
<code>mindspore.Tensor.prod</code>	默认情况下, 使用指定维度的所有元素的乘积代替该维度的其他元素, 以移除该维度。
<code>mindspore.Tensor.ptp</code>	该函数名称是“peak to peak”的缩写。
<code>mindspore.Tensor.rad2deg</code>	详情请参考 <code>mindspore.ops.rad2deg()</code> 。
<code>mindspore.Tensor.random_</code>	在区间 <code>[from_, to - 1]</code> 内生成服从均匀分布的离散均匀分布随机数, 原地更新输入 <code>Tensor</code> 。
<code>mindspore.Tensor.random_categorical</code>	详情请参考 <code>mindspore.ops.random_categorical()</code> 。
<code>mindspore.Tensor.ravel</code>	返回一个展开的一维 <code>Tensor</code> 。
<code>mindspore.Tensor.real</code>	详情请参考 <code>mindspore.ops.real()</code> 。
<code>mindspore.Tensor.reciprocal</code>	详情请参考 <code>mindspore.ops.reciprocal()</code> 。
<code>mindspore.Tensor.register_hook</code>	设置 <code>Tensor</code> 对象的反向 <code>hook</code> 函数。
<code>mindspore.Tensor.remainder</code>	逐元素计算 <i>self</i> 除以 <i>other</i> 后的余数。
<code>mindspore.Tensor.renorm</code>	详情请参考 <code>mindspore.ops.renorm()</code> 。
<code>mindspore.Tensor.repeat</code>	对张量的每一维度按照指定的重复次数复制其中的元素。
<code>mindspore.Tensor.repeat_interleave</code>	沿着轴重复 <code>Tensor</code> 的元素, 类似 <code>mindspore.numpy.repeat()</code> 。
<code>mindspore.Tensor.reshape</code>	详情请参考 <code>mindspore.ops.reshape()</code> 。
<code>mindspore.Tensor.reshape_as</code>	不改变数据的情况下, 将 <code>Tensor</code> 的 <code>shape</code> 改为 <i>other</i> 的 <code>shape</code> 。
<code>mindspore.Tensor.resize</code>	将 <code>Tensor</code> 改为输入的新 <code>shape</code> , 并将不足的元素补 0。
<code>mindspore.Tensor.reverse</code>	详情请参考 <code>mindspore.ops.flip()</code> 。
<code>mindspore.Tensor.reverse_sequence</code>	详情请参考 <code>mindspore.ops.reverse_sequence()</code> 。
<code>mindspore.Tensor.roll</code>	详情请参考 <code>mindspore.ops.roll()</code> 。
<code>mindspore.Tensor.rot90</code>	详情请参考 <code>mindspore.ops.rot90()</code> 。
<code>mindspore.Tensor.round</code>	详情请参考 <code>mindspore.ops.round()</code> 。
<code>mindspore.Tensor.rsqrt</code>	详情请参考 <code>mindspore.ops.rsqrt()</code> 。
<code>mindspore.Tensor.scatter</code>	根据指定索引将 <i>src</i> 中的值更新到 <i>self</i> 中, 并返回输出。
<code>mindspore.Tensor.scatter_</code>	根据 <i>index</i> 使用 <i>src</i> 中的值更新当前张量 <i>self</i> 。
<code>mindspore.Tensor.scatter_add</code>	将 <i>src</i> 中所有的元素添加到 <i>self</i> 中 <i>index</i> 指定的索引处。
<code>mindspore.Tensor.scatter_add_</code>	将 <i>src</i> 中所有的元素添加到 <i>self</i> 中 <i>index</i> 指定的索引处 (属于原地更新操作)。
<code>mindspore.Tensor.scatter_div</code>	详情请参考 <code>mindspore.ops.scatter_div()</code> 。
<code>mindspore.Tensor.scatter_max</code>	详情请参考 <code>mindspore.ops.scatter_max()</code> 。
<code>mindspore.Tensor.scatter_min</code>	详情请参考 <code>mindspore.ops.scatter_min()</code> 。
<code>mindspore.Tensor.scatter_mul</code>	详情请参考 <code>mindspore.ops.scatter_mul()</code> 。
<code>mindspore.Tensor.scatter_sub</code>	详情请参考 <code>mindspore.ops.tensor_scatter_sub()</code> 。
<code>mindspore.Tensor.searchsorted</code>	详情请参考 <code>mindspore.ops.searchsorted()</code> 。
<code>mindspore.Tensor.select</code>	沿着选定的维度在给定的索引处进行切片。
<code>mindspore.Tensor.select_scatter</code>	详情请参考 <code>mindspore.ops.select_scatter()</code> 。
<code>mindspore.Tensor.set_const_arg</code>	指定该 <code>Tensor</code> 在作为网络入参时是否是一个常量。
<code>mindspore.Tensor.sgn</code>	详情请参考 <code>mindspore.ops.sgn()</code> 。
<code>mindspore.Tensor.shape</code>	详情请参考 <code>mindspore.ops.shape()</code> 。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.short</code>	将输入 Tensor 转换为 <code>int16</code> 类型并返回其副本, 与 <code>self.astype(mstype.int16)</code> 等价, 当 Tensor 中的值为浮点数时, 则会丢弃小数部分, 具体请参考 <code>mindspore.Tensor.astype()</code> 。
<code>mindspore.Tensor.sigmoid</code>	详情请参考 <code>mindspore.ops.sigmoid()</code> 。
<code>mindspore.Tensor.sign</code>	详情请参考 <code>mindspore.ops.sign()</code> 。
<code>mindspore.Tensor.signbit</code>	详情请参考 <code>mindspore.ops.signbit()</code> 。
<code>mindspore.Tensor.sin</code>	详情请参考 <code>mindspore.ops.sin()</code> 。
<code>mindspore.Tensor.sinc</code>	详情请参考 <code>mindspore.ops.sinc()</code> 。
<code>mindspore.Tensor.sinh</code>	详情请参考 <code>mindspore.ops.sinh()</code> 。
<code>mindspore.Tensor.size</code>	详情请参考 <code>mindspore.ops.size()</code> 。
<code>mindspore.Tensor.slice_scatter</code>	详情请参考 <code>mindspore.ops.slice_scatter()</code> 。
<code>mindspore.Tensor.slogdet</code>	详情请参考 <code>mindspore.ops.slogdet()</code> 。
<code>mindspore.Tensor.softmax</code>	详情请参考 <code>mindspore.ops.softmax()</code> 。
<code>mindspore.Tensor.sort</code>	按指定顺序对输入 Tensor 的指定维上的元素进行排序。
<code>mindspore.Tensor.split</code>	根据指定的轴将输入 Tensor 切分成块。
<code>mindspore.Tensor.sqrt</code>	详情请参考 <code>mindspore.ops.sqrt()</code> 。
<code>mindspore.Tensor.square</code>	详情请参考 <code>mindspore.ops.square()</code> 。
<code>mindspore.Tensor.squeeze</code>	详情请参考 <code>mindspore.ops.squeeze()</code> 。
<code>mindspore.Tensor.std</code>	详情请参考 <code>mindspore.ops.std()</code> 。
<code>mindspore.Tensor.storage_offset</code>	以储存元素的个数的形式返回 Tensor 在内存中的偏移量。
<code>mindspore.Tensor.stride</code>	在指定维度 <code>dim</code> 中从一个元素跳到下一个元素所必需的步长。
<code>mindspore.Tensor.strides</code>	Tensor 上每个维度跨度的字节元组。
<code>mindspore.Tensor.sub</code>	对 <code>other</code> 缩放 <code>alpha</code> 后与 <code>input</code> 相减。
<code>mindspore.Tensor.sub_</code>	详情请参考 <code>mindspore.mint.sub()</code> 。
<code>mindspore.Tensor.subtract</code>	此接口从 2.4 版本开始已被弃用, 并将在未来版本中被移除。
<code>mindspore.Tensor.sum</code>	计算指定维度上 Tensor 元素的总和。
<code>mindspore.Tensor.sum_to_size</code>	将原 Tensor 按照指定 <code>size</code> 进行求和。
<code>mindspore.Tensor.svd</code>	详情请参考 <code>mindspore.ops.svd()</code> 。
<code>mindspore.Tensor.swapaxes</code>	详情请参考 <code>mindspore.ops.swapaxes()</code> 。
<code>mindspore.Tensor.swapdims</code>	详情请参考 <code>mindspore.ops.swapdims()</code> 。
<code>mindspore.Tensor.T</code>	返回转置后的 Tensor 。
<code>mindspore.Tensor.t</code>	详情请参考 <code>mindspore.ops.t()</code> 。
<code>mindspore.Tensor.take</code>	在指定维度上获取 Tensor 中的元素。
<code>mindspore.Tensor.tan</code>	详情请参考 <code>mindspore.ops.tan()</code> 。
<code>mindspore.Tensor.tanh</code>	详情请参考 <code>mindspore.ops.tanh()</code> 。
<code>mindspore.Tensor.tensor_split</code>	详情请参考 <code>mindspore.ops.tensor_split()</code> 。
<code>mindspore.Tensor.tile</code>	按照给定的次数复制输入 Tensor 。
<code>mindspore.Tensor.to</code>	执行 Tensor 类型的转换。
<code>mindspore.Tensor.to_coo</code>	将常规 Tensor 转为稀疏化的 COO Tensor 。
<code>mindspore.Tensor.to_csr</code>	将常规 Tensor 转为稀疏化的 CSR Tensor 。
<code>mindspore.Tensor.tolist</code>	将 Tensor 转为列表, 如果输入是 Tensor scalar , 将会返回 Python 标量。
<code>mindspore.Tensor.topk</code>	沿给定维度查找 <code>k</code> 个最大或最小的元素, 并返回其值和对应的索引。
<code>mindspore.Tensor.trace</code>	在 Tensor 的对角线方向上的总和。
<code>mindspore.Tensor.transpose</code>	通过给定的维度对输入 Tensor 进行转置。

下页继续

表 2 - 续上页

<code>mindspore.Tensor.triangular_solve</code>	详情请参考 <code>mindspore.mint.triangular_solve()</code> 。
<code>mindspore.Tensor.tril</code>	详情请参考 <code>mindspore.ops.tril()</code> 。
<code>mindspore.Tensor.triu</code>	详情请参考 <code>mindspore.ops.triu()</code> 。
<code>mindspore.Tensor.true_divide</code>	<code>mindspore.Tensor.div()</code> 中 <code>rounding_mode = None</code> 的别名。
<code>mindspore.Tensor.trunc</code>	详情请参考 <code>mindspore.ops.trunc()</code> 。
<code>mindspore.Tensor.type</code>	将 <code>Tensor</code> 的 <code>dtype</code> 转换成 <code>dtype</code> 。
<code>mindspore.Tensor.type_as</code>	将第一个输入的 <code>Tensor</code> 的数据类型转换为第二个输入的 <code>Tensor</code> 的数据类型。
<code>mindspore.Tensor.unbind</code>	详情请参考 <code>mindspore.ops.unbind()</code> 。
<code>mindspore.Tensor.unfold</code>	详情请参考 <code>mindspore.ops.unfold()</code> 。
<code>mindspore.Tensor.uniform</code>	在半开区间 <code>[from_,to)</code> 内生成服从均匀分布的随机数。
<code>mindspore.Tensor.uniform_</code>	通过在半开区间 <code>[from_,to)</code> 内生成服从均匀分布的随机数来原地更新输入 <code>Tensor</code> 。
<code>mindspore.Tensor.unique</code>	对 <code>self</code> 中的元素去重。
<code>mindspore.Tensor.unique_consecutive</code>	详情请参考 <code>mindspore.ops.unique_consecutive()</code> 。
<code>mindspore.Tensor.unsorted_segment_max</code>	详情请参考 <code>mindspore.ops.unsorted_segment_max()</code> 。
<code>mindspore.Tensor.unsorted_segment_min</code>	详情请参考 <code>mindspore.ops.unsorted_segment_min()</code> 。
<code>mindspore.Tensor.unsorted_segment_prod</code>	详情请参考 <code>mindspore.ops.unsorted_segment_prod()</code> 。
<code>mindspore.Tensor.unsqueeze</code>	详情请参考 <code>mindspore.ops.unsqueeze()</code> 。
<code>mindspore.Tensor.var</code>	在指定维度上的方差。
<code>mindspore.Tensor.view</code>	根据输入 <code>shape</code> 重新创建一个 <code>Tensor</code> , 与原 <code>Tensor</code> 数据相同。
<code>mindspore.Tensor.view_as</code>	根据 <code>other</code> 的 <code>shape</code> 改变原 <code>self</code> <code>Tensor</code> 的 <code>shape</code> 。
<code>mindspore.Tensor.vsplit</code>	详情请参考 <code>mindspore.ops.vsplit()</code> 。
<code>mindspore.Tensor.where</code>	详情请参考 <code>mindspore.ops.where()</code> 。
<code>mindspore.Tensor.xdivy</code>	详情请参考 <code>mindspore.ops.xdivy()</code> 。
<code>mindspore.Tensor.xlogy</code>	详情请参考 <code>mindspore.ops.xlogy()</code> 。
<code>mindspore.Tensor.zero_</code>	返回填充 0 的 <code>Tensor</code> 。

mindspore.Tensor.abs

`mindspore.Tensor.abs()`

详情请参考 `mindspore.ops.abs()`。

mindspore.Tensor.absolute

`mindspore.Tensor.absolute()`

`mindspore.Tensor.abs()` 的别名。

mindspore.Tensor.acos

`mindspore.Tensor.acos()`

详情请参考[mindspore.ops.acos\(\)](#)。

mindspore.Tensor.acosh

`mindspore.Tensor.acosh()`

详情请参考[mindspore.ops.acosh\(\)](#)。

mindspore.Tensor.add_

`mindspore.Tensor.add_(other)`

[mindspore.Tensor.add\(\)](#) 的 in-place 版本。

mindspore.Tensor.add

`mindspore.Tensor.add(other)`

self 和 *other* 逐元素相加。

$$out_i = self_i + other_i$$

说明:

- 当 *self* 和 *other* 具有不同的 shape 时, 它们的 shape 必须要能广播为一个共同的 shape。
- *self* 和 *other* 不能同时为 bool 类型。[True, Tensor(True, bool_), Tensor(np.array([True]), bool_)] 等都为 bool 类型。
- *self* 和 *other* 遵循隐式类型转换规则, 使数据类型保持一致。
- *self* 的维度应大于或等于 1。

参数:

- **other** (Union[Tensor, number.Number, bool]) - *other* 是一个 number.Number、bool 值或数据类型为 number 或 bool_ 的 Tensor。

返回:

Tensor, shape 与输入 *self*、*other* 广播后的 shape 相同, 数据类型为两个输入中精度较高的类型。

异常:

- **TypeError** - *other* 不是 Tensor、number.Number 或 bool。

支持平台:

Ascend GPU CPU

样例:


```

>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> # case 1: x and y are both Tensor.
>>> x = Tensor(np.array([1, 2, 3]).astype(np.float32))
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> output = Tensor.add(x, y) # x.add(y)
>>> print(output)
[5. 7. 9.]
>>> # case 2: x is a scalar and y is a Tensor
>>> x = Tensor(1, mindspore.int32)
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> output = Tensor.add(x, y) # x.add(y)
>>> print(output)
[5. 6. 7.]
>>> # the data type of x is int32, the data type of y is float32,
>>> # and the output is the data format of higher precision float32.
>>> print(output.dtype)
Float32

```

`mindspore.Tensor.add(other, *, alpha=1)`

对 *other* 缩放后与 *self* 相加。

$$out_i = self_i + alpha \times other_i$$

说明:

- 当 *self* 和 *other* 的 `shape` 不同时，它们必须能够广播到一个共同的 `shape`。
- *self*、*other* 和 *alpha* 遵守隐式类型转换规则以使数据类型保持一致。

参数:

- **other** (Union[`Tensor`, `number.Number`, `bool`]) - *other* 是一个 `number.Number`、一个 `bool` 或一个数据类型为 `number` 或 `bool_` 的 `Tensor`。

关键字参数:

- **alpha** (`number.Number`) - 应用于 *other* 的缩放因子，默认值为 1。

返回:

`Tensor`，其 `shape` 与 *self*、*other* 广播后的 `shape` 相同，数据类型是 *self*、*other* 和 *alpha* 中精度更高或位数更多的类型。

异常:

- **TypeError** - 如果 *other* 或 *alpha* 不是以下之一：`Tensor`、`number.Number`、`bool`。
- **TypeError** - 如果 *alpha* 是 `float` 类型，但是 *self*、*other* 不是 `float` 类型。
- **TypeError** - 如果 *alpha* 是 `bool` 类型，但是 *self*、*other* 不是 `bool` 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(1, mindspore.int32)
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> alpha = 0.5
>>> output = Tensor.add(x, y, alpha=alpha) # x.add(y, alpha=alpha)
>>> print(output)
[3. 3.5 4.]
>>> # the data type of x is int32, the data type of y is float32,
>>> # alpha is a float, and the output is the data format of higher precision float32.
>>> print(output.dtype)
Float32
```

mindspore.Tensor.addbmm

`mindspore.Tensor.addbmm(batch1, batch2, *, beta=1, alpha=1)`

详情请参考`mindspore.ops.addbmm()`。

mindspore.Tensor.addcddiv

`mindspore.Tensor.addcddiv(tensor1, tensor2, *, value=1)`

详情请参考`mindspore.ops.addcddiv()`。

支持平台:

Ascend

mindspore.Tensor.addcmul

`mindspore.Tensor.addcmul(tensor1, tensor2, value=1)`

详情请参考`mindspore.ops.addcmul()`。

mindspore.Tensor.addmm

`mindspore.Tensor.addmm(mat1, mat2, *, beta=1, alpha=1)`

详情请参考`mindspore.ops.addmm()`。

mindspore.Tensor.addmm_

`mindspore.Tensor.addmm_(mat1, mat2, *, beta=1, alpha=1)`

`mindspore.Tensor.addmm()` 的 in-place 版本。

警告: 这是一个实验性 API, 后续可能修改或删除。

mindspore.Tensor.addmv

`mindspore.Tensor.addmv(mat, vec, *, beta=1, alpha=1)`

详情请参考 `mindspore.ops.addmv()`。

支持平台:

Ascend

mindspore.Tensor.addr

`mindspore.Tensor.addr(vec1, vec2, beta=1, alpha=1)`

详情请参考 `mindspore.ops.addr()`。

mindspore.Tensor.adjoint

`mindspore.Tensor.adjoint()`

详情请参考 `mindspore.ops.adjoint()`。

mindspore.Tensor.all

`mindspore.Tensor.all(axis=None, keep_dims=False)`

检查指定维度上是否所有元素均为 `True`。

参数:

- **axis** (`Union[int, tuple(int), list(int), Tensor]`, 可选) - 要减少的维度。如果为 `None`，减少所有维度。默认 `None`。
- **keep_dims** (`bool`, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: By default, mindspore.Tensor.all tests along all the axes.
>>> x.all()
Tensor(shape=[], dtype=Bool, value= False)
>>>
>>> # case 2: Reduces a dimension along axis 1, with keep_dims False.
>>> x.all(axis=1)
Tensor(shape=[2], dtype=Bool, value= [False,  True])
>>>
>>> # case 3: Reduces a dimension along axis (0, 1), with keep_dims False.
>>> x.all(axis=(0,1))
Tensor(shape=[], dtype=Bool, value= False)
>>>
>>> # case 4: Reduces a dimension along axis [0, 1], with keep_dims True.
```

(下页继续)

(续上页)

```
>>> x.all(axis=[0,1], keep_dims=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[False]])
```

`mindspore.Tensor.all(dim=None, keepdim=False)`

检查指定维度上是否所有元素均为 *True*。

参数:

- **dim** (Union[int, tuple(int), list(int), Tensor], 可选) - 要减少的维度。如果为 *None*，减少所有维度。默认 *None*。
- **keepdim** (bool, 可选) - 输出 tensor 是否保留维度。默认 *False*。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: By default, mindspore.Tensor.all tests along all the axes.
>>> x.all()
Tensor(shape=[], dtype=Bool, value= False)
>>>
>>> # case 2: Reduces a dimension along dim 1, with keepdim False.
>>> x.all(dim=1)
Tensor(shape=[2], dtype=Bool, value= [False, True])
>>>
>>> # case 3: Reduces a dimension along dim (0, 1), with keepdim False.
>>> x.all(dim=(0,1))
Tensor(shape=[], dtype=Bool, value= False)
>>>
>>> # case 4: Reduces a dimension along dim [0, 1], with keepdim True.
>>> x.all(dim=[0,1], keepdim=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[False]])
```

mindspore.Tensor.allclose

`mindspore.Tensor.allclose(other, rtol=1e-05, atol=1e-08, equal_nan=False)`

返回一个布尔型标量，表示 *self* 的每个元素均与 *other* 的对应元素在给定容忍度内“接近”。其中“接近”的数学公式为:

$$|self - other|atol + rtol * |other|$$

警告: 这是一个实验性 API，后续可能修改或删除。

参数:

- **other** (Tensor) - 对比的 Tensor，数据类型必须与 *self* 相同。
- **rtol** (Union[float, int, bool], 可选) - 相对容忍度。默认值：1e-05。
- **atol** (Union[float, int, bool], 可选) - 绝对容忍度。默认值：1e-08。
- **equal_nan** (bool, 可选) - 若为 True，则两个 NaN 被视为相同。默认值：False。

返回：

布尔型标量。

异常：

- **TypeError** - *self* 和 *other* 中的任何一个不是 Tensor。
- **TypeError** - *self* 和 *other* 的数据类型不在支持的类型列表中。
- **TypeError** - *atol* 和 *rtol* 中的任何一个不是 float、int 或 bool。
- **TypeError** - *equal_nan* 不是 bool。
- **TypeError** - *self* 和 *other* 的数据类型不同。
- **ValueError** - *self* 和 *other* 无法广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([1.3, 2.1, 3.2, 4.1, 5.1]), mindspore.float16)
>>> other = Tensor(np.array([1.3, 3.3, 2.3, 3.1, 5.1]), mindspore.float16)
>>> output = input.allclose(other)
>>> print(output)
False
```

mindspore.Tensor.amax

`mindspore.Tensor.amax` (*axis=None, keepdims=False, *, initial=None, where=None*)
 详情请参考 `mindspore.ops.amax()`。

mindspore.Tensor.amin

`mindspore.Tensor.amin` (*axis=None, keepdims=False, *, initial=None, where=None*)
 详情请参考 `mindspore.ops.amin()`。

mindspore.Tensor.aminmax

mindspore.Tensor.**aminmax**(* , axis=0, keepdims=False)

详情请参考[mindspore.ops.aminmax\(\)](#)。

mindspore.Tensor.any

mindspore.Tensor.**any**(axis=None, keep_dims=False)

检查指定维度上是否含有 *True*。

参数：

- **axis** (Union[int, tuple(int), list(int), Tensor], 可选) - 要减少的维度。如果为 *None*，减少所有维度。默认 *None*。
- **keep_dims** (bool, 可选) - 输出 tensor 是否保留维度。默认 *False*。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: By default, mindspore.Tensor.any tests along all the axes.
>>> x.any()
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 2: Reduces a dimension along axis 1, with keep_dims False.
>>> x.any(axis=1)
Tensor(shape=[2], dtype=Bool, value= [ True,  True])
>>>
>>> # case 3: Reduces a dimension along axis (0, 1), with keep_dims False.
>>> x.any(axis=(0,1))
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 4: Reduces a dimension along axis [0, 1], with keep_dims True.
>>> x.any(axis=[0,1], keep_dims=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[ True]])
```

mindspore.Tensor.**any**(dim=None, keepdim=False)

检查指定维度上是否含有 *True*。

参数：

- **dim** (Union[int, tuple(int), list(int), Tensor], 可选) - 要减少的维度。如果为 *None*，减少所有维度。默认 *None*。
- **keepdim** (bool, 可选) - 输出 tensor 是否保留维度。默认 *False*。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: By default, mindspore.Tensor.any tests along all the axes.
>>> x.any()
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 2: Reduces a dimension along dim 1, with keepdim False.
>>> x.any(dim=1)
Tensor(shape=[2], dtype=Bool, value= [ True,  True])
>>>
>>> # case 3: Reduces a dimension along dim (0, 1), with keepdim False.
>>> x.any(dim=(0,1))
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 4: Reduces a dimension along dim [0, 1], with keepdim True.
>>> x.any(dim=[0,1], keepdim=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[ True]])
```

mindspore.Tensor.angle

`mindspore.Tensor.angle()`

详情请参考[mindspore.ops.angle\(\)](#)。

mindspore.Tensor.approximate_equal

`mindspore.Tensor.approximate_equal(other, tolerance=1e-5)`

详情请参考[mindspore.ops.approximate_equal\(\)](#)，当前接口参数 *other* 和参考接口中的参数 *y* 实际为同一参数。

mindspore.Tensor.arccos

`mindspore.Tensor.arccos()`

[mindspore.Tensor.acos\(\)](#) 的别名。

mindspore.Tensor.arccosh

`mindspore.Tensor.arccosh()`

[mindspore.Tensor.acosh\(\)](#) 的别名。

mindspore.Tensor.arcsin

`mindspore.Tensor.arcsin()`
mindspore.Tensor.asin() 的别名。

mindspore.Tensor.arcsinh

`mindspore.Tensor.arcsinh()`
mindspore.Tensor.asinh() 的别名。

mindspore.Tensor.arctan

`mindspore.Tensor.arctan()`
mindspore.Tensor.atan() 的别名。

mindspore.Tensor.arctan2

`mindspore.Tensor.arctan2(other)`
mindspore.Tensor.atan2() 的别名。

mindspore.Tensor.arctanh

`mindspore.Tensor.arctanh()`
mindspore.Tensor.atanh() 的别名。

mindspore.Tensor.argmax

`mindspore.Tensor.argmax(axis=None, keepdims=False)`
返回 `tensor` 在指定轴上的最大值索引。

参数：

- **axis** (Union[int, None], 可选) - 指定计算轴。如果为 `None`，计算 `tensor` 中的所有元素。默认 `None`。
- **keepdims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[9, 3, 4, 5],
...                       [5, 2, 7, 4],
...                       [8, 1, 3, 6]])
>>> # case 1: By default, compute the maximum of all elements.
>>> x.argmax()
```

(下页继续)

(续上页)

```

Tensor(shape=[], dtype=Int64, value= 0)
>>>
>>> # case 2: Compute the maximum along axis 1.
>>> x.argmax(axis=1)
Tensor(shape=[3], dtype=Int64, value= [0, 2, 0])
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> x.argmax(axis=1, keepdims=True)
Tensor(shape=[3, 1], dtype=Int64, value=
[[0],
 [2],
 [0]])

```

`mindspore.Tensor.argmax (dim=None, keepdim=False)`

返回 `tensor` 在指定轴上的最大值索引。

参数:

- **dim** (Union[int, None], 可选) - 指定计算维度。如果为 `None`，计算 `tensor` 中的所有元素。默认 `None`。
- **keepdim** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

`Tensor`

支持平台:

Ascend

样例:

```

>>> import mindspore
>>> x = mindspore.tensor([[9, 3, 4, 5],
...                       [5, 2, 7, 4],
...                       [8, 1, 3, 6]])
>>> # case 1: By default, compute the maximum of all elements.
>>> x.argmax()
Tensor(shape=[], dtype=Int64, value= 0)
>>>
>>> # case 2: Compute the maximum along dim 1.
>>> x.argmax(dim=1)
Tensor(shape=[3], dtype=Int64, value= [0, 2, 0])
>>>
>>> # case 3: If keepdim=True, the output shape will be same of that of the input.
>>> x.argmax(dim=1, keepdim=True)
Tensor(shape=[3, 1], dtype=Int64, value=
[[0],
 [2],
 [0]])

```

mindspore.Tensor.argmax_with_value

`mindspore.Tensor.argmax_with_value (axis=0, keep_dims=False)`

返回 `tensor` 在指定轴上的最大值及其索引。

参数:

- **axis** (Union[int, None], 可选) - 指定计算轴。如果为 None，计算 `tensor` 中的所有元素。默认 0。
- **keep_dims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 False。

返回:

两个 `tensor` 组成的 tuple(max, max_indices)。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[9, 3, 4, 5],
...                       [5, 2, 7, 4],
...                       [8, 1, 3, 6]])
>>> # case 1: By default, compute the maximum along axis 0.
>>> x.argmax_with_value()
(Tensor(shape=[4], dtype=Int64, value= [9, 3, 7, 6]),
 Tensor(shape=[4], dtype=Int64, value= [0, 0, 1, 2]))
>>>
>>> # case 2: Compute the maximum along axis 1.
>>> x.argmax_with_value(axis=1)
(Tensor(shape=[3], dtype=Int64, value= [9, 7, 8]),
 Tensor(shape=[3], dtype=Int64, value= [0, 2, 0]))
>>>
>>> # case 3: If keep_dims=True, the output shape will be same of that of the input.
>>> x.argmax_with_value(axis=1, keep_dims=True)
(Tensor(shape=[3, 1], dtype=Int64, value=
 [[9],
 [7],
 [8]]),
 Tensor(shape=[3, 1], dtype=Int64, value=
 [[0],
 [2],
 [0]]))
>>>
>>> # case 4: If axis=None, compute the maximum of all elements.
>>> x.argmax_with_value(axis=None, keep_dims=True)
(Tensor(shape=[], dtype=Int64, value= 9),
 Tensor(shape=[], dtype=Int64, value= 0))
```


mindspore.Tensor.argmax

`mindspore.Tensor.argmax (axis=None, keepdims=False)`

返回 `tensor` 在指定轴上的最小值索引。

参数:

- **axis** (Union[int, None], 可选) - 指定计算轴。如果为 `None`，计算 `tensor` 中的所有元素。默认 `None`。
- **keepdims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[2, 5, 1, 6],
...                       [3, -7, -2, 4],
...                       [8, -4, 1, -3]])
>>> # case 1: By default, compute the minimum of all elements.
>>> x.argmax()
Tensor(shape=[], dtype=Int32, value= 5)
>>>
>>> # case 2: Compute the minimum along axis 1.
>>> x.argmax(axis=1)
Tensor(shape=[3], dtype=Int32, value= [2, 1, 1])
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> x.argmax(axis=1, keepdims=True)
Tensor(shape=[3, 1], dtype=Int32, value=
[[2],
 [1],
 [1]])
```

`mindspore.Tensor.argmax (dim=None, keepdim=False)`

返回 `tensor` 在指定轴上的最小值索引。

参数:

- **dim** (Union[int, None], 可选) - 指定计算维度。如果为 `None`，计算 `tensor` 中的所有元素。默认 `None`。
- **keepdim** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[2, 5, 1, 6],
...                       [3, -7, -2, 4],
```

(下页继续)

(续上页)

```

...           [8, -4, 1, -3]])
>>> # case 1: By default, compute the minimum of all elements.
>>> x.argmaxin()
Tensor(shape=[], dtype=Int32, value= 5)
>>>
>>> # case 2: Compute the minimum along dim 1.
>>> x.argmaxin(dim=1)
Tensor(shape=[3], dtype=Int32, value= [2, 1, 1])
>>>
>>> # case 3: If keepdim=True, the output shape will be same of that of the input.
>>> x.argmaxin(dim=1, keepdim=True)
Tensor(shape=[3, 1], dtype=Int32, value=
[[2],
 [1],
 [1]])

```

mindspore.Tensor.argmaxin_with_value

`mindspore.Tensor.argmaxin_with_value` (*axis=0, keep_dims=False*)

返回 `tensor` 在指定轴上的最小值及其索引。

参数:

- **axis** (Union[int, None], 可选) - 指定计算轴。如果为 None，计算 `tensor` 中的所有元素。默认 0。
- **keep_dims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 False。

返回:

两个 `tensor` 组成的 `tuple`(min, min_indices)。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> x = mindspore.tensor([[2, 5, 1, 6],
...                       [3, -7, -2, 4],
...                       [8, -4, 1, -3]])
>>> # case 1: By default, compute the minimum along axis 0.
>>> x.argmaxin_with_value()
(Tensor(shape=[4], dtype=Int64, value= [ 2, -7, -2, -3]),
 Tensor(shape=[4], dtype=Int64, value= [0, 1, 1, 2]))
>>>
>>> # case 2: Compute the minimum along axis 1.
>>> x.argmaxin_with_value(axis=1)
(Tensor(shape=[3], dtype=Int64, value= [ 1, -7, -4]),
 Tensor(shape=[3], dtype=Int64, value= [2, 1, 1]))
>>>
>>> # case 3: If keep_dims=True, the output shape will be same of that of the input.
>>> x.argmaxin_with_value(axis=1, keep_dims=True)
(Tensor(shape=[3, 1], dtype=Int64, value=
[[ 1],
 [-7],
 [-4]]),

```

(下页继续)

(续上页)

```

Tensor(shape=[3, 1], dtype=Int64, value=
[[2],
 [1],
 [1]]))
>>>
>>> # case 4: If axis=None, compute the minimum of all elements.
>>> x.argmax_with_value(axis=None, keep_dims=True)
(Tensor(shape=[], dtype=Int64, value= -7),
 Tensor(shape=[], dtype=Int64, value= 0))

```

mindspore.Tensor.argsort

`mindspore.Tensor.argsort (axis=-1, descending=False)`

按指定顺序对 *self* 沿给定维度进行排序，并返回排序后的索引。

参数：

- **axis** (int, 可选) - 指定排序的轴。默认值：-1，表示指定最后一维。当前 Ascend 后端只支持对最后一维进行排序。
- **descending** (bool, 可选) - 输出顺序。如果 *descending* 为 True，按照元素值降序排序，否则升序排序。默认值：False。

返回：

Tensor，排序后 *self* 的索引。数据类型为 int32。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[8, 2, 1], [5, 9, 3], [4, 6, 7]]), mindspore.float16)
>>> sort = Tensor.argsort(x) # x.argsort()
>>> print(sort)
[[2 1 0]
 [2 0 1]
 [0 1 2]]

```

`mindspore.Tensor.argsort (dim=-1, descending=False, stable=False)`

按指定顺序对 *self* 沿给定维度进行排序，并返回排序后的索引。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **dim** (int, 可选) - 指定排序的轴。默认值：-1，表示指定最后一维。当前 Ascend 后端只支持对最后一维进行排序。
- **descending** (bool, 可选) - 输出顺序。如果 *descending* 为 True，按照元素值降序排序，否则升序排序。默认值：False。

- **stable** (bool, 可选) - 是否使用稳定排序算法。默认值: `False`。

返回:

Tensor, 排序后 *self* 的索引。数据类型为 `int64`。

异常:

- **ValueError** - 如果 *dim* 的设定值超出了范围。
- **TypeError** - 如果 *dim* 的数据类型不是 `int32`。
- **TypeError** - 如果 *descending* 的数据类型不是 `bool`。
- **TypeError** - 如果 *stable* 的数据类型不是 `bool`。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[8, 2, 1], [5, 9, 3], [4, 6, 7]]), mindspore.float16)
>>> sort = Tensor.argsort(x) # x.argsort()
>>> print(sort)
[[2 1 0]
 [2 0 1]
 [0 1 2]]
```

mindspore.Tensor.argwhere

`mindspore.Tensor.argwhere()`

详情请参考 `mindspore.ops.argwhere()`。

mindspore.Tensor.asin

`mindspore.Tensor.asin()`

详情请参考 `mindspore.ops.asin()`。

mindspore.Tensor.asinh

`mindspore.Tensor.asinh()`

详情请参考 `mindspore.ops.asinh()`。

`mindspore.Tensor.asnumpy`

`mindspore.Tensor.asnumpy()`

将张量转换为 NumPy 数组。该方法会将 Tensor 本身转换为 NumPy 的 ndarray。这个 Tensor 和函数返回的 ndarray 共享内存地址。对 Tensor 本身的修改会反映到相应的 ndarray 上。

返回：

NumPy 的 ndarray，该 ndarray 与 Tensor 共享内存地址。

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([1, 2], dtype=np.float32))
>>> y = x.asnumpy()
>>> y[0] = 11
>>> print(x)
[11.  2.]
>>> print(y)
[11.  2.]
```

`mindspore.Tensor.assign_value`

`mindspore.Tensor.assign_value(value)`

将另一个 Tensor 的值赋给当前 Tensor。

参数：

- **value** (Tensor) - 用于赋值的 Tensor。

返回：

Tensor，赋值后的 Tensor。

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor([1, 2, 3, 4])
>>> y = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.assign_value(y)
>>> print(x)
[[1 2]
 [3 4]]
```

`mindspore.Tensor.astype`

`mindspore.Tensor.astype(dtype, copy=True)`

将 Tensor 转为指定数据类型，可指定是否返回副本。

参数：

- **dtype** (Union[mindspore.dtype, numpy.dtype, str]) - 指定的 Tensor 数据类型，可以是：`mindspore.dtype.float32`，`numpy.float32` 或 `float32` 的格式。
- **copy** (bool, 可选) - 默认情况下，`astype` 返回新拷贝的 Tensor。如果该参数设为 `False`，则返回输入 Tensor 而不是副本。默认值：`True`。

返回：

Tensor，指定数据类型的 Tensor。

异常：

- **TypeError** - 指定了无法解析的类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.ones((1,2,2,1), dtype=np.float32))
>>> x = x.astype("int32")
>>> print(x.dtype)
Int32
```

mindspore.Tensor.atan

`mindspore.Tensor.atan()`

详情请参考[`mindspore.ops.atan\(\)`](#)。

mindspore.Tensor.atan2

`mindspore.Tensor.atan2(other)`

详情请参考[`mindspore.ops.atan2\(\)`](#)。

mindspore.Tensor.atanh

`mindspore.Tensor.atanh()`

详情请参考[`mindspore.ops.atanh\(\)`](#)。

mindspore.Tensor.baddbmm

`mindspore.Tensor.baddbmm(batch1, batch2, *, beta=1, alpha=1)`

详情请参考[`mindspore.ops.baddbmm\(\)`](#)。

mindspore.Tensor.bernoulli

`mindspore.Tensor.bernoulli(*, generator=None)`

详情请参考[`mindspore.mint.bernoulli\(\)`](#)。

mindspore.Tensor.bfloat16

`mindspore.Tensor.bfloat16()`
将输入 `Tensor` 转换为 `bfloat16` 类型。

返回：

`Tensor`，其数据类型为 `bfloat16`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.int32)
>>> output = input_x.bfloat16()
>>> print(output.dtype)
BFloat16
```

mindspore.Tensor.bincount

`mindspore.Tensor.bincount(weights=None, minlength=0)`

详情请参考 `mindspore.ops.bincount()`。

mindspore.Tensor.bitwise_and

`mindspore.Tensor.bitwise_and(other)`

逐元素执行两个 `Tensor` 的与运算。

说明：参数 `self` 和 `other` 遵循隐式类型转换规则，使数据类型保持一致。

参数：

- **other** (`Tensor`, `Number.number`) - 输入 `Tensor` 或常量，`shape` 与 `self` 相同，或能与 `self` 的 `shape` 广播。

返回：

`Tensor`，与广播后的输入 `shape` 相同，和 `self` 数据类型相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> other = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> output = input.bitwise_and(other)
```

(下页继续)

```
>>> print(output)
[ 0  0  1 -1  1  0  1]
```

mindspore.Tensor.bitwise_not

`mindspore.Tensor.bitwise_not()`

逐元素对当前 Tensor 进行位取反。

警告：这是一个实验性 API，后续可能修改或删除。

返回：

Tensor，其数据类型和 shape 与 *self* 相同。

异常：

- **TypeError** - 如果 *self* 不是 Tensor。
- **RuntimeError** - 如果 *self* 的数据类型不是 int 或 bool。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([True, False, True, False]))
>>> output = input.bitwise_not()
>>> print(output)
[False True False True]
```

mindspore.Tensor.bitwise_left_shift

`mindspore.Tensor.bitwise_left_shift(other)`

详情请参考 `mindspore.ops.bitwise_left_shift()`。

mindspore.Tensor.bitwise_or

`mindspore.Tensor.bitwise_or(other)`

逐元素执行两个 Tensor 的或运算。

说明：参数 *self* 和 *other* 遵循隐式类型转换规则，使数据类型保持一致。

参数：

- **other** (Tensor, Number.number) - 输入 Tensor 或常量，shape 与 *self* 相同，或能与 *self* 的 shape 广播。

返回：

Tensor，与广播后的输入 shape 相同，和 *self* 数据类型相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> other = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> output = input.bitwise_or(other)
>>> print(output)
[ 0  1  1 -1 -1  3  3]
```

mindspore.Tensor.bitwise_right_shift

`mindspore.Tensor.bitwise_right_shift(other)`

详情请参考[mindspore.ops.bitwise_right_shift\(\)](#)。

mindspore.Tensor.bitwise_xor

`mindspore.Tensor.bitwise_xor(other)`

逐元素执行两个 Tensor 的异或运算。

说明：参数 *self* 和 *other* 遵循隐式类型转换规则，使数据类型保持一致。

参数：

- **other** (Tensor, Number.number) - 输入 Tensor 或常量，shape 与 *self* 相同，或能与 *self* 的 shape 广播。

返回：

Tensor，与广播后的输入 shape 相同，和 *self* 数据类型相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> other = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> output = input.bitwise_xor(other)
>>> print(output)
[ 0  1  0  0 -2  3  2]
```

mindspore.Tensor.bmm

`mindspore.Tensor.bmm(mat2)`

详情请参考[`mindspore.ops.bmm\(\)`](#)。

mindspore.Tensor.bool

`mindspore.Tensor.bool()`

将输入 `Tensor` 转换为 `bool` 类型，其中 `Tensor` 中的值为 0 时，则为 `False`；非 0 时，则为 `True`。

返回：

`Tensor`，其数据类型为 `bool`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.float32)
>>> output = input_x.bool()
>>> print(output.dtype)
Bool
```

mindspore.Tensor.broadcast_to

`mindspore.Tensor.broadcast_to(shape)`

详情请参考[`mindspore.ops.broadcast\_to\(\)`](#)。

mindspore.Tensor.byte

`mindspore.Tensor.byte()`

将输入 `Tensor` 转换为 `uint8` 类型。

返回：

`Tensor`，其数据类型为 `uint8`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.float32)
>>> output = input_x.byte()
>>> print(output.dtype)
UInt8
```

mindspore.Tensor.cauchy

`mindspore.Tensor.cauchy` (*median=0.0, sigma=1.0*)

使用柯西分布生成的数值填充当前 Tensor。

$$f(x) = \frac{1}{\pi} \frac{\sigma}{(x - median)^2 + \sigma^2}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **median** (float, 可选) - 柯西分布中定位分布峰值位置的位置参数。默认值：0.0。
- **sigma** (float, 可选) - 柯西分布中最大值一半处的一半宽度的尺度参数。默认值：1.0。

返回：

Tensor，具有与当前 Tensor 相同的 shape 和 dtype。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> x = mindspore.Tensor(np.zeros((1, 2)), dtype=mindspore.float32)
>>> x.cauchy()
Tensor(shape=[1, 2], dtype=Float32, value=
[[8.79836142e-01, 9.37541723e-01]])
```

mindspore.Tensor.ceil

`mindspore.Tensor.ceil()`

详情请参考[`mindspore.ops.ceil\(\)`](#)。

mindspore.Tensor.cholesky

`mindspore.Tensor.cholesky` (*upper=False*)

详情请参考[`mindspore.ops.cholesky\(\)`](#)。

mindspore.Tensor.cholesky_solve

`mindspore.Tensor.cholesky_solve` (*input2, upper=False*)

详情请参考[`mindspore.ops.cholesky\_solve\(\)`](#)。

警告：这是一个实验性 API，后续可能修改或删除。

mindspore.Tensor.choose

`mindspore.Tensor.choose` (*choices*, *mode*='clip')

根据原始 `Tensor` 数组和一个索引数组构造一个新的 `Tensor`。

参数：

- **choices** (`Union[tuple, list, Tensor]`) - 索引选择数组。原始输入 `Tensor` 和 *choices* 的广播维度必须相同。如果 *choices* 本身是一个 `Tensor`，则其最外层的维度（即对应于第零维的维度）将被用来定义 *choices* 数组。
- **mode** (`str`, 可选) - 指定如何处理 $[0, n-1]$ 外部的索引。支持 'raise'、'wrap'、'clip'。
 - `raise`：引发异常（默认）。
 - `wrap`：原值映射为对 `n` 取余后的值。
 - `clip`：大于 `n-1` 的值会被映射为 `n-1`。该模式下禁用负数索引。

默认值：'clip'。

返回：

`Tensor`，合并后的结果。

异常：

- **ValueError** - 输入 `Tensor` 和任一 *choices* 无法广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> choices = [[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23], [30, 31, 32, 33]]
>>> x = Tensor(np.array([2, 3, 1, 0]))
>>> print(x.choose(choices))
[20 31 12  3]
```

mindspore.Tensor.chunk

`mindspore.Tensor.chunk` (*chunks*, *dim*=0)

沿着指定轴 *dim* 将输入 `Tensor` 切分成 *chunks* 个 sub-tensor。

说明：此函数返回的 sub-tensor 数量可能小于通过 *chunks* 指定的 sub-tensor 数量。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **chunks** (`int`) - 要切分的 sub-tensor 数量。
- **dim** (`int`, 可选) - 指定需要分割的维度。默认值：0。

返回：

`tuple[Tensor]`。

异常:

- **TypeError** - *dim* 不是 int 类型。
- **TypeError** - *chunks* 不是 int。
- **ValueError** - 参数 *dim* 超出 $[-self.ndim, self.ndim)$ 范围。
- **ValueError** - 参数 *chunks* 不是正数。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.arange(9).astype("float32"))
>>> output = input_x.chunk(3, dim=0)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  1.00000000e+00,  2.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 3.00000000e+00,  4.00000000e+00,  5.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 6.00000000e+00,  7.00000000e+00,  8.
↪00000000e+00]))
```

`mindspore.Tensor.chunk(chunks, axis=0)`

沿着指定轴 *axis* 将输入 Tensor 切分成 *chunks* 个 sub-tensor。

说明: 此函数返回的数量可能小于通过 *chunks* 指定的数量!

参数:

- **chunks** (int) - 要切分的 sub-tensor 数量。
- **axis** (int, 可选) - 指定需要分割的维度。默认值: 0。

返回:

tuple[Tensor]。

异常:

- **TypeError** - *axis* 不是 int 类型。
- **TypeError** - *chunks* 不是 int。
- **ValueError** - 参数 *axis* 超出 $[-self.ndim, self.ndim)$ 范围。
- **ValueError** - 参数 *chunks* 不是正数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = Tensor(np.arange(9).astype("float32"))
>>> output = input_x.chunk(3, axis=0)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  1.00000000e+00,  2.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 3.00000000e+00,  4.00000000e+00,  5.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 6.00000000e+00,  7.00000000e+00,  8.
↪00000000e+00]))
```

mindspore.Tensor.clamp

`mindspore.Tensor.clamp(min=None, max=None)`
详情请参考 `mindspore.ops.clamp()`。

mindspore.Tensor.clamp_

`mindspore.Tensor.clamp_(min=None, max=None)`
`mindspore.Tensor.clamp()` 的 in-place 版本。

警告：这是一个实验性 API，后续可能修改或删除。

mindspore.Tensor.clip

`mindspore.Tensor.clip(min=None, max=None)`
`mindspore.Tensor.clamp()` 的别名。

mindspore.Tensor.clone

`mindspore.Tensor.clone()`
返回一个当前 `Tensor` 的副本。

警告：这是一个实验性 API，后续可能修改或删除。

说明：该函数是可微分的，梯度将直接从该函数的计算结果流向 *self*。

返回：

`Tensor`，其数据、shape 和数据类型与输入 *self* 相同。

支持平台：

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.ones((3,3)).astype("float32"))
>>> output = input.clone()
>>> print(output)
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

mindspore.Tensor.col2im

`mindspore.Tensor.col2im(output_size, kernel_size, dilation, padding_value, stride)`

详情请参考 `mindspore.ops.col2im()`。

mindspore.Tensor.conj

`mindspore.Tensor.conj()`

详情请参考 `mindspore.ops.conj()`。

mindspore.Tensor.contiguous

`mindspore.Tensor.contiguous()`

将 Tensor 转成一个连续内存的 Tensor, 该 Tensor 包含与原 Tensor 相同的数据。

返回:

一个连续内存的 Tensor, 该 Tensor 包含与原 Tensor 相同的数据。

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor([[1, 2, 3], [4, 5, 6]], dtype=ms.float32)
>>> y = ops.transpose(x, (1, 0))
>>> z = y.contiguous()
>>> print(z.is_contiguous())
True
```

mindspore.Tensor.copy

`mindspore.Tensor.copy()`

复制一个 Tensor 并返回。

说明: 当前实现不支持类似 NumPy 的 `order` 参数。

返回:

复制的 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> a = Tensor(np.ones((3,3)).astype("float32"))
>>> output = a.copy()
>>> print(output)
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

mindspore.Tensor.copy_

`mindspore.Tensor.copy_(src, non_blocking=False)`

将 *src* 中的元素复制到 *Tensor* 中并返回这个 *Tensor*。

警告: 这是一个实验性 API, 后续可能修改或删除。*src* 需可广播为 *self*。*src* 可以是不同的数据类型。

参数:

- **src** (*Tensor*) - 用于复制的 *Tensor*。
- **non_blocking** (*bool*, 可选) - 当前无作用。默认值为 *False*。

返回:

Tensor, 赋值后的 *Tensor*。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> a = Tensor(np.ones((3, 3)).astype("float32"))
>>> b = Tensor(np.zeros((3, 3)).astype("float32"))
>>> a.copy_(b)
>>> print(a)
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```


`mindspore.Tensor.copysign`

`mindspore.Tensor.copysign(other)`

详情请参考`mindspore.ops.copysign()`。

`mindspore.Tensor.cos`

`mindspore.Tensor.cos()`

详情请参考`mindspore.ops.cos()`。

`mindspore.Tensor.cosh`

`mindspore.Tensor.cosh()`

详情请参考`mindspore.ops.cosh()`。

`mindspore.Tensor.count_nonzero`

`mindspore.Tensor.count_nonzero(dim=None)`

计算输入 `Tensor` 指定轴上的非零元素的数量。如果没有指定维度，则计算 `Tensor` 中所有非零元素的数量。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **dim** (`Union[None, int, tuple(int), list(int)]`, 可选) - 要沿其计算非零值数量的维度。默认值：None，计算所有非零元素的个数。

返回：

`Tensor`，指定的轴上非零元素的数量。

异常：

- **TypeError** - `dim` 类型不是 `int`、`tuple(int)`、`list(int)` 或 `None`。
- **ValueError** - `dim` 取值不在 `[-self.ndim, self.ndim)` 范围。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> import mindspore
>>> # case 1: each value specified.
>>> x = Tensor(np.array([[0, 1, 0], [1, 1, 0]]).astype(np.float32))
>>> nonzero_num = x.count_nonzero(dim=[0, 1])
>>> print(nonzero_num)
[[3]]
>>> # case 2: all value is default.
>>> nonzero_num = x.count_nonzero()
```

(下页继续)

(续上页)

```
>>> print (nonzero_num)
3
>>> # case 3: dim value was specified 0.
>>> nonzero_num = x.count_nonzero(dim=[0,])
>>> print (nonzero_num)
[1 2 0]
>>> # case 4: dim value was specified 1.
>>> nonzero_num = x.count_nonzero(dim=[1,])
>>> print (nonzero_num)
[1 2]
```

`mindspore.Tensor.count_nonzero` (*axis=()*, *keep_dims=False*, *dtype=None*)

计算输入 `Tensor` 指定轴上的非零元素的数量。如果没有指定维度，则计算 `Tensor` 中所有非零元素的数量。

参数：

- **axis** (`Union[int, tuple(int), list(int)]`), 可选 - 要沿其计算非零值数量的维度。默认值：(), 计算所有非零元素的个数。
- **keep_dims** (`bool`, 可选) - 是否保留 *axis* 指定的维度。如果为 `True`，保留对应维度 `size` 为 1；如果为 `False`，不保留对应维度。默认值：`False`。
- **dtype** (`Union[Number, mindspore.bool_]`), 可选 - 输出 `Tensor` 的数据类型。默认值：`None`。

返回：

`Tensor`, *axis* 指定的轴上非零元素的数量。数据类型由 *dtype* 指定。

异常：

- **TypeError** - *axis* 不是 `int`、`tuple` 或者 `list`。
- **ValueError** - 如果 *axis* 中的任何值不在 `[-self.ndim, self.ndim)` 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> import mindspore
>>> # case 1: each value specified.
>>> x = Tensor(np.array([[0, 1, 0], [1, 1, 0]]).astype(np.float32))
>>> nonzero_num = x.count_nonzero(x=x, axis=[0, 1], keep_dims=True, dtype=mindspore.int32)
>>> print (nonzero_num)
[[3]]
>>> # case 2: all value is default.
>>> nonzero_num = x.count_nonzero()
>>> print (nonzero_num)
3
>>> # case 3: axis value was specified 0.
>>> nonzero_num = x.count_nonzero(axis=[0,])
>>> print (nonzero_num)
[1 2 0]
>>> # case 4: axis value was specified 1.
>>> nonzero_num = x.count_nonzero(axis=[1,])
>>> print (nonzero_num)
[1 2]
```

(下页继续)

(续上页)

```
>>> # case 5: keep_dims value was specified.
>>> nonzero_num = x.count_nonzero(keep_dims=True)
>>> print(nonzero_num)
[[3]]
>>> # case 6: keep_dims and axis value was specified.
>>> nonzero_num = x.count_nonzero(axis=[0,], keep_dims=True)
>>> print(nonzero_num)
[[1 2 0]]
```

mindspore.Tensor.cov

`mindspore.Tensor.cov(*, correction=1, fweights=None, aweights=None)`

详情请参考 `mindspore.ops.cov()`。

mindspore.Tensor.cross

`mindspore.Tensor.cross(other, dim=None)`

详情请参考 `mindspore.ops.cross()`。

mindspore.Tensor.cummax

`mindspore.Tensor.cummax(axis)`

详情请参考 `mindspore.ops.cummax()`。

mindspore.Tensor.cummin

`mindspore.Tensor.cummin(axis)`

详情请参考 `mindspore.ops.cummin()`。

mindspore.Tensor.cumprod

`mindspore.Tensor.cumprod(dim, dtype=None)`

详情请参考 `mindspore.ops.cumprod()`。

mindspore.Tensor.cumsum

`mindspore.Tensor.cumsum(dim, *, dtype=None)`

计算输入 Tensor *self* 沿轴 *dim* 的累积和。

$$y_i = x_1 + x_2 + x_3 + \dots + x_i$$

参数：

- **dim** (int) - 累积和计算的轴。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 输出数据类型。如果不为 **None**, 则输入会转化为 *dtype*。这有利于防止数值溢出。如果为 **None**, 则输出和输入的数据类型一致。默认值: **None**。

返回:

Tensor, 和输入 Tensor 的 shape 相同。

异常:

- **ValueError** - 如果 *dim* 超出范围。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[3, 4, 6, 10], [1, 6, 7, 9], [4, 3, 8, 7], [1, 3, 7, 9]]).astype(np.
    ↳ float32))
>>> # case 1: along the dim 0
>>> y = x.cumsum(dim=0)
>>> print(y)
[[ 3.  4.  6. 10.]
 [ 4. 10. 13. 19.]
 [ 8. 13. 21. 26.]
 [ 9. 16. 28. 35.]]
>>> # case 2: along the dim 1
>>> y = x.cumsum(dim=1)
>>> print(y)
[[ 3.  7. 13. 23.]
 [ 1.  7. 14. 23.]
 [ 4.  7. 15. 22.]
 [ 1.  4. 11. 20.]]
```

`mindspore.Tensor.cumsum(axis=None, dtype=None)`

计算输入 Tensor *self* 沿轴 *axis* 的累积和。

$$y_i = x_1 + x_2 + x_3 + \dots + x_i$$

说明: 目前 Ascend 平台上, 对于静态 shape 的场景, *self* 的数据类型暂仅支持: int8、uint8、int32, float32 和 float16; 对于动态 shape 的场景, *self* 的数据类型暂仅支持: int32、float32 和 float16。

参数:

- **axis** (int) - 累积和计算的轴。
- **dtype** (*mindspore.dtype*, 可选) - 输出数据类型。如果不为 **None**, 则输入会转化为 *dtype*。这有利于防止数值溢出。如果为 **None**, 则输出和输入的数据类型一致。默认值: **None**。

返回:

Tensor, 和输入 Tensor 的 shape 相同。

异常:

- **ValueError** - 如果 *axis* 超出范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[3, 4, 6, 10], [1, 6, 7, 9], [4, 3, 8, 7], [1, 3, 7, 9]]).astype(np.
↳float32))
>>> # case 1: along the axis 0
>>> y = x.cumsum(axis=0)
>>> print(y)
[[ 3.  4.  6. 10.]
 [ 4. 10. 13. 19.]
 [ 8. 13. 21. 26.]
 [ 9. 16. 28. 35.]]
>>> # case 2: along the axis 1
>>> y = x.cumsum(axis=1)
>>> print(y)
[[ 3.  7. 13. 23.]
 [ 1.  7. 14. 23.]
 [ 4.  7. 15. 22.]
 [ 1.  4. 11. 20.]]
```

mindspore.Tensor.deg2rad

`mindspore.Tensor.deg2rad()`

详情请参考[mindspore.ops.deg2rad\(\)](#)。

mindspore.Tensor.diag

`mindspore.Tensor.diag()`

详情请参考[mindspore.ops.diag\(\)](#)。

`mindspore.Tensor.diag(diagonal=0)`

如果 *input* 是向量 (1-D 张量), 则返回一个二维张量, 其中 *input* 的元素作为对角线。

如果 *input* 是矩阵 (2-D 张量), 则返回具有 *input* 对角线元素的 1-D 张量。

参数 *diagonal* 控制要考虑的对角线:

- 如果 *diagonal* = 0, 则它是主对角线。
- 如果 *diagonal* > 0, 则它位于主对角线的上方。
- 如果 *diagonal* < 0, 则它位于主对角线下方。

警告:

- 这是一个实验性 API, 后续可能修改或删除。
- 图模式和 CPU、GPU 后端不支持 *diagonal* 参数取非零值。

参数:

- **diagonal** (int, 可选) - 输入 Tensor, 默认值为 0。

返回:

Tensor, 具有与输入 Tensor 相同的数据类型, 其 shape 由 *diagonal* 决定:

- 如果输入 *input* 的 shape 为 (x_0) : 输出 shape 为 $(x_0 + |diagonal|, x_0 + |diagonal|)$ 的二维张量。
- 如果输入 *input* 的 shape 为 (x_0, x_1) : 输出 shape 为主对角线上下平移 $(|diagonal|)$ 个单位后所剩元素的长度的一维张量。

异常:

- **ValueError** - *input* 的 shape 不是 1-D 和 2-D。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor
>>> input = Tensor([1, 2, 3, 4]).astype('int32')
>>> output = input.diag()
>>> print(output)
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

mindspore.Tensor.diagflat

mindspore.Tensor.**diagflat** (*offset=0*)

详情请参考[mindspore.ops.diagflat\(\)](#)。

mindspore.Tensor.diagonal

mindspore.Tensor.**diagonal** (*offset=0, axis1=0, axis2=1*)

详情请参考[mindspore.ops.diagonal\(\)](#)。当前接口的参数 *axis1* 和参考接口中的参数 *dim1* 实际为同一参数, 当前接口的参数 *axis2* 和参考接口中的参数 *dim2* 实际为同一参数。

mindspore.Tensor.diagonal_scatter

mindspore.Tensor.**diagonal_scatter** (*src, offset=0, dim1=0, dim2=1*)

详情请参考[mindspore.ops.diagonal_scatter\(\)](#)。

mindspore.Tensor.diff

`mindspore.Tensor.diff (n=1, axis=-1, prepend=None, append=None)`

详情请参考`mindspore.ops.diff()`。

mindspore.Tensor.digamma

`mindspore.Tensor.digamma()`

详情请参考`mindspore.ops.digamma()`。

mindspore.Tensor.div

`mindspore.Tensor.div (other, *, rounding_mode=None)`

详情请参考`mindspore.ops.div()`。

mindspore.Tensor.div_

`mindspore.Tensor.div_ (other, *, rounding_mode=None)`

`mindspore.Tensor.div()` 的 in-place 版本。

mindspore.Tensor.divide

`mindspore.Tensor.divide (other, *, rounding_mode=None)`

`mindspore.Tensor.div()` 的别名。

mindspore.Tensor.dot

`mindspore.Tensor.dot (other)`

计算两个 1DTensor 的点积。

参数：

- **other** (Tensor) - 点积的输入，须为 1D。

返回：

Tensor, shape 是 [], 类型与 *self* 一致。

异常：

- **TypeError** - *other* 的数据类型不是 tensor。
- **RuntimeError** - *self* 和 *other* 的数据类型不一致。
- **RuntimeError** - *self* 和 *other* 的 shape 不一致。
- **RuntimeError** - *self* 或 *other* 不是 1D。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> input = Tensor([2.0, 3.0], mindspore.float32)
>>> other = Tensor([2.0, 1.0], mindspore.float32)
>>> output = Tensor.dot(input, other) # input.dot(other)
>>> print(output)
[7.    ]
>>> print(output.dtype)
Float32
```

mindspore.Tensor.double

`mindspore.Tensor.double()`

将输入 `Tensor` 转换为 *float64* 类型。

返回:

`Tensor`, 其数据类型为 *float64*。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.int32)
>>> output = input_x.double()
>>> print(output.dtype)
Float64
```

mindspore.Tensor.dsplitt

`mindspore.Tensor.dsplitt(indices_or_sections)`

详情请参考 *mindspore.ops.dsplitt()*。

mindspore.Tensor.dtype

property `mindspore.Tensor.dtype`

返回 `Tensor` 的数据类型 (*mindspore.dtype*)。

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([1, 2], dtype=np.float32))
>>> print(x.dtype)
Float32
```


mindspore.Tensor.eigvals

`mindspore.Tensor.eigvals()`

详情请参考[`mindspore.ops.eigvals\(\)`](#)。

警告：这是一个实验性 API，后续可能修改或删除。

mindspore.Tensor.eq

`mindspore.Tensor.eq(other)`

详情请参考[`mindspore.ops.eq\(\)`](#)。

mindspore.Tensor.equal

`mindspore.Tensor.equal(other)`

详情请参考[`mindspore.ops.equal\(\)`](#)。

mindspore.Tensor.erf

`mindspore.Tensor.erf()`

详情请参考[`mindspore.ops.erf\(\)`](#)。

mindspore.Tensor.erfc

`mindspore.Tensor.erfc()`

详情请参考[`mindspore.ops.erfc\(\)`](#)。

mindspore.Tensor.erfinv

`mindspore.Tensor.erfinv()`

详情请参考[`mindspore.ops.erfinv\(\)`](#)。

mindspore.Tensor.exp

`mindspore.Tensor.exp()`

详情请参考[`mindspore.ops.exp\(\)`](#)。

mindspore.Tensor.exp_

`mindspore.Tensor.exp_()`
 原地更新版本的`mindspore.Tensor.exp()`。

警告：这是一个实验性 API，后续可能修改或删除。

mindspore.Tensor.expand

`mindspore.Tensor.expand(size)`
 详情请参考`mindspore.ops.broadcast_to()`，当前接口的参数 *size* 和参考接口中的参数 *shape* 实际为同一参数。

mindspore.Tensor.expand_as

`mindspore.Tensor.expand_as(other)`
 将输入张量的 *shape* 扩展为另一个输入张量的 *shape*。输出张量的维度必须遵守广播规则，即输入张量的 *shape* 维度小于或者等于另一个输入张量的 *shape* 维度。

参数：

- **other** (Tensor) - 目标张量。其 *shape* 为输入张量扩展的目标 *shape*。

返回：

维度与另一个输入张量 *other* 的相同的 Tensor，且数据类型与输入张量 *self* 相同。

异常：

- **TypeError** - 如果另一个输入 *other* 不是张量。
- **ValueError** - 如果 *self* 和 *other* 的 *shape* 不兼容。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 2, 3], [1, 2, 3]]).astype(np.float32))
>>> other = Tensor(np.array([[1, 1, 1], [1, 1, 1]]).astype(np.float32))
>>> output = x.expand_as(other)
>>> print(output)
[[1. 2. 3.]
 [1. 2. 3.]
```

`mindspore.Tensor.expand_as(x)`

将输入张量的维度扩展为目标张量的维度。

参数：

- **x** (Tensor) - 目标张量。其 *shape* 必须符合扩展的规则。

返回：

维度与目标张量的相同的 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import dtype as mstype
>>> input = Tensor([1, 2, 3], dtype=mstype.float32)
>>> x = Tensor(np.ones((2, 3)), dtype=mstype.float32)
>>> output = input.expand_as(x=x)
>>> print(output)
[[1. 2. 3.]
 [1. 2. 3.]]
```

mindspore.Tensor.expand_dims

mindspore.Tensor.**expand_dims**(axis)

详情请参考[mindspore.ops.expand_dims\(\)](#)。

mindspore.Tensor.expm1

mindspore.Tensor.**expm1**()

详情请参考[mindspore.ops.expm1\(\)](#)。

mindspore.Tensor.fill_

mindspore.Tensor.**fill_**(value)

用指定的值填充 *self*。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **value** (Union[Tensor, number.Number, bool]) - 用来填充 *self* 的值。

返回：

Tensor。

异常：

- **RunTimeError** - *self* 或 *value* 的数据类型不支持。
- **RunTimeError** - 当 *value* 是 Tensor 时，它应该是 0-D Tensor 或 shape=[1] 的 1-D Tensor。

支持平台：

Ascend

样例：

```

>>> import mindspore
>>> from mindspore import ops
>>> x = ops.zeros((3, 3))
>>> print(x)
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
>>> output = x.fill_(1.0)
>>> print(output)
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
>>> print(x)
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

```

`mindspore.Tensor.fill_diagonal`

`mindspore.Tensor.fill_diagonal(fill_value, wrap=False)`

将 *self* Tensor 的主对角线，填充成指定的值，并返回结果。*self* 必须至少为 2D，并且如果维度大于 2，则需要所有维度上的长度均相等。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **fill_value** (float) - 对角线的填充值。
- **wrap** (bool, 可选) - 控制在矩阵为高矩阵（矩阵的行比列多）的情况下，对角线的元素是否继续在剩余的行上填充。默认值：False。

返回：

- **y** (Tensor) - 和 *self* 具有相同的 shape 和数据类型。

异常：

- **TypeError** - 如果 *self* 的数据类型不是 float32、int32、int64。
- **ValueError** - 如果 *self* 的维度不大于 1。
- **ValueError** - 在 *self* 的维度大于 2 时，所有维度的长度不相等。

支持平台：

Ascend GPU CPU

样例：

```

>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.ones((6, 3)), mindspore.float32)
>>> output = x.fill_diagonal(5.0, wrap=True)
>>> print(output)

```

(下页继续)

(续上页)

```
[[5. 1. 1.]
 [1. 5. 1.]
 [1. 1. 5.]
 [1. 1. 1.]
 [5. 1. 1.]
 [1. 5. 1.]]
```

mindspore.Tensor.flatten

`mindspore.Tensor.flatten(start_dim=0, end_dim=-1)`

沿着从 `start_dim` 到 `end_dim` 的维度，对输入 Tensor 进行展平。

参数：

- **start_dim** (int, 可选) - 要展平的第一个维度。默认值：0。
- **end_dim** (int, 可选) - 要展平的最后一个维度。默认值：-1。

返回：

Tensor。如果没有维度被展平，返回原始的输入 `self`，否则返回展平后的 Tensor。如果 `self` 是零维 Tensor，将会返回一个一维 Tensor。

异常：

- **TypeError** - `start_dim` 或 `end_dim` 不是 int 类型。
- **ValueError** - 规范化后，`start_dim` 大于 `end_dim`。
- **ValueError** - `start_dim` 或 `end_dim` 不在 `[-self.dim, self.dim-1]` 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones(shape=[1, 2, 3, 4]), mindspore.float32)
>>> output = input_x.flatten(0, -1)
>>> print(output.shape)
(24,)
```

`mindspore.Tensor.flatten(order='C', *, start_dim=0, end_dim=-1)`

沿着从 `start_dim` 到 `end_dim` 的维度，对输入 Tensor 进行展平。

参数：

- **order** (str, 可选) - 仅支持 'C' 和 'F'。'C' 表示按行优先顺序 (C 风格) 展平，'F' 表示按列优先顺序 (Fortran 风格) 展平。默认值：'C'。

关键字参数：

- **start_dim** (int, 可选) - 要展平的第一个维度。默认值：0。
- **end_dim** (int, 可选) - 要展平的最后一个维度。默认值：-1。

返回：

Tensor。如果没有维度被展平, 返回原始的输入 *self*, 否则返回展平后的 Tensor。如果 *self* 是零维 Tensor, 将会返回一个一维 Tensor。

异常:

- **TypeError** - *order* 不是 string 类型。
- **ValueError** - *order* 是 string 类型, 但不是 'C' 或 'F' 。
- **TypeError** - *start_dim* 或 *end_dim* 不是 int 类型。
- **ValueError** - 规范化后, *start_dim* 大于 *end_dim* 。
- **ValueError** - *start_dim* 或 *end_dim* 不在 [-self.dim, self.dim-1] 范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones(shape=[1, 2, 3, 4]), mindspore.float32)
>>> output = input_x.flatten(order='C')
>>> print(output.shape)
(24,)
```

mindspore.Tensor.flip

`mindspore.Tensor.flip(dims)`

详情请参考[`mindspore.ops.flip\(\)`](#)。

mindspore.Tensor.fliplr

`mindspore.Tensor.fliplr()`

详情请参考[`mindspore.ops.fliplr\(\)`](#)。

mindspore.Tensor.flipud

`mindspore.Tensor.flipud()`

详情请参考[`mindspore.ops.flipud\(\)`](#)。

mindspore.Tensor.float

`mindspore.Tensor.float()`

将输入 Tensor 转换为 *float32* 类型。

返回:

Tensor, 其数据类型为 *float32*。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.int32)
>>> output = input_x.float()
>>> print(output.dtype)
Float32
```

mindspore.Tensor.float_power

`mindspore.Tensor.float_power(other)`

详情请参考[mindspore.mint.float_power\(\)](#)。

mindspore.Tensor.floor

`mindspore.Tensor.floor()`

详情请参考[mindspore.ops.floor\(\)](#)。

mindspore.Tensor.floor_

`mindspore.Tensor.floor_()`

[mindspore.Tensor.floor\(\)](#) 的 in-place 版本。

警告: 这是一个实验性 API, 后续可能修改或删除。

mindspore.Tensor.floor_divide

`mindspore.Tensor.floor_divide(other)`

按元素将 *self* Tensor 除以输入 Tensor, 并向下取整。

self 和 *other* 的输入遵循隐式类型转换规则, 使数据类型一致。输入必须是两个 Tensor 或一个 Tensor 和一个 Scalar。当输入是两个 Tensor 时, 它们的数据类型不能同时为 bool, 其 shape 可以广播。当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常量。

$$out_i = \text{floor}\left(\frac{self_i}{other_i}\right)$$

其中 *floor* 表示 Floor 算子。有关更多详细信息, 请参阅[mindspore.mint.floor](#) 算子。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **other** (Union[Tensor, Number, bool]) - 第二个输入, 为数值型, 或 bool, 或数据类型为数值型或 bool 的 Tensor。

返回:

Tensor, 输出的 shape 与广播后的 shape 相同, 数据类型取两个输入中精度较高或数字较高的。

异常:

- **TypeError** - 如果 *self* 和 *other* 不是以下之一: Tensor, number.Number 或 bool。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor
>>> import numpy as np
>>> input = Tensor(np.array([2, 4, -1]), mindspore.int32)
>>> other = Tensor(np.array([3, 3, 3]), mindspore.int32)
>>> output = input.floor_divide(other)
>>> print(output)
[ 0  1 -1]
>>> input = Tensor(2.0, mindspore.float32)
>>> other = Tensor(2.0, mindspore.float32)
>>> output = input.floor_divide(other)
>>> print(output)
1.0
```

`mindspore.Tensor.floor_divide_`

`mindspore.Tensor.floor_divide_(other)`

按元素将 *self* Tensor 除以 *other* Tensor, 并向下取整。

$$out_i = \text{floor}\left(\frac{self_i}{other_i}\right)$$

其中 *floor* 表示 Floor 算子。有关更多详细信息, 请参阅 `mindspore.mint.floor` 算子。

警告: 这是一个实验性 API, 后续可能修改或删除。

说明: 当 *self* 和 *other* 具有不同的 shape 时, *other* 必须能被广播成 *self*。

参数:

- **other** (Union[Tensor, Number, bool]) - 第二个输入, 为数值型, 或 bool, 或数据类型为数值型或 bool 的 Tensor。

返回:

Tensor, shape 与 *self* 的 shape 相同, 数据类型和 *self* 的数据类型相同。

异常:

- **TypeError** - 如果 *other* 不是以下之一: Tensor, number.Number 或 bool。
- **RuntimeError** - 如果 *other* 不能被广播成 *self*。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([2, 4, -1]), mindspore.int32)
>>> other = Tensor(np.array([3, 3, 3]), mindspore.int32)
>>> output = x.floor_divide_(other)
>>> print(output)
[ 0  1 -1]
>>> print(x)
[ 0  1 -1]
```

mindspore.Tensor.flush_from_cache

`mindspore.Tensor.flush_from_cache()`

如果 Tensor 开启缓存作用, 则将缓存数据刷新到 host 侧。

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([1, 2], dtype=np.float32))
>>> y = x.flush_from_cache()
>>> print(y)
None
```

mindspore.Tensor.fmax

`mindspore.Tensor.fmax(other)`

详情请参考[mindspore.ops.fmax\(\)](#)。

mindspore.Tensor.fmod

`mindspore.Tensor.fmod(other)`

详情请参考[mindspore.ops.fmod\(\)](#)。

mindspore.Tensor.fold

`mindspore.Tensor.fold(output_size, kernel_size, dilation=1, padding=0, stride=1)`

详情请参考[mindspore.ops.fold\(\)](#)。

mindspore.Tensor.frac

`mindspore.Tensor.frac()`

详情请参考[mindspore.ops.frac\(\)](#)。

mindspore.Tensor.from_numpy

static `mindspore.Tensor.from_numpy(array)`

将 Numpy 数组转换为张量。

当数据为非 C 连续时，数据会被拷贝成 C 连续数据后创建张量，否则则通过不复制数据的方式将 Numpy 数组转换为张量。

参数：

- **array** (numpy.array) - 输入数组。

返回：

与输入的张量具有相同的数据类型的 Tensor。

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = np.array([1, 2])
>>> output = Tensor.from_numpy(x)
>>> print(output)
[1 2]
```

mindspore.Tensor.gather

`mindspore.Tensor.gather(dim, index)`

返回输入 Tensor 在指定 *index* 对应的元素组成的切片。公式如下：

$$output[(i_0, i_1, \dots, i_{dim}, i_{dim+1}, \dots, i_n)] = input[(i_0, i_1, \dots, index[(i_0, i_1, \dots, i_{dim}, i_{dim+1}, \dots, i_n)], i_{dim+1}, \dots, i_n)]$$

警告：在 Ascend 后端，以下场景将导致不可预测的行为：

- 正向执行流程中，当 *index* 的取值不在范围 $[-self.shape[dim], self.shape[dim])$ 内；
- 反向执行流程中，当 *index* 的取值不在范围 $[0, self.shape[dim])$ 内。

参数：

- **dim** (int) - 指定要切片的维度索引。取值范围 $[-self.rank, self.rank)$ 。
- **index** (Tensor) - 指定原始 Tensor 中要切片的索引。数据类型必须是 int32 或 int64。需要同时满足以下条件：
 - $index.rank == self.rank$ ；
 - 对于 $axis \neq dim$ ， $index.shape[axis] \leq self.shape[axis]$ ；
 - *index* 的取值在有效区间 $[-self.shape[dim], self.shape[dim])$ ；

返回：

Tensor, 数据类型与 *self* 保持一致, *shape* 与 *index* 保持一致。

异常:

- **ValueError** - *index* 的 *shape* 取值非法。
- **ValueError** - *dim* 取值不在有效范围 $[-self.rank, self.rank)$ 。
- **ValueError** - *index* 的值不在有效范围。
- **TypeError** - *index* 的数据类型非法。

支持平台:

Ascend GPU CPU

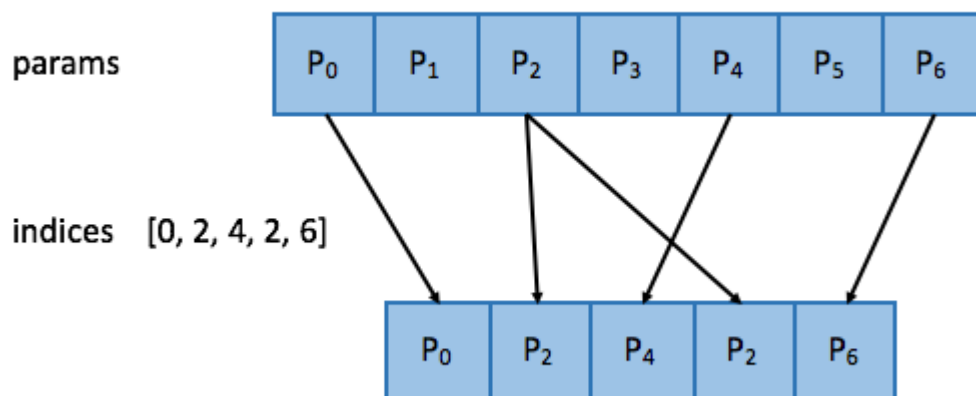
样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([[[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]], mindspore.float32)
>>> index = Tensor(np.array([[0, 0], [1, 1]]), mindspore.int32)
>>> output = input.gather(1, index)
>>> print(output)
[[-0.1 -0.1]
 [0.5   0.5]]
```

`mindspore.Tensor.gather(input_indices, axis, batch_dims=0)`

返回输入 Tensor 在指定 *axis* 上 *input_indices* 索引对应的元素组成的切片。

下图展示了 Gather 常用的计算过程:



其中, *params* 代表输入 *input_params*, *indices* 代表要切片的索引 *input_indices*。

说明:

- *input_indices* 的值必须在 $[0, input\_params.shape[axis])$ 范围内。CPU 与 GPU 平台越界访问将会抛出异常, Ascend 平台越界访问的返回结果是未定义的。
- Ascend 平台上, *input_params* 的数据类型当前不能是 `bool_`。

参数:

- **input_indices** (Tensor) - 要切片的索引 Tensor, shape 为 $(y_1, y_2, \dots, y_S)$ 。指定原始 Tensor 中要切片的索引。数据类型必须是 int32 或 int64。
- **axis** (Union(int, Tensor[int])) - 指定要切片的维度索引。它必须要大于或等于 `batch_dims`。若 `axis` 为 Tensor, 其 size 必须为 1。
- **batch_dims** (int, 可选) - 指定 batch 维的数量。它必须要小于或等于 `input_indices` 的 rank。默认值: 0。

返回:

Tensor, shape 为 `self_params.shape[: axis] + self_indices.shape[batch_dims :] + self_params.shape[axis + 1 :]`。

异常:

- **TypeError** - `axis` 不是 int 或 Tensor。
- **ValueError** - `axis` 为 Tensor 时, size 不为 1。
- **TypeError** - `input_params` 不是 Tensor。
- **TypeError** - `input_indices` 不是 int 类型的 Tensor。
- **RuntimeError** - `input_indices` 在 CPU 或 GPU 平台超出 `[0, input_params.shape[axis]]` 范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> # case1: input_indices is a Tensor with shape (5, ).
>>> input_params = Tensor(np.array([1, 2, 3, 4, 5, 6, 7]), mindspore.float32)
>>> input_indices = Tensor(np.array([0, 2, 4, 2, 6]), mindspore.int32)
>>> axis = 0
>>> output = input_params.gather(input_indices=input_indices, axis=axis)
>>> print(output)
[1. 3. 5. 3. 7.]
>>> # case2: input_indices is a Tensor with shape (2, 2). When the input_params has one_
↳ dimension,
>>> # the output shape is equal to the input_indices shape.
>>> input_indices = Tensor(np.array([[0, 2], [2, 6]]), mindspore.int32)
>>> axis = 0
>>> output = input_params.gather(input_indices=input_indices, axis=axis)
>>> print(output)
[[1. 3.]
 [3. 7.]]
>>> # case3: input_indices is a Tensor with shape (2, ) and
>>> # input_params is a Tensor with shape (3, 4) and axis is 0.
>>> input_params = Tensor(np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]), mindspore.
↳ float32)
>>> input_indices = Tensor(np.array([0, 2]), mindspore.int32)
>>> axis = 0
>>> output = input_params.gather(input_indices=input_indices, axis=axis)
>>> print(output)
[[ 1.  2.  3.  4.]
 [ 9. 10. 11. 12.]]
>>> # case4: input_indices is a Tensor with shape (2, ) and
```

(下页继续)

(续上页)

```
>>> # input_params is a Tensor with shape (3, 4) and axis is 1, batch_dims is 1.
>>> input_params = Tensor(np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]), mindspore.
↳float32)
>>> input_indices = Tensor(np.array([0, 2, 1]), mindspore.int32)
>>> axis = 1
>>> batch_dims = 1
>>> output = input_params.gather(input_indices, axis, batch_dims)
>>> print(output)
[ 1.  7. 10.]
```

`mindspore.Tensor.gather_elements`

`mindspore.Tensor.gather_elements(dim, index)`

详情请参考`mindspore.ops.gather_elements()`。

`mindspore.Tensor.gather_nd`

`mindspore.Tensor.gather_nd(indices)`

详情请参考`mindspore.ops.gather_nd()`。

`mindspore.Tensor.gcd`

`mindspore.Tensor.gcd(other)`

详情请参考`mindspore.ops.gcd()`。

`mindspore.Tensor.ge`

`mindspore.Tensor.ge(other)`

`mindspore.Tensor.greater_equal()` 的别名。

`mindspore.Tensor.geqrf`

`mindspore.Tensor.geqrf()`

详情请参考`mindspore.ops.geqrf()`。

`mindspore.Tensor.ger`

`mindspore.Tensor.ger(vec2)`

详情请参考`mindspore.ops.ger()`。

mindspore.Tensor.greater

`mindspore.Tensor.greater(other)`

详情请参考`mindspore.ops.greater()`。

mindspore.Tensor.greater_equal

`mindspore.Tensor.greater_equal(other)`

详情请参考`mindspore.ops.greater_equal()`。

mindspore.Tensor.gt

`mindspore.Tensor.gt(other)`

详情请参考`mindspore.ops.gt()`。

mindspore.Tensor.H

property `mindspore.Tensor.H`

返回共轭和转置的矩阵 (2-D Tensor) 的视图。如果输入 `x` 是复数矩阵, `x.H` 等价于 `mindspore.Tensor.swapaxes(0, 1).conj()`; 如果是实数矩阵, 则等价于 `mindspore.Tensor.swapaxes(0, 1)`。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.H
>>> print(output)
[[1 3]
 [2 4]]
```

mindspore.Tensor.half

`mindspore.Tensor.half()`

将输入 `Tensor` 转换为 `float16` 类型。

返回:

`Tensor`, 其数据类型为 `float16`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.int32)
>>> output = input_x.half()
>>> print(output.dtype)
Float16
```

mindspore.Tensor.hardshrink

`mindspore.Tensor.hardshrink(lambd=0.5)`

详情请参考[`mindspore.ops.hardshrink\(\)`](#)。

mindspore.Tensor.has_init

property `mindspore.Tensor.has_init`

判断 Tensor 是否已经初始化。

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.has_init
>>> print(output)
False
```

mindspore.Tensor.heaviside

`mindspore.Tensor.heaviside(values)`

详情请参考[`mindspore.ops.heaviside\(\)`](#)。

mindspore.Tensor.histc

`mindspore.Tensor.histc(bins=100, min=0, max=0)`

详情请参考[`mindspore.ops.histc\(\)`](#)。

支持平台：

Ascend GPU CPU

mindspore.Tensor.hsplitt

`mindspore.Tensor.hsplitt (indices_or_sections)`
详情请参考[`mindspore.ops.hsplitt\(\)`](#)。

mindspore.Tensor.hypot

`mindspore.Tensor.hypot (other)`
详情请参考[`mindspore.ops.hypot\(\)`](#)。

mindspore.Tensor.i0

`mindspore.Tensor.i0()`
详情请参考[`mindspore.ops.bessel\_i0\(\)`](#)。

mindspore.Tensor.igamma

`mindspore.Tensor.igamma (other)`
详情请参考[`mindspore.ops.igamma\(\)`](#)。

mindspore.Tensor.igammac

`mindspore.Tensor.igammac (other)`
详情请参考[`mindspore.ops.igammac\(\)`](#)。

mindspore.Tensor.imag

`mindspore.Tensor.imag()`
详情请参考[`mindspore.ops.imag\(\)`](#)。

mindspore.Tensor.index_add

`mindspore.Tensor.index_add (indices, y, axis, use_lock=True, check_index_bound=True)`
将 Tensor *y* 加到 Parameter 或 Tensor *self* 的指定 *axis* 轴和指定 *indices* 位置。要求 *axis* 轴的取值范围为 $[0, \text{len}(\text{self.dim}) - 1]$ ，*indices* 中元素的取值范围为 $[0, \text{self.shape}[\text{axis}] - 1]$ 。

参数：

- **indices** (Tensor) - 指定 Tensor *y* 加到 *self* 的 *axis* 轴的指定下标位置，要求数据类型为 `int32`。要求 *indices* shape 的维度为一维，并且 *indices* shape 的大小与 *y* shape 在 *axis* 轴上的大小一致。*indices* 中元素取值范围为 $[0, b)$ ，其中 *b* 的值为 *self* shape 在 *axis* 轴上的大小。
- **y** (Tensor) - 与 *self* 相加的 Tensor。
- **axis** (int) - 指定沿哪根轴相加。
- **use_lock** (bool, 可选) - 是否对参数更新过程加锁保护。如果为 `True`，在更新参数 *self* 的值时使用原子操作以实现加锁保护，如果为 `False`，*self* 的值可能会不可预测。默认值：`True`。
- **check_index_bound** (bool, 可选) - `True` 表示检查 *indices* 边界，`False` 表示不检查。默认值：`True`。

返回：

相加后的 Tensor。shape 和数据类型与输入 *self* 相同。

异常：

- **TypeError** - *indices* 或者 *y* 的类型不是 Tensor。
- **ValueError** - *axis* 的值超出 *self* shape 的维度范围。
- **ValueError** - *self* shape 的维度和 *y* shape 的维度不一致。
- **ValueError** - *indices* shape 的维度不是一维，或者 *indices* shape 的大小与 *y* shape 在 *axis* 轴上的大小不一致。
- **ValueError** - 除 *axis* 轴外，*self* shape 和 *y* shape 的大小不一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.float32)
>>> indices = Tensor(np.array([0, 2]), mindspore.int32)
>>> y = Tensor(np.array([[0.5, 1.0], [1.0, 1.5], [2.0, 2.5]]), mindspore.float32)
>>> output = x.index_add(indices, y, axis = 1)
>>> print(output)
[[ 1.5  2.   4. ]
 [ 5.   5.   7.5]
 [ 9.   8.  11.5]]
```

`mindspore.Tensor.index_add(dim, index, source, *, alpha=1)`

详情请参考 `mindspore.ops.index_add()`。其中 `Tensor.index_add` 与 `mindspore.ops.index_add()` 参数对应关系如下：*dim* -> *axis*、*index* -> *indices*、*source* * *alpha* -> *y*。

mindspore.Tensor.index_add_

`mindspore.Tensor.index_add_(dim, index, source, *, alpha=1)`

根据 *index* 中的索引顺序，将 *alpha* 乘以 *source* 的元素累加到 *self* 中。例如，如果 *dim* == 0, *index*[*i*] == *j*，且 *alpha* = -1，那么 *source* 的第 *i* 行将从 *self* 的第 *j* 行中被减去。*source* 的第 *dim* 维度必须与 *index* 的长度相同，且所有其他维度必须与 *self* 匹配，否则将引发错误。对于一个三维张量，输出定义如下：

$$\begin{aligned} self[index[i], :, :] &+= alpha * source[i, :, :] && \#if \ dim == 0 \\ self[:, index[i], :] &+= alpha * source[:, i, :] && \#if \ dim == 1 \\ self[:, :, index[i]] &+= alpha * source[:, :, i] && \#if \ dim == 2 \end{aligned}$$

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **dim** (int) - 指定 *index* 属于哪个维度。
- **index** (Tensor) - 指定 *self* 和 *source* 在轴 *dim* 的指定下标位置相加，数据类型为 *int32*。要求 *index* shape 的维度为一维，并且 *index* shape 的大小与 *source* shape 在 *dim* 轴上的大小一致。*index* 中元素取值范围为 [0, *b*)，其中 *b* 的值为 *self* shape 在 *dim* 轴上的大小。

- **source** (Tensor) - 输入的要进行相加的 Tensor，一定要与 *self* 有相同的数据类型，与 *self* 在 *dim* 维度有相同的 shape。

关键字参数：

- **alpha** (number, 可选) - *source* 的乘数。默认值：1。

返回：

相加后的 Tensor。shape 和数据类型与输入 *self* 相同。

异常：

- **TypeError** - *index* 或者 *source* 的类型不是 Tensor。
- **ValueError** - *dim* 的值超出 *source* shape 的维度范围。
- **ValueError** - *index* shape 的维度和 *source* shape 的维度不一致。
- **ValueError** - *index* shape 的维度不是 1D 或者 *index* shape 的大小与 *source* shape 在 *dim* 轴上的大小不一致。
- **ValueError** - 除 *dim* 轴外，*self* shape 和 *source* shape 的大小不一致。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.float32)
>>> index = Tensor(np.array([0, 2]), mindspore.int32)
>>> y = Tensor(np.array([[0.5, 1.0], [1.0, 1.5], [2.0, 2.5]]), mindspore.float32)
>>> output = x.index_add_(1, index, y, alpha=1)
>>> print(output)
[[ 1.5  2.   4. ]
 [ 5.   5.   7.5]
 [ 9.   8.  11.5]]
>>> print(x)
[[ 1.5  2.   4. ]
 [ 5.   5.   7.5]
 [ 9.   8.  11.5]]
```

mindspore.Tensor.index_fill

`mindspore.Tensor.index_fill(axis, index, value)`

详情请参考[mindspore.ops.index_fill\(\)](#)。

mindspore.Tensor.index_put

`mindspore.Tensor.index_put(indices, values, accumulate=False)`

根据 *indices* 中的下标值, 使用 *values* 中的数值替换 Tensor *self* 中的相应元素的值。非原地更新版本的 `mindspore.Tensor.index_put_()`。

警告: 以下场景将导致不可预测的行为:

- 如果 *accumulate* 为 *False*, 且 *indices* 存在重复元素。

参数:

- **indices** (tuple[Tensor], list[Tensor]) - 元素类型是 int32 或者 int64, 用于对 *self* 中的元素进行索引。*indices* 中的 Tensor 的秩应为 1-D, *indices* 的 size 应小于或等于 *self* 的秩, *indices* 中的 Tensor 应是可广播的。
- **values** (Tensor) - 一个一维的 Tensor, 数据类型与 *self* 相同。如果其 size 为 1, 则它是可广播的。
- **accumulate** (bool, 可选) - 如果 *accumulate* 为 True, *values* 中的元素被累加到 *self* 的相应元素上; 否则, 用 *values* 中的元素取代 *self* 的相应元素。默认值: False。

返回:

Tensor, 其数据类型和 shape 与 *self* 相同。

异常:

- **TypeError** - 如果 *self* 的 dtype 与 *values* 的 dtype 不同。
- **TypeError** - 如果 *indices* 的 dtype 不是 tuple[Tensor] 或者 list[Tensor]。
- **TypeError** - 如果 *indices* 中的 Tensor 的 dtype 不是 int32 或者 int64。
- **TypeError** - 如果 *indices* 中的 Tensor 的 dtype 是不一致的。
- **TypeError** - 如果 *accumulate* 的 dtype 不是 bool。
- **ValueError** - 如果 *values* 的秩不是 1-D。
- **ValueError** - 当 *self* 的 rank 与 *indices* 的 size 相等时, 如果 *values* 的 size 不为 1 或者不为 *indices* 中 Tensor 的最大 size。
- **ValueError** - 当 *self* 的 rank 大于 *indices* 的 size 时, 如果 *values* 的 size 不为 1 或者不为 *self* 的最后一维的 shape。
- **ValueError** - 如果 *indices* 中的 Tensor 的秩不是 1-D。
- **ValueError** - 如果 *indices* 中的 Tensor 不是可广播的。
- **ValueError** - 如果 *indices* 的 size 大于 *self* 的秩。

支持平台:

Ascend CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6]]).astype(np.int32))
>>> values = Tensor(np.array([3]).astype(np.int32))
>>> indices = [Tensor(np.array([0, 1, 1]).astype(np.int32)), Tensor(np.array([1, 2, 1]).
↪astype(np.int32))]
```

(下页继续)

(续上页)

```
>>> accumulate = True
>>> output = x.index_put(indices, values, accumulate)
>>> print(output)
[[1 5 3]
 [4 8 9]]
```

`mindspore.Tensor.index_put_`

`mindspore.Tensor.index_put_(indices, values, accumulate=False)`

根据 *indices* 中的下标值，使用 *values* 中的数值替换 Tensor *self* 中相应元素的值。*Tensor.index_put_(indices, values)* 等价于 *tensor[indices] = values*，更新并返回 *self*。

警告：以下场景将导致不可预测的行为：

- 如果 *accumulate* 为 *False*，且 *indices* 存在重复元素。

参数：

- **indices** (tuple[Tensor], list[Tensor]) - 元素类型是 `bool`、`uint8`、`int32` 或者 `int64`，用于对 *self* 中的元素进行索引。*indices* 的 size 应小于或等于 *self* 的秩，*indices* 中的 Tensor 应是可广播的。
- **values** (Tensor) - 其数据类型与 *self* 相同。如果其 size 为 1，则它是可广播的。
- **accumulate** (bool, 可选) - 如果 *accumulate* 为 *True*，*values* 中的元素被累加到 *self* 的相应元素上；否则，用 *values* 中的元素取代 *self* 的相应元素。默认值: *False*。

返回：

Tensor *self*。

异常：

- **TypeError** - 如果 *self* 的 dtype 与 *values* 的 dtype 不同。
- **TypeError** - 如果 *indices* 的 dtype 不是 tuple[Tensor] 或者 list[Tensor]。
- **TypeError** - 如果 *indices* 中的 Tensor 的 dtype 不是 `bool`、`uint8`、`int32` 或者 `int64`。
- **TypeError** - 如果 *indices* 中的 Tensor 的 dtype 是不一致的。
- **TypeError** - 如果 *accumulate* 的 dtype 不是 `bool`。
- **ValueError** - 当 *self* 的 rank 与 *indices* 的 size 相等时，如果 *values* 的 size 不为 1 或者不为 *indices* 中 Tensor 的最大 size。
- **ValueError** - 当 *self* 的 rank 大于 *indices* 的 size 时，如果 *values* 的 size 不为 1 或者不为 *self* 的最后一维的 shape。
- **ValueError** - 如果 *indices* 中的 Tensor 不是可广播的。
- **ValueError** - 如果 *indices* 的 size 大于 *self* 的秩。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6]]).astype(np.int32))
>>> values = Tensor(np.array([3]).astype(np.int32))
>>> indices = [Tensor(np.array([0, 1, 1]).astype(np.int32)), Tensor(np.array([1, 2, 1]).
    ↳astype(np.int32))]
>>> accumulate = True
>>> x.index_put_(indices, values, accumulate)
>>> print(x)
[[1 5 3]
 [4 8 9]]
```

mindspore.Tensor.index_select

`mindspore.Tensor.index_select(axis, index)`

返回一个新的 Tensor，该 Tensor 沿维度 *axis* 按 *index* 中给定的索引对 *self* 进行选择。返回的 Tensor 和输入 Tensor(*self*) 的维度数量相同，其第 *axis* 维度的大小和 *index* 的长度相同；其他维度和 *self* 相同。

说明: *index* 的值必须在 $[0, self.shape[axis])$ 范围内，超出该范围结果未定义。

参数:

- **axis** (int) - 根据索引进行选择的维度。
- **index** (Tensor) - 包含索引的一维 Tensor。

返回:

Tensor，数据类型与输入 *self* 相同。

异常:

- **TypeError** - *index* 的类型不是 Tensor。
- **TypeError** - *axis* 的类型不是 int。
- **ValueError** - *axis* 值超出范围 $[-input.ndim, input.ndim - 1]$ 。
- **ValueError** - *index* 不是一维 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor
>>> import numpy as np
>>> input = Tensor(np.arange(16).astype(np.float32).reshape(2, 2, 4))
>>> print(input)
[[[ 0.  1.  2.  3.]
  [ 4.  5.  6.  7.]
  [ 8.  9. 10. 11.]
  [12. 13. 14. 15.]]]
>>> index = Tensor([0,], mindspore.int32)
```

(下页继续)

(续上页)

```
>>> y = input.index_select(1, index)
>>> print(y)
[[[ 0.  1.  2.  3.]]
 [[ 8.  9. 10. 11.]]]
```

`mindspore.Tensor.index_select(dim, index)`

返回一个新的 Tensor，该 Tensor 沿维度 *dim* 按 *index* 中给定的索引对 *self* 进行选择。返回的 Tensor 和输入 Tensor(*self*) 的维度数量相同，其第 *dim* 维度的大小和 *index* 的长度相同；其他维度和 *self* 相同。

说明：index 的值必须在 $[0, self.shape[dim])$ 范围内，超出该范围结果未定义。

参数：

- **dim** (int) - 根据索引进行选择的维度。
- **index** (Tensor) - 包含索引的一维 Tensor。

返回：

Tensor，数据类型与输入 *self* 相同。

异常：

- **TypeError** - *index* 的类型不是 Tensor。
- **TypeError** - *dim* 的类型不是 int。
- **ValueError** - *dim* 值超出范围 $[-input.ndim, input.ndim - 1]$ 。
- **ValueError** - *index* 不是一维 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> import numpy as np
>>> input = Tensor(np.arange(16).astype(np.float32).reshape(2, 2, 4))
>>> print(input)
[[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]]]
>>> index = Tensor([0,], mindspore.int32)
>>> y = input.index_select(1, index)
>>> print(y)
[[[ 0.  1.  2.  3.]
 [ 8.  9. 10. 11.]]]
```

mindspore.Tensor.init_data

`mindspore.Tensor.init_data (slice_index=None, shape=None, opt_shard_group=None)`

获取此 Tensor 的数据。

说明：对于同一个 Tensor，只可以调用一次 `init_data` 函数。

参数：

- **slice_index** (int) - 参数切片的索引。在初始化参数切片的时候使用，保证使用相同切片的设备可以生成相同的 Tensor。默认值：None。
- **shape** (list[int]) - 切片的 shape，在初始化参数切片时使用。默认值：None。
- **opt_shard_group** (str) - 优化器分片组，在自动或半自动并行模式下用于获取参数的切片。关于优化器分组，请参考 [优化器并行](#)。默认值：None。

返回：

初始化的 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore.common.initializer import initializer, Constant
>>> x = initializer(Constant(1), [2, 2], ms.float32)
>>> out = x.init_data()
>>> print(out)
[[1. 1.]
 [1. 1.]]
```

mindspore.Tensor.inner

`mindspore.Tensor.inner (other)`

详情请参考 `mindspore.ops.inner()`。

mindspore.Tensor.inplace_update

`mindspore.Tensor.inplace_update (v, indices)`

详情请参考 `mindspore.ops.inplace_update()`。

mindspore.Tensor.int

`mindspore.Tensor.int()`

将输入 `Tensor` 转换为 `int32` 类型，其中 `Tensor` 中的值为浮点数时，则会丢弃小数部分。

返回：

`Tensor`，其数据类型为 `int32`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.float32)
>>> output = input_x.int()
>>> print(output.dtype)
Int32
```

mindspore.Tensor.inv

`mindspore.Tensor.inv()`

详情请参考[`mindspore.ops.inv\(\)`](#)。

mindspore.Tensor.inverse

`mindspore.Tensor.inverse()`

详情请参考[`mindspore.ops.inverse\(\)`](#)。

支持平台：

Ascend GPU CPU

mindspore.Tensor.invert

`mindspore.Tensor.invert()`

详情请参考[`mindspore.ops.invert\(\)`](#)。

mindspore.Tensor.isclose

`mindspore.Tensor.isclose(other, rtol=1e-05, atol=1e-08, equal_nan=False)`

返回一个 `bool` 类型的 `Tensor`，表示 `input` 的每个元素与 `other` 的对应元素在给定容忍度内是否“接近”。其中“接近”的数学公式为：

$$|input - other| \leq atol + rtol * |other|$$

参数：

- **other** (`Tensor`) - 对比的第二个输入。

- **rtol** (float, 可选) - 相对容忍度值, 默认: $1e-05$ 。
- **atol** (float, 可选) - 绝对容忍度值, 默认: $1e-08$ 。
- **equal_nan** (bool, 可选) - 如果设置为 `True`, 可以认为两个 NaN 是相等的, 默认值: `False`。

返回:

Tensor。shape 与广播后的 shape 相同, 数据类型是 bool。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([1.3, 2.1, 3.2, 4.1, 5.1]), mindspore.float16)
>>> other = Tensor(np.array([1.3, 3.3, 2.3, 3.1, 5.1]), mindspore.float16)
>>> output = Tensor.isclose(input, other)
>>> print(output)
[ True False False False  True]
```

`mindspore.Tensor.isclose(x2, rtol=1e-05, atol=1e-08, equal_nan=False)`

返回一个 bool 类型的 Tensor, 表示 *input* 的每个元素与 *x2* 的对应元素在给定容忍度内是否“接近”。其中“接近”的数学公式为:

$$|input - x2| \leq atol + rtol * |x2|$$

参数:

- **x2** (Tensor) - 对比的第二个输入。数据类型必须与 *input* 相同。
- **rtol** (Union[float, int, bool], 可选) - 相对容忍度值, 默认: $1e-05$ 。
- **atol** (Union[float, int, bool], 可选) - 绝对容忍度值, 默认: $1e-08$ 。
- **equal_nan** (bool, 可选) - 如果设置为 `True`, 可以认为两个 NaN 是相等的, 默认值: `False`。

返回:

Tensor, shape 与广播后的 shape 相同, 数据类型是 bool。

异常:

- **TypeError** - *x2* 的类型不是 Tensor。
- **TypeError** - *input* 和 *x2* 的数据类型不在支持的类型列表中。支持的类型有 float16、float32、float64、int8、int16、int32、int64 和 uint8, Ascend 平台额外支持 bfloat16 和 bool 类型。
- **TypeError** - *atol* 或 *rtol* 不是 float、int 或 bool。
- **TypeError** - *equal_nan* 不是 bool。
- **TypeError** - *input* 和 *x2* 的数据类型不同。
- **ValueError** - *input* 和 *x2* 无法广播。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([1.3, 2.1, 3.2, 4.1, 5.1]), mindspore.float16)
>>> x2 = Tensor(np.array([1.3, 3.3, 2.3, 3.1, 5.1]), mindspore.float16)
>>> output = Tensor.isclose(input, x2)
>>> print(output)
[ True False False False  True]
```

mindspore.Tensor.isfinite

`mindspore.Tensor.isfinite()`

详情请参考[`mindspore.ops.isfinite\(\)`](#)。

mindspore.Tensor.is_complex

`mindspore.Tensor.is_complex()`

详情请参考[`mindspore.ops.is\_complex\(\)`](#)。

mindspore.Tensor.is_contiguous

`mindspore.Tensor.is_contiguous()`

判断 Tensor 的内存是否连续。

返回:

Bool, 如果 Tensor 的内存是连续的, 则返回 True, 否则返回 False。

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor([[1, 2, 3], [4, 5, 6]], dtype=ms.float32)
>>> y = ops.transpose(x, (1, 0))
>>> print(y.is_contiguous())
False
```

mindspore.Tensor.is_floating_point

`mindspore.Tensor.is_floating_point()`

详情请参考[`mindspore.ops.is\_floating\_point\(\)`](#)。

mindspore.Tensor.isinf

`mindspore.Tensor.isinf()`

详情请参考[`mindspore.ops.isinf\(\)`](#)。

支持平台：

Ascend CPU GPU

mindspore.Tensor.isnan

`mindspore.Tensor.isnan()`

详情请参考[`mindspore.ops.isnan\(\)`](#)。

mindspore.Tensor.isneginf

`mindspore.Tensor.isneginf()`

详情请参考[`mindspore.ops.isneginf\(\)`](#)。

mindspore.Tensor.isposinf

`mindspore.Tensor.isposinf()`

详情请参考[`mindspore.ops.isposinf\(\)`](#)。

mindspore.Tensor.isreal

`mindspore.Tensor.isreal()`

详情请参考[`mindspore.ops.isreal\(\)`](#)。

mindspore.Tensor.is_signed

`mindspore.Tensor.is_signed()`

判断输入 Tensor 的 dtype 是否有符号数类型。

返回：

Bool, 如果该 Tensor 的 dtype 是有符号数类型, 则返回 True, 否则返回 False。

样例：

```

>>> import mindspore as ms
>>> x = ms.Tensor([1, 2, 3], ms.int64)
>>> y = ms.Tensor([1, 2, 3], ms.uint64)
>>> output = x.is_signed()
>>> output2 = y.is_signed()
>>> print(output)
True
>>> print(output2)
False

```

mindspore.Tensor.item

`mindspore.Tensor.item()`

返回只包含一个元素的 Tensor 的元素值。

返回：

标量，类型由 Tensor 的 dtype 决定。

异常：

- **ValueError** - Tensor 的元素个数不止一个。
- **TypeError** - 调用 item 接口的 Tensor 的元素数据类型不被支持。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> x = Tensor(1.2, ms.float32)
>>> print(x.item())
1.2
```

mindspore.Tensor.itemset

`mindspore.Tensor.itemset(*args)`

将标量插入到 Tensor（并将标量转换为 Tensor 的数据类型）。

至少有 1 个参数，并且最后一个参数被定义为设定值。Tensor.itemset(*args) 等同于 `Tensor[args] = item`。

参数：

- **args** (Union[(numbers.Number), (int/tuple(int), numbers.Number)]) - 指定索引和值的参数。如果 args 包含一个参数（标量），则其仅在 Tensor 大小为 1 的情况下使用。如果 args 包含两个参数，则最后一个参数是要设置的值且必须是标量，而第一个参数指定单个 Tensor 元素的位置。参数值是整数或者元组。

返回：

一个新的 Tensor，其值为 `Tensor[args] = item`。

异常：

- **ValueError** - 第一个参数的长度不等于 Tensor 的 ndim。
- **IndexError** - 只提供了一个参数，并且原来的 Tensor 不是标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1,2,3],[4,5,6]], dtype=np.float32))
>>> print(x.itemset((0,1), 4))
[[1. 4. 3.]
 [4. 5. 6.]]
```

(下页继续)

(续上页)

```
>>> print(x)
[[1. 2. 3.]
 [4. 5. 6.]]
```

mindspore.Tensor.itemsize

property mindspore.Tensor.itemsize

返回一个 Tensor 元素的长度（以字节为单位）。

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.itemsize
>>> print(output)
8
```

mindspore.Tensor.lcm

mindspore.Tensor.lcm(*other*)

详情请参考[mindspore.ops.lcm\(\)](#)。

mindspore.Tensor.ldexp

mindspore.Tensor.ldexp(*other*)

详情请参考[mindspore.ops.ldexp\(\)](#)。

mindspore.Tensor.le

mindspore.Tensor.le(*other*)

详情请参考[mindspore.ops.le\(\)](#)。

mindspore.Tensor.lerp

mindspore.Tensor.lerp(*end, weight*)

详情请参考[mindspore.ops.lerp\(\)](#)。

mindspore.Tensor.less

`mindspore.Tensor.less (other)`
详情请参考[`mindspore.ops.less\(\)`](#)。

mindspore.Tensor.less_equal

`mindspore.Tensor.less_equal (other)`
详情请参考[`mindspore.ops.less\_equal\(\)`](#)。

mindspore.Tensor.log

`mindspore.Tensor.log()`
详情请参考[`mindspore.ops.log\(\)`](#)。

mindspore.Tensor.log10

`mindspore.Tensor.log10()`
详情请参考[`mindspore.ops.log10\(\)`](#)。

mindspore.Tensor.log1p

`mindspore.Tensor.log1p()`
详情请参考[`mindspore.ops.log1p\(\)`](#)。

mindspore.Tensor.log2

`mindspore.Tensor.log2()`
详情请参考[`mindspore.ops.log2\(\)`](#)。

mindspore.Tensor.logaddexp

`mindspore.Tensor.logaddexp (other)`
详情请参考[`mindspore.ops.logaddexp\(\)`](#)。

mindspore.Tensor.logaddexp2

`mindspore.Tensor.logaddexp2 (other)`
详情请参考[`mindspore.ops.logaddexp2\(\)`](#)。

mindspore.Tensor.logcumsumexp

`mindspore.Tensor.logcumsumexp(axis)`

详情请参考[mindspore.ops.logcumsumexp\(\)](#)。

警告：这是一个实验性 API，后续可能修改或删除。

mindspore.Tensor.logdet

`mindspore.Tensor.logdet()`

详情请参考[mindspore.ops.logdet\(\)](#)。

mindspore.Tensor.logical_and

`mindspore.Tensor.logical_and(other)`

详情请参考[mindspore.ops.logical_and\(\)](#)。

mindspore.Tensor.logical_not

`mindspore.Tensor.logical_not()`

详情请参考[mindspore.ops.logical_not\(\)](#)。

mindspore.Tensor.logical_or

`mindspore.Tensor.logical_or(other)`

详情请参考[mindspore.ops.logical_or\(\)](#)。

mindspore.Tensor.logical_xor

`mindspore.Tensor.logical_xor(other)`

逐元素计算两个 Tensor 的逻辑异或运算。

$$out_i = self_i \oplus other_i$$

说明：

- *self* 和 *other* 的输入遵循隐式类型转换规则，使数据类型一致。
- *other* 是一个 bool 时，bool 对象只能是一个常量。

参数：

- **other** (Union[Tensor, bool]) - bool 或者数据类型可被隐式转换为 bool 的 Tensor。

返回：

Tensor，其 shape 与广播后的 shape 相同，数据类型为 bool。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([True, False, True]), mindspore.bool_)
>>> other = Tensor(np.array([True, True, False]), mindspore.bool_)
>>> output = input.logical_xor(other)
>>> print(output)
[ False True True]
>>> x = Tensor(1, mindspore.bool_)
>>> other = Tensor(0, mindspore.bool_)
>>> output = input.logical_xor(other)
>>> print(output)
True
```

mindspore.Tensor.logit

`mindspore.Tensor.logit (eps=None)`

详情请参考[mindspore.ops.logit\(\)](#)。

mindspore.Tensor.logsumexp

`mindspore.Tensor.logsumexp (dim, keepdim=False)`

详情请参考[mindspore.ops.logsumexp\(\)](#)。

mindspore.Tensor.log_normal

`mindspore.Tensor.log_normal (mean=1.0, std=2.0)`

使用给定均值 *mean* 和标准差 *std* 的对数正态分布的数值填充当前 Tensor。

$$f(x; 1.0, 2.0) = \frac{1}{x\delta\sqrt{2\pi}} e^{-\frac{(\ln x - \mu)^2}{2\delta^2}}$$

其中 μ 、 δ 分别是对数正态分布的均值和标准差。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **mean** (float, 可选) - 对数正态分布的均值。默认值：1.0。
- **std** (float, 可选) - 对数正态分布的标准差。默认值：2.0。

返回：

Tensor，具有与当前 Tensor 相同的 shape 和 dtype。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> x = mindspore.Tensor(np.array([[1, 2], [3, 4]]), dtype=mindspore.float32)
>>> output = x.log_normal()
>>> print(output)
[[1.2788825 2.3305743]
 [14.944194 0.16303174]]
```

mindspore.Tensor.long

`mindspore.Tensor.long()`

将输入 `Tensor` 转换为 `int64` 类型, 其中 `Tensor` 中的值为浮点数时, 则会丢弃小数部分。

返回:

`Tensor`, 其数据类型为 `int64`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor(np.ones([2,2]), mindspore.int32)
>>> output = input_x.long()
>>> print(output.dtype)
Int64
```

mindspore.Tensor.lt

`mindspore.Tensor.lt(other)`

`mindspore.Tensor.less()` 的别名。

mindspore.Tensor.lu_solve

`mindspore.Tensor.lu_solve(LU_data, LU_pivots)`

详情请参考 `mindspore.ops.lu_solve()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

mindspore.Tensor.masked_fill

`mindspore.Tensor.masked_fill(mask, value)`
详情请参考`mindspore.ops.masked_fill()`。

mindspore.Tensor.masked_fill_

`mindspore.Tensor.masked_fill_(mask, value)`
`mindspore.Tensor.masked_fill()` 的 in-place 版本。

警告：这是一个实验性 API，后续可能修改或删除。

mindspore.Tensor.masked_scatter

`mindspore.Tensor.masked_scatter(mask, x)`
根据 `mask`，使用 `tensor` 中的值，更新 Tensor 本身的值，返回一个 Tensor。`mask` 和 Tensor 本身的 shape 必须相等或者 `mask` 是可广播的。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **mask** (Tensor[bool]) - 一个 bool Tensor，其 shape 可以被广播到 Tensor 本身。
- **x** (Tensor) - 一个 Tensor，其数据类型与 Tensor 本身相同。`tensor` 中的元素数量必须大于等于 `mask` 中的 True 元素的数量。

返回：

Tensor，其数据类型和 shape 与 Tensor 本身相同。

异常：

- **TypeError** - 如果 `mask` 或者 `x` 不是 Tensor。
- **TypeError** - 如果 Tensor 本身的数据类型不被支持。
- **TypeError** - 如果 `mask` 的 dtype 不是 bool。
- **TypeError** - 如果 Tensor 本身的维度数小于 `mask` 的维度数。
- **ValueError** - 如果 `mask` 不能广播到 Tensor 本身。
- **ValueError** - 如果 `x` 中的元素数目小于 Tensor 本身需要更新的元素数目。

支持平台：

Ascend CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([1., 2., 3., 4.]), mindspore.float32)
>>> mask = Tensor(np.array([True, True, False, True]), mindspore.bool_)
>>> tensor = Tensor(np.array([5., 6., 7.]), mindspore.float32)
>>> output = x.masked_scatter(mask, tensor)
>>> print(output)
[5. 6. 3. 7.]
```

mindspore.Tensor.masked_select

mindspore.Tensor.**masked_select** (*mask*)

详情请参考[mindspore.ops.masked_select\(\)](#)。

mindspore.Tensor.matmul

mindspore.Tensor.**matmul** (*tensor2*)

计算两个数组的矩阵乘积。

说明:

- 不支持 NumPy 参数 *out*、*casting*、*order*、*subok*、*signature*、*extobj*。
- *self* 和 *tensor2* 的数据类型必须一致。
- 在 Ascend 平台上，*self* 和 *tensor2* 的维度必须在 1 到 6 之间。
- 在 GPU 平台上，*self* 和 *tensor2* 支持的数据类型是 ms.float16 和 ms.float32。

参数:

- **tensor2** (Tensor) - 输入 Tensor，不支持 Scalar，*self* 的最后一维度和 *tensor2* 的倒数第二维度相等，且 *self* 和 *tensor2* 彼此支持广播。

返回:

Tensor 或 Scalar，输入的矩阵乘积。仅当 *self* 和 *tensor2* 为一维向量时，输出为 Scalar。

异常:

- **TypeError** - *self* 的 dtype 和 *tensor2* 的 dtype 不一致。
- **ValueError** - *self* 的最后一维度和 *tensor2* 的倒数第二维度不相等，或者输入的是 Scalar。
- **ValueError** - *self* 和 *tensor2* 彼此不能广播。
- **RuntimeError** - 在 Ascend 平台上，*self* 或 *tensor2* 的维度小于 1 或大于 6。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> # case 1 : Reasonable application of broadcast mechanism
>>> input = Tensor(np.arange(2 * 3 * 4).reshape(2, 3, 4), mindspore.float32)
>>> other = Tensor(np.arange(4 * 5).reshape(4, 5), mindspore.float32)
>>> output = input.matmul(other)
>>> print(output)
[[[ 70.  76.  82.  88.  94.]
  [ 190.  212.  234.  256.  278.]
  [ 310.  348.  386.  424.  462.]]
 [[ 430.  484.  538.  592.  646.]
  [ 550.  620.  690.  760.  830.]
  [ 670.  756.  842.  928. 1014.]]]
>>> print(output.shape)
(2, 3, 5)
>>> # case 2 : the rank of `tensor2` is 1
>>> input = Tensor(np.ones([1, 2]), mindspore.float32)
>>> other = Tensor(np.ones([2,]), mindspore.float32)
>>> output = input.matmul(other)
>>> print(output)
[2.]
>>> print(output.shape)
(1,)

```

mindspore.Tensor.max

`mindspore.Tensor.max()`

返回输入 Tensor 的最大值。

返回：

Tensor，值为输入 Tensor 的最大值，类型与 *input* 相同。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output = x.max()
>>> print(output)
[0.7]

```

`mindspore.Tensor.max(dim, keepdim=False)`

在给定轴上计算输入 Tensor 的最大值，并返回最大值和索引值。

参数：

- **dim** (int) - 指定计算维度。
- **keepdim** (bool, 可选) - 表示是否保持维度，如果为 True，输出将与输入保持相同的维度；如果为 False，输出将减少维度。默认值：False。

返回：

`tuple(Tensor)`，返回两个元素类型为 `Tensor` 的 `tuple`，包含输入 `Tensor` 沿指定维度 `dim` 的最大值和相应的索引。

- **values** (`Tensor`) - 输入 `Tensor` 沿给定维度的最大值，数据类型和 `self` 相同，`shape` 和 `index` 相同。
- **index** (`Tensor`) - 输入 `Tensor` 的沿给定维度的最大值索引，数据类型为 `int64`。如果 `keepdim` 为 `True`，输出 `Tensor` 的 `shape` 是 $(self_1, self_2, \dots, self_{dim-1}, 1, self_{dim+1}, \dots, self_N)$ 。否则输出 `shape` 为 $(self_1, self_2, \dots, self_{dim-1}, self_{dim+1}, \dots, self_N)$ 。

异常：

- **TypeError** - 如果 `keepdim` 不是 `bool` 类型。
- **TypeError** - 如果 `dim` 不是 `int` 类型。
- **TypeError** - 如果输入 `Tensor` 的数据类型为 `Complex`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output, index = x.max(0, keepdim=True)
>>> print(output, index)
[0.7] [3]
```

`mindspore.Tensor.max(axis=None, keepdims=False, *, initial=None, where=True, return_indices=False)`

返回 `Tensor` 的最大值或轴方向上的最大值。

说明：`axis` 为 `None` 时，`keepdims` 及以后参数均不会生效，同时索引固定返回 0。

参数：

- **axis** (`Union[None, int, list, tuple of ints]`, 可选) - 轴，在该轴方向上进行操作。默认情况下，使用扁平输入。如果该参数为整数元组，则在多个轴上选择最大值，而不是在单个轴或所有轴上进行选择。默认值：`None`。
- **keepdims** (`bool`, 可选) - 如果这个参数为 `True`，被删去的维度保留在结果中，且维度大小设为 1。有了这个选项，结果就可以与输入数组进行正确的广播运算。默认值：`False`。

关键字参数：

- **initial** (`scalar`, 可选) - 输出元素的最小值。如果对空切片进行计算，则该参数必须设置。默认值：`None`。
- **where** (`bool Tensor`, 可选) - 一个 `bool` 数组，被广播以匹配数组维度和选择包含在降维中的元素。如果传递了一个非默认值，则还必须提供初始值。默认值：`True`。
- **return_indices** (`bool`, 可选) - 是否返回最大值的下标。默认值：`False`。如果 `axis` 是一个 `list` 或一个 `int` 类型的 `tuple`，则必须取值为 `False`。

返回：

`Tensor` 或标量，输入 `Tensor` 的最大值。如果 `axis` 为 `None`，则结果是一个标量值。如果提供了 `axis`，则结果是 `Tensor` `ndim - 1` 维度的一个数组。

异常：

- **TypeError** - 参数具有前面未指定的类型。

参见:

- `mindspore.Tensor.argmaxin()` : 返回沿轴最小值的索引。
- `mindspore.Tensor.argmax()` : 返回沿轴最大值的索引。
- `mindspore.Tensor.min()` : 返回整个 Tensor 最小值或沿轴最小值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> a = Tensor(np.arange(4).reshape((2, 2)).astype('float32'))
>>> output = a.max()
>>> print(output)
3.0
>>> value, indices = a.max(axis=0, return_indices=True)
>>> print(value)
[2. 3.]
>>> print(indices)
[1 1]
```

mindspore.Tensor.maximum

`mindspore.Tensor.maximum(other)`

详情请参考[mindspore.ops.maximum\(\)](#)。

mindspore.Tensor.mean

`mindspore.Tensor.mean(dim=None, keepdim=False, *, dtype=None)`

默认情况下, 移除输入所有维度, 返回 input 中所有元素的平均值, 也可仅缩小指定维度 `dim` 大小至 1。`keepdim` 控制输出和输入的维度是否相同。

说明: Tensor 类型的 `dim` 仅用作兼容旧版本, 不推荐使用。

参数:

- **dim** (Union[int, tuple(int), list(int), Tensor], 可选) - 要减少的维度。默认值: None。缩小所有维度, 只允许常量值。假设 `self` 的秩为 `r`, 其取值范围为 `[-r, r)`。
- **keepdim** (bool, 可选) - 如果为 True, 则保留缩小的维度并且大小为 1。否则移除维度。默认值: False。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 期望返回的 Tensor 数据类型。默认值: None。

返回:

Tensor。与输入 Tensor 拥有相同的数据类型。

- 如果 `dim` 设置为 None, 并且 `keepdim` 设置为 False, 则输出一个零维 Tensor, 表示输入 Tensor 中所有元素的平均值。

- 如果 *dim* 设置为 `int`, 取值为 1, 并且 *keepdim* 设置为 `False`, 则输出的 `shape` 为: $(x_0, x_2, \dots, x_R)$ 。
- 如果 *dim* 设置为 `tuple(int)`, 取值为 (1, 2), 并且 *keepdim* 设置为 `False`, 则输出的 `shape` 为: $(x_0, x_3, \dots, x_R)$ 。
- 如果 *dim* 是一个一维 `Tensor`, 取值为 [1, 2], 并且 *keepdim* 设置 `False`, 则输出的 `shape` 为: $(x_0, x_3, \dots, x_R)$ 。

异常:

- **TypeError** - 如果 *dim* 不是一下类型中的一种: `int`、`tuple`、`list` 或者 `Tensor`。
- **TypeError** - 如果 *keepdim* 不是 `bool`。
- **ValueError** - 如果 *dim* 超出取值范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> output = Tensor.mean(x, 1, keepdim=True)
>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by averaging all elements in the dimension.
>>> x = Tensor(np.array([[[[2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2]],
... [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
... [[6, 6, 6, 6, 6, 6], [8, 8, 8, 8, 8, 8], [10, 10, 10, 10, 10, 10]]]),
... mindspore.float32)
>>> output = Tensor.mean(x)
>>> print(output)
5.0
>>> print(output.shape)
()
>>> # case 2: Reduces a dimension along the dim 0
>>> output = Tensor.mean(x, 0, True)
>>> print(output)
[[[4. 4. 4. 4. 4. 4.]
  [5. 5. 5. 5. 5. 5.]
  [6. 6. 6. 6. 6. 6.]]]
>>> # case 3: Reduces a dimension along the dim 1
>>> output = Tensor.mean(x, 1, True)
>>> print(output)
[[[2. 2. 2. 2. 2. 2.]
  [5. 5. 5. 5. 5. 5.]
  [8. 8. 8. 8. 8. 8.]]]
>>> # case 4: Reduces a dimension along the dim 2
>>> output = Tensor.mean(x, 2, True)
>>> print(output)
[[[ 2.]
  [ 2.]
  [ 2.]]
 [[ 4.]
  [ 5.]
  [ 6.]]
 [[ 6.]
  [ 6.]
  [ 6.]
```

(下页继续)

(续上页)

```
[ 8.]
[10.]]]
```

`mindspore.Tensor.mean(axis=None, keep_dims=False)`

详情请参考 `mindspore.ops.mean()`。

mindspore.Tensor.median

`mindspore.Tensor.median(axis=-1, keepdims=False)`

输出 Tensor 指定维度上的中值与索引。

警告:

- 如果 *input* 的中值不为唯一, 则 *indices* 不一定对应第一个出现的中值。该接口的具体实现方式和后端类型相关, CPU 和 GPU 的返回值可能不相同。

参数:

- **axis** (int, 可选) - 指定计算的轴。默认值: -1。
- **keepdims** (bool, 可选) - 输出 Tensor 是否需要保留 *axis* 的维度。默认值: False。

返回:

- **y** (Tensor) - 返回指定维度上的中值。数据类型与 *input* 相同。
- **indices** (bool) - 指定中值索引。数据类型为 int64。

异常:

- **TypeError** - *axis* 不是 int。
- **TypeError** - *keepdims* 不是 bool。
- **ValueError** - *axis* 的范围不在 $[-\text{dims}, \text{dims} - 1]$, *dims* 表示 *x* 的维度长度。

支持平台:

GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[0.57, 0.11, 0.21], [0.38, 0.50, 0.57], [0.36, 0.16, 0.44]]).
↳ astype(np.float32))
>>> y = x.median(axis=0, keepdims=False)
>>> print(y)
(Tensor(shape=[3], dtype=Float32, value= [ 3.79999995e-01,  1.59999996e-01,  4.39999998e-
↳ 01]),
Tensor(shape=[3], dtype=Int64, value= [1, 2, 2]))
```

`mindspore.Tensor.median()`

返回输入的中位数。

返回:

- **y** (Tensor) - 输出中值。

支持平台：

Ascend

`mindspore.Tensor.median(dim=-1, keepdim=False)`

输出指定维度 *dim* 上的中值与其对应的索引。如果 *dim* 为 `None`，则计算 Tensor 中所有元素的中值。

参数：

- **dim** (int, 可选) - 指定计算的轴。默认值：`None`。
- **keepdim** (bool, 可选) - 是否保留 *dim* 指定的维度。默认值：`False`。

返回：

- **y** (Tensor) - 输出中值，数据类型与 *input* 相同。
 - 如果 *dim* 为 `None`，*y* 只有一个元素。
 - 如果 *keepdim* 为 `True`，*y* 的 *shape* 除了在 *dim* 维度上为 1 外，与 *input* 一致。
 - 其他情况下，*y* 比 *input* 缺少 *dim* 指定的维度。
- **indices** (Tensor) - 中值的索引。*shape* 与 *y* 一致，数据类型为 `int64`。

异常：

- **TypeError** - *dim* 不是 `int`。
- **TypeError** - *keepdim* 不是 `bool` 值。
- **ValueError** - *dim* 不在 `[-x.dim, x.dim-1]` 范围内。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[0.57, 0.11, 0.21], [0.38, 0.50, 0.57], [0.36, 0.16, 0.44]]).
↳ astype(np.float32))
>>> y = x.median(dim=0, keepdim=False)
>>> print(y)
(Tensor(shape=[3], dtype=Float32, value= [ 3.79999995e-01,  1.59999996e-01,  4.39999998e-
↳ 01]),
Tensor(shape=[3], dtype=Int64, value= [1, 2, 2]))
```

mindspore.Tensor.t

`mindspore.Tensor.t()`

详情请参考 `mindspore.ops.t()`。

支持平台：

Ascend

mindspore.Tensor.mH

property mindspore.Tensor.mH

访问此属性等价于调用 `self.adjoint()` 方法。

详情请参考 `mindspore.ops.adjoint()`。

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[0. + 0.j, 1. + 1.j], [2. + 2.j, 3. + 3.j]]))
>>> output = x.mH
>>> print(output)
[[0.-0.j 2.-2.j]
 [1.-1.j 3.-3.j]]
```

mindspore.Tensor.min

mindspore.Tensor.min()

返回输入 Tensor 的最小值。

返回：

Tensor，值为输入 Tensor 的最小值，类型与 *input* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output = Tensor.min(x)
>>> print(output)
0.0
```

mindspore.Tensor.min(dim, keepdim=False)

在给定轴上计算输入 Tensor 的最小值，并返回最小值和索引值。

参数：

- **dim** (int) - 指定计算维度。
- **keepdim** (bool, 可选) - 表示是否减少维度，如果为 `True`，输出将与输入保持相同的维度；如果为 `False`，输出将减少维度。默认值：`False`。

返回：

tuple(Tensor)，返回两个元素类型为 Tensor 的 tuple，包含输入 Tensor 沿指定维度 *dim* 的最小值和相应的索引。

- **values** (Tensor) - 输入 Tensor 沿给定维度的最小值，*shape* 和 *index* 相同，数据类型和 *self* 相同。
- **index** (Tensor) - 输入 Tensor 沿给定维度的最小值索引，数据类型为 `int64`。如果 *keepdim* 为 `True`，输出 Tensor 的 *shape* 是 $(self_1, self_2, \dots, self_{dim-1}, 1, self_{dim+1}, \dots, self_N)$ 。否则输出 *shape* 为 $(self_1, self_2, \dots, self_{dim-1}, self_{dim+1}, \dots, self_N)$ 。

异常:

- **TypeError** - 如果 *keepdim* 不是 bool 类型。
- **TypeError** - 如果 *dim* 不是 int 类型。
- **TypeError** - 如果输入 Tensor 的数据类型为 Complex。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output, index = x.min(0, keepdim=True)
>>> print(output, index)
[0.0] [0]
```

`mindspore.Tensor.min(axis=None, keepdims=False, *, initial=None, where=True, return_indices=False)`

返回 Tensor 元素中的最小值或沿 *axis* 轴方向上的最小值。

说明:

- *axis* 为 None 时, *keepdims* 及以后参数均不会生效, 同时索引固定返回 0。

参数:

- **axis** (Union[None, int, list, tuple of ints], 可选) - 轴, 在该轴方向上进行操作。默认情况下, 使用扁平输入。如果该参数为整数元组, 则在多个轴上选择最小值, 而不是在单个轴或所有轴上进行选择。默认值: None。
- **keepdims** (bool, 可选) - 如果这个参数为 True, 被删去的维度保留在结果中, 且维度设为 1。有了这个选项, 结果就可以与输入数组进行正确的广播运算。默认值: False。

关键字参数:

- **initial** (scalar, 可选) - 输出元素的最小值。如果对空切片进行计算, 则该参数必须设置。默认值: None。
- **where** (Tensor[bool], 可选) - 一个 bool 类型的 Tensor, 被广播以匹配数组维度和选择包含在降维中的元素。如果传递了一个非默认值, 则必须提供初始值。默认值: True。
- **return_indices** (bool, 可选) - 是否返回最小值的下标。默认值: False。如果 *axis* 是 'list' 或 'int' 类型的 'tuple', 则必须取值为 False。

返回:

Tensor 或标量, 输入 Tensor 的最小值。如果 *axis* 为 None, 则结果是一个标量值。如果提供了 *axis*, 则结果是 Tensor `ndim - 1` 维度的一个数组。

异常:

- **TypeError** - 参数的数据类型与上述不一致。

参见:

- `mindspore.Tensor.argmax()`: 返回沿轴最大值的索引。
- `mindspore.Tensor.argmin()`: 返回沿轴最小值的索引。

- `mindspore.Tensor.max()` : 返回整个 Tensor 最大值或沿轴最大值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> a = Tensor(np.arange(4).reshape((2, 2)).astype('float32'))
>>> output = Tensor.min(a)
>>> print(output)
0.0
>>> output = Tensor.min(a, axis=0)
>>> print(output)
[0. 1.]
>>> output = Tensor.min(a, axis=0, initial=9, where=Tensor([False]))
>>> print(output)
[9. 9.]
>>> output = Tensor.min(a, axis=0, initial=9, where=Tensor([False, True]))
>>> print(output)
[9. 1.]
>>> value, indices = Tensor.min(a, axis=0, return_indices=True)
>>> print(value)
[0. 1.]
>>> print(indices)
[0 0]
```

mindspore.Tensor.minimum

`mindspore.Tensor.minimum(other)`

详情请参考[`mindspore.ops.minimum\(\)`](#)。

mindspore.Tensor.mm

`mindspore.Tensor.mm(mat2)`

计算两个矩阵的乘积。

如果 *self* 是一个 $(n \times m)$ 的 Tensor, *mat2* 是一个 $(m \times p)$ 的 Tensor, *out* 则会是一个 $(n \times p)$ 的 Tensor。

说明:

- 此函数不能支持广播。若需要可广播的方法, 请参考[`mindspore.ops.matmul\(\)`](#)。
-

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **mat2** (Tensor) - 矩阵相乘的第二个矩阵, *self* 的最后一维度必须和 *mat2* 的第一维度相等。

返回:

Tensor, 输入的矩阵乘积。

异常:

- **TypeError** - *self* 或者 *mat2* 不是一个 Tensor。
- **RuntimeError** - *self* 的最后一维度和 *mat2* 的倒数第二维度不相等。
- **RuntimeError** - *self* 或者 *mat2* 不是 float16、float32 或 bfloat16 类型。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> x1 = ms.Tensor(np.random.rand(2, 3), ms.float32)
>>> x2 = ms.Tensor(np.random.rand(3, 4), ms.float32)
>>> out = x1.mm(x2)
>>> print(out.shape)
(2, 4)
```

mindspore.Tensor.moveaxis

`mindspore.Tensor.moveaxis(source, destination)`

详情请参考[mindspore.ops.moveaxis\(\)](#)。

mindspore.Tensor.movedim

`mindspore.Tensor.movedim(source, destination)`

详情请参考[mindspore.ops.movedim\(\)](#)。

mindspore.Tensor.move_to

`mindspore.Tensor.move_to(to, blocking=True)`

同步或异步的方式将 Tensor 拷贝到目标设备上, 默认同步方式。只支持 PyNative 模式。

参数:

- **to** (str) - 字符串类型, 取值为 "Ascend"、"GPU"、"CPU" 其中之一。
- **blocking** (bool) - 同步或者异步拷贝方式, **blocking** 默认为 True, 即同步拷贝。

返回:

存储在目标设备上的新 Tensor, 与原 Tensor 有相同的 shape 和 type。

异常:

- **ValueError** - 如果 *blocking* 不是 bool 类型。
- **ValueError** - 如果 *to* 不是 "Ascend"、"GPU"、"CPU" 其中之一。
- **ValueError** - 如果执行模式不是 PyNative 模式。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> x = ms.Tensor([1, 2, 3], ms.int64)
>>> new_tensor = x.move_to("CPU")
```

mindspore.Tensor.msort

`mindspore.Tensor.msort()`

详情请参考 `mindspore.ops.msort()`。

mindspore.Tensor.mT

property `mindspore.Tensor.mT`

返回将最后两个维度交换的 `Tensor`。

访问 `x.mT` 属性等价于调用 `x.swapaxes(-2, -1)` 方法。

详情请参考 `mindspore.Tensor.swapaxes()`。

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.ones((2, 3, 4)))
>>> output = x.mT
>>> print(output.shape)
(2, 4, 3)
```

mindspore.Tensor.mul

`mindspore.Tensor.mul(other)`

详情请参考 `mindspore.ops.mul()`。

mindspore.Tensor.mul_

`mindspore.Tensor.mul_(other)`

两个 `Tensor` 逐元素相乘。

$$out_i = tensor_i * other_i$$

警告: 这是一个实验性 API, 后续可能修改或删除。

说明:

- 当两个输入具有不同的 `shape` 时, `other` 必须能被广播成 `self`。

- *self* 和 *other* 不能同时为 bool 类型。[True, Tensor(True, bool_), Tensor(np.array([True]), bool_)] 等都为 bool 类型。

参数:

- **other** (Union[Tensor, number.Number, bool]) - *other* 可以是 number.Number 或 bool, 也可以是数据类型为 number.Number 或 bool 的 Tensor。

返回:

Tensor, shape 与 *self* 的 shape 相同, 数据类型和 *self* 的类型相同。

异常:

- **TypeError** - *other* 不是 Tensor、number.Number 或 bool。
- **RuntimeError** - *other* 不能被广播成 *self*。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> y = Tensor(np.array([4.0, 5.0, 6.0]), mindspore.float32)
>>> output = x.mul_(y)
>>> print(output)
[ 4. 10. 18.]
>>> print(x)
[ 4. 10. 18.]
```

mindspore.Tensor.multinomial

`mindspore.Tensor.multinomial(num_samples, replacement=True, seed=None)`

详情请参考 `mindspore.ops.multinomial()`。

mindspore.Tensor.multiply

`mindspore.Tensor.multiply(value)`

详情请参考 `mindspore.ops.mul()`。当前接口的参数 *value* 和参考接口中的参数 *other* 实际为同一参数。

mindspore.Tensor.mvlgamma

`mindspore.Tensor.mvlgamma(p)`

详情请参考 `mindspore.ops.mvlgamma()`。

mindspore.Tensor.nan_to_num

`mindspore.Tensor.nan_to_num (nan=None, posinf=None, neginf=None)`

详情请参考[`mindspore.ops.nan\_to\_num\(\)`](#)。

支持平台：

Ascend CPU

mindspore.Tensor.nanmean

`mindspore.Tensor.nanmean (axis=None, keepdims=False, *, dtype=None)`

详情请参考[`mindspore.ops.nanmean\(\)`](#)。

mindspore.Tensor.nanmedian

`mindspore.Tensor.nanmedian (axis=-1, keepdims=False)`

详情请参考[`mindspore.ops.nanmedian\(\)`](#)。

mindspore.Tensor.nansum

`mindspore.Tensor.nansum (dim=None, keepdim=False, *, dtype=None)`

计算输入 Tensor 指定维度元素的和，将非数字 (NaNs) 处理为零。

警告：

- 仅支持 Atlas A2 训练系列产品。
- 这是一个实验性 API，后续可能修改或删除。

参数：

- **dim** (`Union[int, tuple(int)]`, 可选) - 求和的维度。取值范围 `[-rank(self), rank(self))`。默认值：None，对 Tensor 中的所有元素求和。
- **keepdim** (`bool`, 可选) - 输出 Tensor 是否保持维度。默认值：False，不保留维度。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 输出 Tensor 的类型。默认值：None。

返回：

Tensor，输入 Tensor 指定维度的元素和，将非数字 (NaNs) 处理为零。

- 如果 `dim` 为 None，且 `keepdim` 为 False，则输出一个零维 Tensor，表示输入 Tensor 中所有元素的和。
- 如果 `dim` 为 int，值为 2，并且 `keepdim` 为 False，则输出的 shape 为：`(self1, self3, ..., selfR)`。
- 如果 `dim` 为 `tuple(int)` 或 `list(int)`，值为 (2, 3)，并且 `keepdim` 为 False，则输出的 shape 为 `(self1, self4, ..., selfR)`。

异常：

- **TypeError** - `keepdim` 不是 bool 类型。
- **TypeError** - `self` 的数据类型或 `dtype` 是 complex 类型。

- **ValueError** - *dim* 不在 $[-\text{rank}(\text{self}), \text{rank}(\text{self})]$ 。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[float("nan"), 2, 3], [1, 2, float("nan")]]), mindspore.float32)
>>> output1 = x.nansum(dim=0, keepdim=False, dtype=mindspore.float32)
>>> output2 = x.nansum(dim=0, keepdim=True, dtype=mindspore.float32)
>>> print(output1)
[1. 4. 3.]
>>> print(output2)
[[1. 4. 3.]]
```

`mindspore.Tensor.nansum(axis=None, keepdims=False, *, dtype=None)`

计算输入 `Tensor` 指定维度元素的和，将非数字 (NaNs) 处理为零。

参数：

- **axis** (`Union[int, tuple(int)]`, 可选) - 求和的维度。假设 *self* 的秩为 *r*，取值范围 $[-r, r)$ 。默认值：None，对 `Tensor` 中的所有元素求和。
- **keepdims** (`bool`, 可选) - 输出 `Tensor` 是否保持维度。默认值：False，不保留维度。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 输出 `Tensor` 的类型。默认值：None。

返回：

`Tensor`，输入 `Tensor` 指定维度的元素和，将非数字 (NaNs) 处理为零。

- 如果 *axis* 为 None，且 *keep_dims* 为 False，则输出一个零维 `Tensor`，表示输入 `Tensor` 中所有元素的和。
- 如果 *axis* 为 int，值为 2，并且 *keep_dims* 为 False，则输出的 shape 为：(*self*₁, *self*₃, ..., *self*_R)。
- 如果 *axis* 为 `tuple(int)` 或 `list(int)`，值为 (2, 3)，并且 *keep_dims* 为 False，则输出的 shape 为 (*self*₁, *self*₄, ..., *self*_R)。

异常：

- **TypeError** - *keepdims* 不是 bool 类型。
- **TypeError** - *self* 的数据类型或 *dtype* 是 complex 类型。
- **ValueError** - *axis* 的范围不在 $[-r, r)$ ，*r* 表示 *self* 的秩。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[float("nan"), 2, 3], [1, 2, float("nan")]]), mindspore.float32)
>>> output1 = x.nansum(axis=0, keepdims=False, dtype=mindspore.float32)
>>> output2 = x.nansum(axis=0, keepdims=True, dtype=mindspore.float32)
```

(下页继续)

(续上页)

```
>>> print(output1)
[1. 4. 3.]
>>> print(output2)
[[1. 4. 3.]]
```

mindspore.Tensor.narrow

mindspore.Tensor.narrow (*dim, start, length*)

沿着指定的轴，从指定的起始位置获取指定长度的 Tensor。

参数：

- **dim** (int) - 指定的轴。
- **start** (Union[int, Tensor]) - 指定的起始位置。
- **length** (int) - 指定长度。

返回：

output (Tensor) - narrow 后的 Tensor。

异常：

- **ValueError** - *dim* 值超出范围 [-self.ndim, self.ndim)。
- **ValueError** - *start* 值超出范围 [-self.shape[dim], self.shape[dim]]。
- **ValueError** - *length* 值超出范围 [0, self.shape[dim]-start]。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.int32)
>>> output = x.narrow(0, 0, 2)
>>> print(output)
[[ 1 2 3]
 [ 4 5 6]]
>>> output = x.narrow(1, 1, 2)
>>> print(output)
[[ 2 3]
 [ 5 6]
 [ 8 9]]
```

mindspore.Tensor.nbytes

property mindspore.Tensor.nbytes

返回 Tensor 占用的总字节数。

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.nbytes
>>> print(output)
32
```

mindspore.Tensor.ndim

property mindspore.Tensor.ndim

返回 Tensor 维度的数量。

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.ndim
>>> print(output)
2
```

mindspore.Tensor.ndimension

mindspore.Tensor.ndimension()

mindspore.Tensor.ndim 的别名。

mindspore.Tensor.ne

mindspore.Tensor.ne(*other*)

mindspore.Tensor.not_equal() 的别名。

mindspore.Tensor.neg

mindspore.Tensor.neg()

详情请参考 *mindspore.ops.neg()*。

mindspore.Tensor.negative

`mindspore.Tensor.negative()`
`mindspore.Tensor.neg()` 的别名。

mindspore.Tensor.nelement

`mindspore.Tensor.nelement()`
`mindspore.Tensor.numel()` 的别名。

mindspore.Tensor.new_ones

`mindspore.Tensor.new_ones(size, dtype=None)`
返回一个大小为 `size` 的 Tensor，填充值为 1。

参数：

- **size** (`Union[int, tuple(int), list(int)]`) - 定义输出的 shape。
- **dtype** (`mindspore.dtype`, 可选) - 输出的数据类型。默认值：None，返回的 Tensor 使用 *self* 相同的数据类型。

返回：

Tensor，shape 和数据类型由输入定义，填充值为 1。

异常：

- **TypeError** - 如果 `size` 不是一个 int，或元素为 int 的元组/列表。
- **TypeError** - 如果 `dtype` 不是一个 MindSpore 的数据类型。
- **ValueError** - 如果 `size` 包含负数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(), mindspore.int32)
>>> x.new_ones((2, 3))
Tensor(shape=[2, 3], dtype=Int32, value=
[[1, 1, 1],
 [1, 1, 1]])
```

`mindspore.Tensor.new_zeros`

`mindspore.Tensor.new_zeros` (*size*, *dtype=None*)

返回一个大小为 *size* 的 Tensor，填充值为 0。

参数：

- **size** (`Union[int, tuple(int), list(int)]`) - 定义输出的 shape。
- **dtype** (`mindspore.dtype`, 可选) - 输出的数据类型。默认值: `None`，返回的 Tensor 使用与 *self* 相同的数据类型。

返回：

Tensor，shape 和 dtype 由输入定义，填充值为 0。

异常：

- **TypeError** - 如果 *size* 不是一个 int，或元素为 int 的元组/列表。
- **TypeError** - 如果 *dtype* 不是一个 MindSpore 的数据类型。
- **ValueError** - 如果 *size* 包含负数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(), mindspore.int32)
>>> x.new_zeros((2, 3))
Tensor(shape=[2, 3], dtype=Int32, value=
[[0, 0, 0],
 [0, 0, 0]])
```

`mindspore.Tensor.nextafter`

`mindspore.Tensor.nextafter` (*other*)

详情请参考 `mindspore.ops.nextafter()`。

`mindspore.Tensor.nonzero`

`mindspore.Tensor.nonzero` (*, *as_tuple=False*)

返回所有非零元素下标位置。

说明: *self* 的秩。

- Ascend: 其秩可以等于 0，O2 模式除外。
- CPU/GPU: 其秩应大于等于 1。

关键字参数：

- **as_tuple** (bool, 可选) - 是否以 tuple 形式输出。如果为 `False`，输出 Tensor，默认值: `False`。如果为 `True`，输出 `Tuple[Tensor]`，只支持 Ascend。

返回：

- 当 `as_tuple=False`，输出 `Tensor`，维度为 2，类型为 `int64`，表示输入中所有非零元素的下标。
- 当 `as_tuple=True`，输出 `Tuple[Tensor]`，类型为 `int64`，长度为输入张量的维度，每一元素是输入张量在该维度下所有非零元素的下标的 1D 张量。

异常：

- **TypeError** - 如果 `self` 不是 `Tensor`。
- **TypeError** - 如果 `as_tuple` 不是 `bool`。
- **RuntimeError** - 在 CPU 或者 GPU 或者 Ascend 的 O2 模式中，如果 `self` 的维度为 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 0], [-5, 0]]), mindspore.int32)
>>> output = x.nonzero()
>>> print(output)
[[0 0 0]
 [0 1 0]]
>>> x = Tensor(np.array([1, 0, 2, 0, 3]), mindspore.int32)
>>> output = x.nonzero(as_tuple=False)
>>> print(output)
[[0]
 [2]
 [4]]
>>> x = Tensor(np.array([[1, 0], [-5, 0]]), mindspore.int32)
>>> output = x.nonzero(as_tuple=True)
>>> print(output)
(Tensor(shape=[2], dtype=Int64, value=[0, 0]),
 Tensor(shape=[2], dtype=Int64, value=[0, 1]),
 Tensor(shape=[2], dtype=Int64, value=[0, 0]))
>>> x = Tensor(np.array([1, 0, 2, 0, 3]), mindspore.int32)
>>> output = x.nonzero(as_tuple=True)
>>> print(output)
(Tensor(shape=[3], dtype=Int64, value=[0, 2, 4]), )
```

`mindspore.Tensor.norm`

`mindspore.Tensor.norm` (`ord=None`, `dim=None`, `keepdim=False`, `*`, `dtype=None`)

详情请参考 `mindspore.ops.norm()`。

mindspore.Tensor.normal_

`mindspore.Tensor.normal_ (mean=0, std=1, *, generator=None)`

使用随机数原地更新 Tensor，且随机数的采样服从由参数 *mean* 和 *std* 所构成的正态分布。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **mean** (number, 可选) - 正态分布的均值，默认值：0。
- **std** (number, 可选) - 正态分布的标准差，默认值：1。

关键字参数：

- **generator** (*mindspore.Generator*, 可选) - 伪随机数生成器。默认值：None，使用默认伪随机数生成器。

返回：

返回一个 Tensor，该 Tensor 由服从正态分布的随机数填充，且 *type* 和 *shape* 与原 Tensor 一致。

异常：

- **TypeError** - *mean* 或 *std* 的 dtype 不是 number，即数据类型不是 bool、int、float 或 complex 之一。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> x = mindspore.Tensor(np.array([[1, 2], [3, 4]]), dtype=mindspore.float32)
>>> output = x.normal_()
>>> print(output)
[[0.2788825 1.3305743]
 [1.244194 1.16303174]]
```

mindspore.Tensor.not_equal

`mindspore.Tensor.not_equal (other)`

详情请参考 *mindspore.ops.ne()*。

mindspore.Tensor.numel

`mindspore.Tensor.numel()`

详情请参考 *mindspore.ops.numel()*。

mindspore.Tensor.numpy

`mindspore.Tensor.numpy()`
`mindspore.Tensor.asnumpy()` 的别名。

mindspore.Tensor.orgqr

`mindspore.Tensor.orgqr(input2)`
详情请参考`mindspore.ops.orgqr()`。

mindspore.Tensor.ormqr

`mindspore.Tensor.ormqr(input2, input3, left=True, transpose=False)`
详情请参考`mindspore.ops.ormqr()`，其中参数 `input2` 和 `input3` 分别对应`mindspore.ops.ormqr()` 的参数 `tau` 和 `other`。

mindspore.Tensor.outer

`mindspore.Tensor.outer(vec2)`
详情请参考`mindspore.ops.outer()`。

mindspore.Tensor.permute

`mindspore.Tensor.permute(*axis)`
`Tensor.permute` 支持 `axis` 以不定数量的位置参数形式入参自动解包，与`mindspore.ops.permute()` 的入参有微小的差异。详情请参考`mindspore.ops.permute()`。

mindspore.Tensor.positive

`mindspore.Tensor.positive()`
详情请参考`mindspore.ops.positive()`。

mindspore.Tensor.pow

`mindspore.Tensor.pow(exponent)`
详情请参考`mindspore.ops.pow()`。

mindspore.Tensor.prod

`mindspore.Tensor.prod(dim=None, keepdim=False, dtype=None)`

默认情况下，使用指定维度的所有元素的乘积代替该维度的其他元素，以移除该维度。也可仅缩小该维度大小至 1。
keepdim 控制输出和输入的维度是否相同。

参数：

- **dim** (`Union[int, tuple(int), list(int), Tensor]`, 可选) - 要减少的维度。默认值: `None`，缩小所有维度。只允许常量值。假设 *self* 的秩为 *r*，取值范围 `[-r,r]`。
- **keepdim** (`bool`, 可选) - 如果为 `True`，则保留缩小的维度，大小为 1。否则移除维度。默认值: `False`。
- **dtype** (`mindspore.dtype`, 可选) - 期望输出 `Tensor` 的类型。默认值: `None`。

返回：

`Tensor`。

- 如果 *dim* 为 `None`，且 *keepdim* 为 `False`，则输出一个零维 `Tensor`，表示输入 `Tensor` 中所有元素的乘积。
- 如果 *dim* 为 `int`，取值为 1，并且 *keepdim* 为 `False`，则输出的 `shape` 为 `(self0, self2, ..., selfR)`。
- 如果 *dim* 为 `tuple(int)` 或 `list(int)`，取值为 `(1, 2)`，并且 *keepdim* 为 `False`，则输出 `Tensor` 的 `shape` 为 `(self0, self3, ..., selfR)`。
- 如果 *dim* 为一维 `Tensor`，例如取值为 `[1, 2]`，并且 *keepdim* 为 `False`，则输出 `Tensor` 的 `shape` 为 `(self0, self3, ..., selfR)`。

异常：

- **TypeError** - *dim* 不是以下数据类型之一: `int`、`Tuple`、`list` 或 `Tensor`。
- **TypeError** - *keepdim* 不是 `bool` 类型。
- **ValueError** - *dim* 超出范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> output = Tensor.prod(x, 1, True)
>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by multiplying all elements in the dimension.
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
mindspore.float32)
>>> output = Tensor.prod(x)
>>> print(output)
2.2833798e+33
>>> print(output.shape)
()
>>> # case 2: Reduces a dimension along axis 0.
>>> output = Tensor.prod(x, 0, True)
```

(下页继续)

(续上页)

```

>>> print(output)
[[[ 28.  28.  28.  28.  28.  28.]
  [ 80.  80.  80.  80.  80.  80.]
  [162. 162. 162. 162. 162. 162.]]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = Tensor.prod(x, 1, True)
>>> print(output)
[[[ 6.  6.  6.  6.  6.  6.]
  [120. 120. 120. 120. 120. 120.]
  [504. 504. 504. 504. 504. 504.]]]
>>> # case 4: Reduces a dimension along axis 2.
>>> output = Tensor.prod(x, 2, True)
>>> print(output)
[[[1.00000e+00]
  [6.40000e+01]
  [7.29000e+02]]
 [[4.09600e+03]
  [1.56250e+04]
  [4.66560e+04]]
 [[1.17649e+05]
  [2.62144e+05]
  [5.31441e+05]]]

```

`mindspore.Tensor.prod(axis=None, keep_dims=False, dtype=None)`

详情请参考[mindspore.ops.prod\(\)](#)。

mindspore.Tensor.ptp

`mindspore.Tensor.ptp(axis=None, keepdims=False)`

该函数名称是“peak to peak”的缩写。计算沿着 axis 的最大值与最小值的差值。

说明：不支持 NumPy 参数 `dtype` 和 `out`。

参数：

- **axis** (Union[None, int, tuple(int)]) - 轴，在轴方向上可以计算范围。默认计算扁平数组的方差。默认值：None。
- **keepdims** (bool) - 如果设为 True，被删去的维度保留在结果中，且维度大小设为 1。有了这个选项，结果将针对输入数组正确传递。默认值为 False。

返回：

Tensor。

异常：

- **TypeError** - `self` 不是 Tensor，或者 `axis` 和 `keepdims` 具有前面未指定的类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor
>>> x = Tensor([[4.0, 9.0, 2.0, 10.0], [6.0, 9.0, 7.0, 12.0]]).astype("float32")
>>> print(x.ptp(axis=1))
[8. 6.]
>>> print(x.ptp(axis=0))
[2. 0. 5. 2.]
```

mindspore.Tensor.rad2deg

mindspore.Tensor.**rad2deg**()

详情请参考[mindspore.ops.rad2deg\(\)](#)。

mindspore.Tensor.random_

mindspore.Tensor.**random_**(*from_=0, to=None, *, generator=None*)

在区间 [*from_*, *to* - 1] 内生成服从均匀分布的离散均匀分布随机数，原地更新输入 Tensor。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **from_** (Union[number.Number, Tensor], 可选) - 均匀分布的下界，可以是一个标量值或只有单个元素的任意维度的 Tensor，默认值：0。
- **to** (Union[number.Number, Tensor], 可选) - 均匀分布的上界，可以是一个标量值或只有单个元素的任意维度的 Tensor。默认为输入数据类型的取值上限。默认值：None。

关键字参数：

- **generator** ([mindspore.Generator](#), 可选) - 伪随机数生成器。默认值：None，使用默认伪随机数生成器。

返回：

返回输入 tensor。

异常：

- **TypeError** - *from_* 或 *to* 不是整型。
- **RuntimeError** - 如果 *from_* 大于等于 *to*。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor
>>> a = Tensor([[2, 3, 4], [1, 2, 3]])
>>> from_ = 0
>>> to = 5
>>> print(a.random_(from_, to).shape)
(2, 3)
```

mindspore.Tensor.random_categorical

`mindspore.Tensor.random_categorical(num_sample, seed=0, dtype=mstype.int64)`

详情请参考[`mindspore.ops.random\_categorical\(\)`](#)。

mindspore.Tensor.ravel

`mindspore.Tensor.ravel()`

返回一个展开的一维 Tensor。

返回：

一维 Tensor，含有与输入相同的元素。

参见：

- [`mindspore.Tensor.reshape\(\)`](#)：为 Tensor 指定新的 shape，不改变其数据。
- [`mindspore.Tensor.flatten\(\)`](#)：返回展开为一维的 Tensor 副本。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.ones((2,3,4), dtype=np.float32))
>>> output = x.ravel()
>>> print(output.shape)
(24,)
```

mindspore.Tensor.real

`mindspore.Tensor.real()`

详情请参考[`mindspore.ops.real\(\)`](#)。

mindspore.Tensor.reciprocal

`mindspore.Tensor.reciprocal()`

详情请参考[`mindspore.ops.reciprocal\(\)`](#)。

`mindspore.Tensor.register_hook`

`mindspore.Tensor.register_hook(hook)`

设置 `Tensor` 对象的反向 `hook` 函数。

说明:

- `hook` 必须有如下代码定义: `grad` 是反向传递给 `Tensor` 对象的梯度。用户可以在 `hook` 中打印梯度数据或者返回新的输出梯度。
- `hook` 返回新的梯度输出, 不能不设置返回值: `hook(grad) -> New grad_output`。
- 高阶求导不支持 `register_hook`。
- 静态图模式下需满足如下约束:
 - `hook` 同样需满足静态图模式下的语法约束。
 - 不支持在图内对 `hook` 进行删除。
 - 不支持在 `Tensor` 被使用之后再次对 `Tensor` 注册 `hook`。
 - 不支持在图内对 `Tensor` 注册多个 `hook`。
 - 图内对 `Tensor` 注册 `hook` 将返回 `Tensor` 本身。

参数:

- **hook** (function) - 捕获 `Tensor` 反向传播时的梯度, 并输出或更改该梯度的 `hook` 函数。

返回:

返回与该 `hook` 函数对应的 `handle` 对象。可通过调用 `handle.remove()` 来删除添加的 `hook` 函数。

异常:

- **TypeError** - 如果 `hook` 不是 Python 函数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> ms.set_context(mode=ms.PYNATIVE_MODE)
>>> def hook_fn(grad):
...     return grad * 2
...
>>> def hook_test(x, y):
...     z = x * y
...     z.register_hook(hook_fn)
...     z = z * y
...     return z
...
>>> ms_grad = ms.grad(hook_test, grad_position=(0,1))
>>> output = ms_grad(Tensor(1, ms.float32), Tensor(2, ms.float32))
>>> print(output)
(Tensor(shape=[], dtype=Float32, value=8), Tensor(shape=[], dtype=Float32, value=6))
```

mindspore.Tensor.remainder

`mindspore.Tensor.remainder(other)`

逐元素计算 *self* 除以 *other* 后的余数。结果与除数同号且绝对值小于除数的绝对值。

支持广播和隐式数据类型提升。

```
remainder(input, other) == input - input.div(other, rounding_mode="floor") * other
```

说明：输入不支持复数类型。至少一个输入为 `Tensor`，且不能都为 `bool` 型 `Tensor`。除数 *self* 是数据类型为 `number` 或 `bool_` 的 `Tensor`。

参数：

- **other** (`Union[Tensor, numbers.Number, bool]`) - 被除数为数值型、`bool` 或数据类型为数值型或 `bool` 的 `Tensor`。当除数是 `Tensor` 时，则被除数是数值型、`bool` 或数据类型为数值型或 `bool` 的 `Tensor`。

返回：

`Tensor`，经过隐式类型提升和广播。

异常：

- **TypeError** - 如果 *self* 和 *other* 不是以下类型之一：(`Tensor`, `Tensor`), (`Tensor`, `Number`), (`Tensor`, `bool`), (`Number`, `Tensor`) 或 (`bool`, `Tensor`)。
- **ValueError** - 如果 *self* 和 *other* 不能被广播。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([-4.0, 5.0, 6.0]).astype(np.float32))
>>> y = Tensor(np.array([3.0, 2.0, 3.0]).astype(np.float64))
>>> output = x.remainder(y)
>>> print(output)
[2.  1.  0.]
```

`mindspore.Tensor.remainder(divisor)`

逐元素计算第一个元素除以第二个元素的余数。

self 和 *divisor* 的输入遵守隐式类型转换规则，以使数据类型一致。输入必须是两个 `Tensor` 或者一个 `Tensor` 和一个 `Scalar`。当输入是两个 `Tensor` 时，两个 `dtype` 都不能是 `bool` 类型，`shape` 可以广播。当输入是 `Tensor` 和 `Scalar` 时，这个 `Scalar` 只能是常数。

```
remainder(input, other) == input - input.div(other, rounding_mode="floor") * other
```

警告：

- 当输入元素超过 2048 时，可能会有精度问题。
- 在 Ascend 和 CPU 上的计算结果可能不一致。

- 如果 shape 表示为 (D1,D2...Dn), 那么 $D1 * D2 * \dots * DN \leq 1000000$, $n \leq 8$ 。

说明: 第一个输入 *self* 为 dtype 是 Number 的 Tensor。

参数:

- **divisor** (Union[Tensor, numbers.Number, bool]) - 当第一个输入是一个 Tensor 时, 第二个输入可以是 Number、bool 或者 dtype 是 Number 的 Tensor。

返回:

Tensor, 具有和其中一个输入广播后相同的 shape, 数据类型是两个输入中精度较高的数据类型。

异常:

- **TypeError** - *self* 和 *divisor* 的类型不是 Tensor、Number 或 bool。
- **ValueError** - *self* 和 *divisor* 的 shape 不能广播为对方的 shape。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([-4.0, 5.0, 6.0]).astype(np.float16))
>>> y = Tensor(np.array([3.0, 2.0, 3.0]).astype(np.float16))
>>> output = x.remainder(divisor=y)
>>> print(output)
[2.  1.  0.]
```

mindspore.Tensor.renorm

mindspore.Tensor.**renorm**(*p, axis, maxnorm*)

详情请参考 [mindspore.ops.renorm\(\)](#)。

mindspore.Tensor.repeat

mindspore.Tensor.**repeat**(**repeats*)

对张量的每一维度按照指定的重复次数复制其中的元素。

此方法会拷贝此张量的数据。

输出张量的 shape 可以描述如下, 其中 n 为 *repeats* 中的元素个数:

$$shape_i = \begin{cases} repeats_i * input.shape_i & \text{if } 0 \leq i < input.rank \\ repeats_i & \text{if } input.rank \leq i < n \end{cases}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

说明：如需对单个轴的每个元素单独指定重复次数，请参考 `mindspore.Tensor.repeat_interleave()`。

参数：

- ***repeats** (int) - *self* 在每个维度上的重复次数，应当为非负数，且 1 表示在该维度上保持不变。*repeats* 的元素个数应当不小于 *self* 的维度数。当 *self* 的维度数小于 *repeats* 的元素个数时，*self* 将被广播到与 *repeats* 元素个数相同的维度（如样例所示）。

返回：

Tensor，按照指定的重复次数拷贝元素后的新张量。

异常：

- **RuntimeError** - *repeats* 的元素个数少于 *self* 的维度数，或 *repeats* 中有元素为负数。
- **RuntimeError** - *repeats* 的元素个数或 *self* 的维度数超过 8。
- **TypeError** - *repeats* 参数类型错误。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor
>>> a = Tensor([[0, 1, 2], [3, 4, 5]])
>>> print(a.repeat(3, 2))
[[0 1 2 0 1 2]
 [3 4 5 3 4 5]
 [0 1 2 0 1 2]
 [3 4 5 3 4 5]
 [0 1 2 0 1 2]
 [3 4 5 3 4 5]]
>>> print(a.repeat(2, 1, 3)) # a is treated as a shape [1, 2, 3]
[[[0 1 2 0 1 2 0 1 2]
   [3 4 5 3 4 5 3 4 5]]
 [[0 1 2 0 1 2 0 1 2]
   [3 4 5 3 4 5 3 4 5]]]
```

`mindspore.Tensor.repeat(repeats)`

对张量的每一维度按照指定的重复次数复制其中的元素。

此方法会拷贝此张量的数据。

除了由接受变长 int 型参数，变为接受 1 个 list 或 tuple 型参数外，其他操作与使用 **repeats* 参数的重载保持一致。

输出张量的 shape 可以描述如下，其中 *n* 为 *repeats* 中的元素个数：

$$shape_i = \begin{cases} repeats_i * input.shape_i & \text{if } 0 \leq i < input.rank \\ repeats_i & \text{if } input.rank \leq i < n \end{cases}$$

警告：这是一个实验性 API，后续可能修改或删除。

说明：如需对单个轴的每个元素单独指定重复次数，请参考 `mindspore.Tensor.repeat_interleave()`。

参数：

- **repeats** (Union[tuple[int], list[int]]) - *self* 在每个维度上的重复次数，应当为非负数，且 1 表示在该维度上保持不变。*repeats* 的元素个数应当不小于 *self* 的维度数。当 *self* 的维度数小于 *repeats* 的元素个数时，*self* 将被广播到与 *repeats* 元素个数相同的维度（如样例所示）。

返回：

Tensor，按照指定的重复次数拷贝元素后的新张量。

异常：

- **RuntimeError** - *repeats* 的元素个数少于 *self* 的维度数，或 *repeats* 中有元素为负数。
- **RuntimeError** - *repeats* 的元素个数或 *self* 的维度数超过 8。
- **TypeError** - *repeats* 参数类型错误。

参见：

- `mindspore.Tensor.reshape()`：为 Tensor 指定新的 shape，不更改 Tensor 的数据。
- `mindspore.Tensor.resize()`：就地改变 Tensor 的 shape 和 size。
- `mindspore.Tensor.repeat_interleave()`：将一个 Tensor 指定轴上的每个元素按指定次数进行重复。
- `mindspore.Tensor.tile()`：对一个 Tensor 在每个轴上重复指定次数，但对参数 *repeats* 的个数没有要求。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor
>>> a = Tensor([[0, 1, 2], [3, 4, 5]])
>>> print(a.repeat([3, 2]))
[[0 1 2 0 1 2]
 [3 4 5 3 4 5]
 [0 1 2 0 1 2]
 [3 4 5 3 4 5]
 [0 1 2 0 1 2]
 [3 4 5 3 4 5]]
>>> print(a.repeat(repeats=(2, 1, 3))) # a is treated as a shape [1, 2, 3]
[[[0 1 2 0 1 2 0 1 2]
   [3 4 5 3 4 5 3 4 5]]
 [[0 1 2 0 1 2 0 1 2]
   [3 4 5 3 4 5 3 4 5]]]
```

mindspore.Tensor.repeat_interleave

`mindspore.Tensor.repeat_interleave(repeats, dim=None, *, output_size=None)`

沿着轴重复 Tensor 的元素，类似 `mindspore.numpy.repeat()`。

警告：仅 Atlas A2 训练系列产品支持。

说明：进行重复操作的入参 Tensor，类型必须为 float16, float32, int8, uint8, int16, int32 或者 int64。

参数：

- **repeats** (Union[int, tuple, list, Tensor]) - 指定复制次数，为正数。
- **dim** (int, 可选) - 指定复制轴，默认值：None。如果为 None，输入 Tensor 会被展平并且输出结果也会被展平。

关键字参数：

- **output_size** (int, 可选) - 给定轴的总输出大小 (即参数 repeats 元素之和)，默认值：None。

返回：

Tensor，值沿指定轴复制。如果输入的 shape 为 $(s_1, s_2, \dots, s_n)$ ，轴为 i ，则输出的 shape 为 $(s_1, s_2, \dots, s_i * repeats, \dots, s_n)$ 。输出的数据类型与输入相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input1 = Tensor(np.array([[0, 1, 2], [3, 4, 5]]), mindspore.int32)
>>> output1 = input1.repeat_interleave(repeats=2, dim=0, output_size=None)
>>> input2 = Tensor(np.array([[1, 2], [3, 4]]), mindspore.int32)
>>> output2 = input2.repeat_interleave(Tensor(np.array([1, 2])), dim=0, output_size=None)
>>> print(output1)
>>> print(output2)
[[0 1 2]
 [0 1 2]
 [3 4 5]
 [3 4 5]]
[[1 2]
 [3 4]
 [3 4]]
```

`mindspore.Tensor.repeat_interleave(repeats, dim=None)`

沿着轴重复 Tensor 的元素，类似 `mindspore.numpy.repeat()`。

说明：进行重复操作的入参 Tensor，类型必须为 float16, float32, int8, uint8, int16, int32 或者 int64。

参数：

- **repeats** (Union[int, tuple, list, Tensor]) - 指定复制次数，为正数。
- **dim** (int, 可选) - 指定复制轴，默认值：None。如果为 None，输入 Tensor 会被展平并且输出结果也会被展平。

返回：

Tensor，值沿指定轴复制。如果输入的 shape 为 $(s_1, s_2, \dots, s_n)$ ，轴为 i ，则输出的 shape 为 $(s_1, s_2, \dots, s_i * repeats, \dots, s_n)$ 。输出的数据类型与输入相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input1 = Tensor(np.array([[0, 1, 2], [3, 4, 5]]), mindspore.int32)
>>> output1 = input1.repeat_interleave(repeats=2, dim=0)
>>> input2 = Tensor(np.array([[1, 2], [3, 4]]), mindspore.int32)
>>> output2 = input2.repeat_interleave(Tensor(np.array([1, 2])), dim=0)
>>> print(output1)
>>> print(output2)
[[0 1 2]
 [0 1 2]
 [3 4 5]
 [3 4 5]]
[[1 2]
 [3 4]
 [3 4]]
```

mindspore.Tensor.reshape

`mindspore.Tensor.reshape(*shape)`

详情请参考 [mindspore.ops.reshape\(\)](#)。

mindspore.Tensor.reshape_as

`mindspore.Tensor.reshape_as(other)`

不改变数据的情况下，将 Tensor 的 shape 改为 *other* 的 shape。

参数：

- **other** (Tensor) - 计算结果的 shape 与 *other* 的 shape 一致。

返回：

Tensor，与 *other* 的 shape 一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor([[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]], dtype=ms.float32)
>>> y = Tensor(np.arange(6).reshape(3,2))
>>> output = x.reshape_as(y)
>>> print(output)
[[-0.1  0.3]
 [ 3.6  0.4]
 [ 0.5 -3.2]]
```

mindspore.Tensor.resize

`mindspore.Tensor.resize(*new_shape)`

将 Tensor 改为输入的新 shape，并将不足的元素补 0。

说明：此方法不更改输入数组的大小，也不返回 NumPy 中的任何内容，而是返回一个具有输入大小的新 Tensor。不支持 Numpy 参数 *refcheck*。

参数：

- **new_shape** (Union[ints, tuple of ints]) - 指定 Tensor 的新 shape。

返回：

Tensor。

参见：

- `mindspore.Tensor.reshape()`：为 Tensor 指定新的 shape，不改变其数据。
- `mindspore.Tensor.repeat()`：重复 Tensor 的元素。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6]], dtype=np.float32))
>>> y = x.resize(3, 3)
>>> print(y)
[[1. 2. 3.]
 [4. 5. 6.]
 [0. 0. 0.]]
>>> y = x.resize(2, 2)
>>> print(y)
[[1. 2.]
 [3. 4.]]
```

mindspore.Tensor.reverse

`mindspore.Tensor.reverse(axis)`

详情请参考`mindspore.ops.flip()`。 `Tensor.reverse` 中的 `axis` 参数与`mindspore.ops.flip()` 中的 `dims` 参数等价。

mindspore.Tensor.reverse_sequence

`mindspore.Tensor.reverse_sequence(seq_lengths, seq_dim=0, batch_dim=0)`

详情请参考`mindspore.ops.reverse_sequence()`。

mindspore.Tensor.roll

`mindspore.Tensor.roll(shifts, dims)`

详情请参考`mindspore.ops.roll()`。

mindspore.Tensor.rot90

`mindspore.Tensor.rot90(k, dims)`

详情请参考`mindspore.ops.rot90()`。

mindspore.Tensor.round

`mindspore.Tensor.round(decimals=0)`

详情请参考`mindspore.ops.round()`。

mindspore.Tensor.rsqrt

`mindspore.Tensor.rsqrt()`

详情请参考`mindspore.ops.rsqrt()`。

mindspore.Tensor.scatter

`mindspore.Tensor.scatter(dim, index, src)`

根据指定索引将 `src` 中的值更新到 `self` 中，并返回输出。对于一个 3D 的 `Tensor`，输出形式如下所示：

```

output[index[i][j][k]][j][k] = src[i][j][k]    # if dim == 0
output[i][index[i][j][k]][k] = src[i][j][k]    # if dim == 1
output[i][j][index[i][j][k]] = src[i][j][k]    # if dim == 2

```

说明：如果 `src` 为 `Tensor`，则仅当 `src` 的 `shape` 和 `index` 的 `shape` 相同时，支持求反向梯度。输入 `Tensor self` 的秩必须至少为 1。

参数:

- **dim** (int) - 要进行更新操作的轴。取值范围是 $[-r, r)$, 其中 r 是 *self* 的秩。
- **index** (Tensor) - 输入 Tensor 的索引, 数据类型为 int32 或 int64 的正整数。其 rank 必须和 *self* 一致。取值范围是 $[-s, s)$, 这里的 s 是 *self* 在 *axis* 指定轴的 size。
- **src** (Tensor, float) - 指定对 *self* 进行更新操作的数据。可以为 Tensor, 此时其数据类型必须与输入 *self* 的数据类型相同。也可以是个 float 类型的标量。

返回:

Tensor, shape 和数据类型与输入 *self* 相同。

异常:

- **TypeError** - *index* 的数据类型既不是 int32, 也不是 int64。
- **ValueError** - *self*、*index* 和 *src* 中, 任意一者的秩小于 1。
- **ValueError** - *src* 的秩和 *self* 的秩不一样。
- **TypeError** - *self* 的数据类型和 *src* 的数据类型不一致。
- **RuntimeError** - *index* 中存在负数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> input = Tensor(np.array([[1, 2, 3, 4, 5]]), dtype=ms.float32)
>>> src = Tensor(np.array([[8, 8]]), dtype=ms.float32)
>>> index = Tensor(np.array([[2, 4]]), dtype=ms.int64)
>>> out = input.scatter(dim=1, index=index, src=src)
>>> print(out)
[[1. 2. 8. 4. 8.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 0, 0], [2, 2, 2], [4, 4, 4]]), dtype=ms.int64)
>>> out = input.scatter(dim=0, index=index, src=src)
>>> print(out)
[[1. 2. 3. 0. 0.]
 [0. 0. 0. 0. 0.]
 [4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0.]
 [7. 8. 9. 0. 0.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 2, 4], [0, 2, 4], [0, 2, 4]]), dtype=ms.int64)
>>> out = input.scatter(dim=1, index=index, src=src)
>>> print(out)
[[1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6.]
 [7. 0. 8. 0. 9.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

`mindspore.Tensor.scatter` (*axis*, *index*, *src*)

根据指定索引将 *src* 中的值更新到 *self* 中，并返回输出。有关更多详细信息，请参阅 `mindspore.ops.tensor_scatter_elements()`。

说明：如果 *src* 为 Tensor，则仅当 *src* 的 *shape* 和 *index* 的 *shape* 相同时，支持求反向梯度。输入 Tensor *self* 的秩必须至少为 1。

参数：

- **axis** (int) - 要进行更新操作的轴。取值范围是 $[-r, r)$ ，其中 *r* 是 *self* 的秩。
- **index** (Tensor) - 输入 Tensor 的索引，数据类型为 int32 或 int64 的正整数。其 rank 必须和 *self* 一致。取值范围是 $[-s, s)$ ，这里的 *s* 是 *self* 在 *axis* 指定轴的 size。
- **src** (Tensor, float) - 指定对 *self* 进行更新操作的数据。可以为 Tensor，此时其数据类型必须与输入 *self* 的数据类型相同。也可以是个 float 类型的标量。

返回：

Tensor，shape 和数据类型与输入 *self* 相同。

异常：

- **TypeError** - *index* 的数据类型既不是 int32，也不是 int64。
- **ValueError** - *self*、*index* 和 *src* 中，任意一者的秩小于 1。
- **ValueError** - *src* 的秩和 *self* 的秩不一样。
- **TypeError** - *self* 的数据类型和 *src* 的数据类型不一致。
- **RuntimeError** - *index* 中存在负数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> input = Tensor(np.array([[1, 2, 3, 4, 5]]), dtype=ms.float32)
>>> src = Tensor(np.array([[8, 8]]), dtype=ms.float32)
>>> index = Tensor(np.array([[2, 4]]), dtype=ms.int64)
>>> out = input.scatter(axis=1, index=index, src=src)
>>> print(out)
[[1. 2. 8. 4. 8.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 0, 0], [2, 2, 2], [4, 4, 4]]), dtype=ms.int64)
>>> out = input.scatter(axis=0, index=index, src=src)
>>> print(out)
[[1. 2. 3. 0. 0.]
 [0. 0. 0. 0. 0.]
 [4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0.]
 [7. 8. 9. 0. 0.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 2, 4], [0, 2, 4], [0, 2, 4]]), dtype=ms.int64)
```

(下页继续)

(续上页)

```
>>> out = input.scatter(axis=1, index=index, src=src)
>>> print(out)
[[1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6.]
 [7. 0. 8. 0. 9.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

mindspore.Tensor.scatter_

`mindspore.Tensor.scatter_(dim, index, src)`

根据 *index* 使用 *src* 中的值更新当前张量 *self*。

对当前张量 *self* 被 *dim* 选中的维度使用 *index* 进行索引，对其他维度按顺序遍历，将 *src* 中的值更新到 *self* 中，并返回 *self* 自身。此操作是 `mindspore.Tensor.gather()` 的原地更新版本的逆操作。

此操作提供另外 3 个重载，提供对 *reduce* 参数和标量值的支持。

下面看一个三维的例子：

```
self[index[i][j][k]][j][k] = src[i][j][k] # if dim == 0
self[i][j][index[i][j][k]] = src[i][j][k] # if dim == 2
```

警告：

- 如果 *index* 有多个索引指向 *self* 内的同一个位置，则 *self* 中该位置的最终值是不确定的。
- 在 Ascend 平台上，如果 *index* 中元素的值不在 $[-self.shape[dim], self.shape[dim])$ 的范围内，则其行为是不确定的。
- 这是一个实验性 API，后续可能修改或删除。

说明：仅当 *src* 的 shape 和 *index* 的 shape 相同时，支持求 *self* 向 *src* 的反向梯度。

参数：

- **dim** (int) - 要进行操作轴。取值范围是 $[-r, r)$ ，其中 *r* 是 *self* 的秩。
- **index** (Tensor) - 在 *dim* 指定的目标轴上访问 *self* 时使用的索引，数据类型为 int32 或 int64。如果为空 Tensor，则将直接返回，不进行任何操作；否则其 rank 必须和 *self* 一致，且每个元素取值范围是 $[-s, s)$ ，这里的 *s* 是 *self* 在 *dim* 指定轴的大小。
- **src** (Tensor) - 指定对 *self* 进行更新操作的数据。其 rank 与 dtype 必须与 *self* 的相同。

返回：

Tensor，返回被修改后的 *self* 自身。

异常：

- **TypeError** - *self*、*index* 或 *src* 的类型不支持。
- **RuntimeError** - *dim* 的取值超出 $[-r, r)$ 的限制。
- **RuntimeError** - *self* 的秩超过 8。

- **RuntimeError** - 向量 *self* , *index* 或 *src* 的 dtype 不被支持。
- **RuntimeError** - *self* 与 *src* 类型不一致。
- **RuntimeError** - *self* 、*index* 与 *src* 的秩不一致且 *index* 不为空。
- **RuntimeError** - 存在一个维度 *d* 使得 $index.size(d) > src.size(d)$ 。
- **RuntimeError** - 存在一个维度 *d* 使得 $index.size(d) > self.size(d)$ 。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, int64, float32
>>> this_tensor = Tensor([[1, 2], [3, 4]], dtype=float32)
>>> index = Tensor([[1, 0], [1, 0]], dtype=int64)
>>> src = Tensor([[4, 3], [2, 1]], dtype=float32)
>>> this_tensor.scatter_(1, index, src)
>>> print(this_tensor)
[[3., 4.],
 [1., 2.]]
```

`mindspore.Tensor.scatter_(dim, index, src, *, reduce)`

根据 *index* 使用 *src* 中的值更新当前张量 *self* 。

使用 *reduce* 指定的规约操作, 对 *self* 被 *dim* 选中的维度使用 *index* 进行索引, 对其他维度按顺序遍历, 将 *src* 中的值累加或累乘到 *self* 中, 并返回 *self* 自身。此操作是 `mindspore.Tensor.gather()` 的原地更新版本的逆操作。除替换操作将根据参数 *reduce* 的值更改为累加或累乘外, 其他操作与不带有 *reduce* 参数的接受 *src* 的重载保持一致。

下面看一个三维的例子:

```
self[index[i][j][k]][j][k] += src[i][j][k] # if dim == 0, reduce == "add"

self[i][j][index[i][j][k]] *= src[i][j][k] # if dim == 2, reduce == "multiply"
```

警告:

- 如果 *index* 有多个索引指向 *self* 内的同一个位置, 则 *self* 中该位置的最终值是不确定的。
- 在 Ascend 平台上, 如果 *index* 中元素的值不在 $[-self.shape[dim], self.shape[dim])$ 的范围内, 则其行为是不确定的。
- 这是一个实验性 API, 后续可能修改或删除。

说明: 此重载不支持反向梯度计算, 如计算梯度将返回全 0 结果。

参数:

- **dim** (int) - 要进行操作的轴。取值范围是 $[-r, r)$, 其中 *r* 是 *self* 的秩。
- **index** (Tensor) - 在 *dim* 指定的目标轴上访问 *self* 时使用的索引, 数据类型为 int32 或 int64。如果为空 Tensor, 则将直接返回, 不进行任何操作; 否则其 rank 必须和 *self* 一致, 且每个元素取值范围是 $[-s, s)$, 这里的 *s* 是 *self* 在 *dim* 指定轴的大小。
- **src** (Tensor) - 指定对 *self* 进行更新操作的数据。其 rank 与 dtype 必须与 *self* 的相同。

关键字参数：

- **reduce** (str) - 进行的规约操作，支持 "add", "multiply"。当 *reduce* 设置为 "add" 时，*src* 将根据 *index* 累加到 *self*。当 *reduce* 设置为 "multiply" 时，*src* 将根据 *index* 累乘到 *self*。

返回：

Tensor，返回被修改后的 *self* 自身。

异常：

- **TypeError** - *self*、*index* 或 *src* 的类型不支持。
- **ValueError** - *reduce* 为字符串但取值不为 "add", "multiply"。
- **RuntimeError** - *dim* 的取值超出 $[-r, r)$ 的限制。
- **RuntimeError** - *self* 的秩超过 8。
- **RuntimeError** - 向量 *self*，*index* 或 *src* 的 dtype 不被支持。
- **RuntimeError** - *self* 与 *src* 类型不一致。
- **RuntimeError** - *self*、*index* 与 *src* 的秩不一致且 *index* 不为空。
- **RuntimeError** - 存在一个维度 *d* 使得 $index.size(d) > src.size(d)$ 。
- **RuntimeError** - 存在一个维度 *d* 使得 $index.size(d) > self.size(d)$ 。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor, int64, float32
>>> this_tensor = Tensor([[1, 2], [3, 4]], dtype=float32)
>>> index = Tensor([[1, 0], [1, 0]], dtype=int64)
>>> src = Tensor([[4, 3], [2, 1]], dtype=float32)
>>> this_tensor.scatter_(1, index, src, reduce='add')
>>> print(this_tensor)
[[4., 6.],
 [4., 6.]]
```

`mindspore.Tensor.scatter_(dim, index, value)`

根据 *index* 使用 *value* 更新当前张量 *self*。

对 *self* 被 *dim* 选中的维度使用 *index* 进行索引，对其他维度按顺序遍历，将 *value* 中的值更新到 *self* 中，并返回 *self* 自身。此操作是 `mindspore.Tensor.gather()` 的原地更新版本的逆操作。可以认为将 *value* 广播为 shape 及 dtype 与 *self* 一致的 Tensor 后，其他操作与不带有 *reduce* 参数的接受 *src* 的重载保持一致。

下面看一个三维的例子：

```
self[index[i][j][k]][j][k] = value # if dim == 0

self[i][j][index[i][j][k]] = value # if dim == 2
```

警告：

- 如果 *index* 有多个索引指向 *self* 内的同一个位置，则 *self* 中该位置的最终值是不确定的。

- 在 Ascend 平台上, 如果 *index* 中元素的值不在 $[-self.shape[dim], self.shape[dim])$ 的范围内, 则其行为是不确定的。
- 这是一个实验性 API, 后续可能修改或删除。

参数:

- **dim** (int) - 要进行操作轴。取值范围是 $[-r, r)$, 其中 *r* 是 *self* 的秩。
- **index** (Tensor) - 在 *dim* 指定的目标轴上访问 *self* 时使用的索引, 数据类型为 int32 或 int64。如果为空 Tensor, 则将直接返回, 不进行任何操作; 否则其 rank 必须和 *self* 一致, 且每个元素取值范围是 $[-s, s)$, 这里的 *s* 是 *self* 在 *dim* 指定轴的大小。
- **value** (int, float, bool) - 指定对 *self* 进行更新操作的数据。可视为将尝试将其广播为 shape 及 dtype 与 *self* 一致的 Tensor 并视为 *src* 参与运算。

返回:

Tensor, 返回被修改后的 *self* 自身。

异常:

- **TypeError** - *self*、*index* 或 *value* 的类型不支持。
- **RuntimeError** - *dim* 的取值超出 $[-r, r)$ 的限制。
- **RuntimeError** - *self* 的秩超过 8。
- **RuntimeError** - 张量 *self* 或 *index* 的 dtype 不被支持。
- **RuntimeError** - *index* 不为空且秩与 *self* 不一致。
- **RuntimeError** - 存在一个维度 *d* 使得 $index.size(d) > self.size(d)$ 。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, int64, float32
>>> this_tensor = Tensor([[1, 2], [3, 4]], dtype=float32)
>>> index = Tensor([[0], [1]], dtype=int64)
>>> this_tensor.scatter_(0, index, 10)
>>> print(this_tensor)
[[10., 2.],
 [10., 4.]]
```

`mindspore.Tensor.scatter_(dim, index, value, *, reduce)`

根据 *index* 使用 *value* 更新当前张量 *self*。

使用 *reduce* 指定的规约操作, 对 *self* 被 *dim* 选中的维度使用 *index* 进行索引, 对其他维度按顺序遍历, 将 *value* 中的值累加或累乘到 *self* 中, 并返回 *self* 自身。此操作是 `mindspore.Tensor.gather()` 的原地更新版本的逆操作。除替换操作将根据参数 *reduce* 的值更改为累加或累乘外, 其他行为与不带有 *reduce* 参数的接受 *value* 的重载保持一致。

下面看一个三维的例子:

```
self[i][index[i][j][k]][k] += value # if dim == 1, reduce == "add"

self[i][j][index[i][j][k]] *= value # if dim == 2, reduce == "multiply"
```

警告:

- 如果 *index* 有多个索引指向 *self* 内的同一个位置, 则 *self* 中该位置的最终值是不确定的。
- 在 Ascend 平台上, 如果 *index* 中元素的值不在 $[-self.shape[dim], self.shape[dim])$ 的范围内, 则其行为是不确定的。
- 这是一个实验性 API, 后续可能修改或删除。

说明: 此重载不支持反向梯度计算, 如计算梯度将返回全 0 结果。

参数:

- **dim** (int) - 要进行操作轴。取值范围是 $[-r, r)$, 其中 *r* 是 *self* 的秩。
- **index** (Tensor) - 在 *dim* 指定的目标轴上访问 *self* 时使用的索引, 数据类型为 int32 或 int64。如果为空 Tensor, 则将直接返回, 不进行任何操作; 否则其 rank 必须和 *self* 一致, 且每个元素取值范围是 $[-s, s)$, 这里的 *s* 是 *self* 在 *dim* 指定轴的大小。
- **value** (int, float, bool) - 指定对 *self* 进行更新操作的数据。可视为将尝试将其广播为 shape 及 dtype 与 *self* 一致的 Tensor 并视为 *src* 参与运算。

关键字参数:

- **reduce** (str) - 进行的规约操作, 支持 "add", "multiply"。当 *reduce* 设置为 "add" 时, *src* 将根据 *index* 累加到 *self*。当 *reduce* 设置为 "multiply" 时, *src* 将根据 *index* 累乘到 *self*。

返回:

Tensor, 返回被修改后的 *self* 自身。

异常:

- **TypeError** - *self*、*index* 或 *value* 的类型不支持。
- **ValueError** - *reduce* 为字符串但取值不为 "add", "multiply"。
- **RuntimeError** - *dim* 的取值超出 $[-r, r)$ 的限制。
- **RuntimeError** - *self* 的秩超过 8。
- **RuntimeError** - 张量 *self* 或 *index* 的 dtype 不被支持。
- **RuntimeError** - *index* 不为空且秩与 *self* 不一致。
- **RuntimeError** - 存在一个维度 *d* 使得 $index.size(d) > self.size(d)$ 。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, int64, float32
>>> this_tensor = Tensor([[1, 2], [3, 4]], dtype=float32)
>>> index = Tensor([[0], [1]], dtype=int64)
>>> this_tensor.scatter_(0, index, 3, reduce="multiply")
>>> print(this_tensor)
[[3., 2.],
 [9., 4.]]
```

mindspore.Tensor.scatter_add

mindspore.Tensor.**scatter_add**(dim, index, src)

将 *src* 中所有的元素添加到 *self* 中 *index* 指定的索引处。其中 *dim* 控制 *scatter_add* 操作的轴。*self*、*index*、*src* 三者的 *rank* 都必须大于或等于 1。

下面看一个三维的例子：

```

self[index[i][j][k]][j][k] += src[i][j][k] # if dim == 0
self[i][index[i][j][k]][k] += src[i][j][k] # if dim == 1
self[i][j][index[i][j][k]] += src[i][j][k] # if dim == 2

```

说明：输入 Tensor *self* 的 *rank* 必须大于等于 1。

参数：

- **dim** (int) - *self* 执行 *scatter_add* 操作的轴。取值范围是 $[-r, r)$ ，其中 *r* 是 *self* 的 *rank*。
- **index** (Tensor) - *self* 要进行 *scatter_add* 操作的目标索引。数据类型为 *int32* 或 *int64*，*rank* 必须和 *self* 一致。除了 *dim* 指定的维度，*index* 的每一维的 *size* 都需要小于等于 *self* 对应维度的 *size*。
- **src** (Tensor) - 指定与 *self* 进行 *scatter_add* 操作的 Tensor，其数据类型与 *self* 类型相同，*shape* 中每一维的 *size* 必须大于等于 *index*。

返回：

Tensor，*shape* 和数据类型与输入 *self* 相同。

异常：

- **TypeError** - *index* 的数据类型不满足 *int32* 或 *int64*。
- **ValueError** - *self*、*index* 和 *src* 中，任意一者的 *rank* 小于 1。
- **ValueError** - *self*、*index* 和 *src* 的 *rank* 不一致。
- **ValueError** - 除了 *dim* 指定的维度，*index* 的任意维的 *size* 大于 *self* 对应维度的 *size*。
- **ValueError** - *src* 任意维度 *size* 小于 *index* 对应维度的 *size*。

支持平台：

Ascend

样例：

```

>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> input = Tensor(np.array([[1, 2, 3, 4, 5]]), dtype=ms.float32)
>>> src = Tensor(np.array([[8, 8]]), dtype=ms.float32)
>>> index = Tensor(np.array([[2, 4]]), dtype=ms.int64)
>>> out = input.scatter_add(dim=1, index=index, src=src)
>>> print(out)
[[1.  2. 11.  4. 13.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)

```

(下页继续)

(续上页)

```

>>> index = Tensor(np.array([[0, 0, 0], [2, 2, 2], [4, 4, 4]]), dtype=ms.int64)
>>> out = input.scatter_add(dim=0, index=index, src=src)
>>> print(out)
[[1. 2. 3. 0. 0.]
 [0. 0. 0. 0. 0.]
 [4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0.]
 [7. 8. 9. 0. 0.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 2, 4], [0, 2, 4], [0, 2, 4]]), dtype=ms.int64)
>>> out = input.scatter_add(dim=1, index=index, src=src)
>>> print(out)
[[1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6.]
 [7. 0. 8. 0. 9.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]

```

`mindspore.Tensor.scatter_add(indices, updates)`

根据指定的更新值 *updates* 和输入索引 *indices*，通过相加运算更新输入 Tensor 的值。当同一索引有不同值时，更新的结果将是所有值的总和。此操作与 `mindspore.ops.scatter_nd_add()` 类似，但更新后的结果是返回一个新的输出 Tensor，而不是直接更新 *self*。

indices 的最后一个轴是每个索引向量的深度。对于每个索引向量，*updates* 中必须有相应的值。*updates* 的 shape 应该等于 *self[indices]* 的 shape。有关更多详细信息，请参见样例。

$$output[indices] = input\_x + update$$

说明：输入 Tensor *self* 的维度必须不小于 *indices.shape[-1]*。

如果 *indices* 中的值超出输入 *self* 索引范围：

- GPU 平台上相应的 *updates* 不会更新到 *self* 且不会抛出索引错误。
- CPU 平台上直接抛出索引错误。
- Ascend 平台不支持越界检查，若越界可能会造成未知错误。

参数：

- **indices** (Tensor) - 输入 Tensor 的索引，数据类型为 int32 或 int64。其 rank 至少为 2。
- **updates** (Tensor) - 指定与 *self* 相加操作的 Tensor，其数据类型与 *self* 相同。并且其 shape 应等于 *indices.shape[: -1] + input_x.shape[indices.shape[-1] :]*。

返回：

Tensor, shape 和数据类型与输入 *self* 相同。

异常：

- **TypeError** - *indices* 的数据类型不为 int32 或 int64。
- **ValueError** - *self* 的 rank 小于 *indices.shape* 的最后一维。
- **RuntimeError** - 在 CPU 平台中，*indices* 中的值超出了 *self* 的索引范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = Tensor(np.array([[[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]], mindspore.float32)
>>> indices = Tensor(np.array([[0, 0], [0, 0]]), mindspore.int32)
>>> updates = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> output = input_x.scatter_add(indices, updates)
>>> print(output)
[[ 3.1  0.3  3.6]
 [ 0.4  0.5 -3.2]]
```

`mindspore.Tensor.scatter_add_`

`mindspore.Tensor.scatter_add_(dim, index, src)`

将 `src` 中所有的元素添加到 `self` 中 `index` 指定的索引处 (属于原地更新操作)。其中 `dim` 控制 `scatter add` 操作的轴。`self`、`index`、`src` 三者的 `rank` 都必须大于或等于 1。

下面看一个三维的例子:

```
self[index[i][j][k]][j][k] += src[i][j][k] # if dim == 0
self[i][index[i][j][k]][k] += src[i][j][k] # if dim == 1
self[i][j][index[i][j][k]] += src[i][j][k] # if dim == 2
```

参数:

- **dim** (int) - `self` 执行 `scatter_add` 操作的轴。取值范围是 `[-r, r)`, 其中 `r` 是 `self` 的 `rank`。
- **index** (Tensor) - `self` 要进行 `scatter_add` 操作的目标索引。数据类型为 `int32` 或 `int64`, `rank` 必须和 `self` 一致。除了 `dim` 指定的维度, `index` 的每一维的 `size` 都需要小于或等于 `self` 对应维度的 `size`。
- **src** (Tensor) - 指定与 `self` 进行 `scatter_add` 操作的 Tensor, 其数据类型与 `self` 类型相同, `shape` 中每一维的 `size` 必须大于或等于 `index`。

返回:

Tensor, `shape` 和数据类型与输入 `self` 相同。

异常:

- **TypeError** - `index` 的数据类型不是 `int32` 或 `int64`。
- **ValueError** - `self`、`index` 和 `src` 中, 任意一者的 `rank` 小于 1。
- **ValueError** - `self`、`index` 和 `src` 的 `rank` 不一致。
- **ValueError** - 除了 `dim` 指定的维度, `index` 的任意维的 `size` 大于 `self` 对应维度的 `size`。
- **ValueError** - `src` 任意维度 `size` 小于 `index` 对应维度的 `size`。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> input = Tensor(np.array([[1, 2, 3, 4, 5]]), dtype=ms.float32)
>>> src = Tensor(np.array([[8, 8]]), dtype=ms.float32)
>>> index = Tensor(np.array([[2, 4]]), dtype=ms.int64)
>>> out = input.scatter_add_(1, index, src)
>>> print(out)
[[1.  2. 11.  4. 13.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 0, 0], [2, 2, 2], [4, 4, 4]]), dtype=ms.int64)
>>> out = input.scatter_add_(0, index, src)
>>> print(out)
[[1.  2.  3.  0.  0.]
 [0.  0.  0.  0.  0.]
 [4.  5.  6.  0.  0.]
 [0.  0.  0.  0.  0.]
 [7.  8.  9.  0.  0.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 2, 4], [0, 2, 4], [0, 2, 4]]), dtype=ms.int64)
>>> out = input.scatter_add_(1, index, src)
>>> print(out)
[[1.  0.  2.  0.  3.]
 [4.  0.  5.  0.  6.]
 [7.  0.  8.  0.  9.]
 [0.  0.  0.  0.  0.]
 [0.  0.  0.  0.  0.]]
```

mindspore.Tensor.scatter_div

`mindspore.Tensor.scatter_div(indices, updates)`

详情请参考[mindspore.ops.scatter_div\(\)](#)。

mindspore.Tensor.scatter_max

`mindspore.Tensor.scatter_max(indices, updates)`

详情请参考[mindspore.ops.scatter_max\(\)](#)。

mindspore.Tensor.scatter_min

`mindspore.Tensor.scatter_min(indices, updates)`

详情请参考[mindspore.ops.scatter_min\(\)](#)。

mindspore.Tensor.scatter_mul

`mindspore.Tensor.scatter_mul(indices, updates)`
 详情请参考[mindspore.ops.scatter_mul\(\)](#)。

mindspore.Tensor.scatter_sub

`mindspore.Tensor.scatter_sub(indices, updates)`
 详情请参考[mindspore.ops.tensor_scatter_sub\(\)](#)。

mindspore.Tensor.searchsorted

`mindspore.Tensor.searchsorted(v, side='left', sorter=None)`
 详情请参考[mindspore.ops.searchsorted\(\)](#)。

mindspore.Tensor.select

`mindspore.Tensor.select(dim, index)`
 沿着选定的维度在给定的索引处进行切片。

警告：这是一个实验性 API，可能会更改或删除。

参数：

- **dim** (int) - 指定切片的维度。
- **index** (int) - 指定索引值。

返回：

Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> input = Tensor([[2, 3, 4, 5], [3, 2, 4, 5]])
>>> y = Tensor.select(input, 0, 0)
>>> print(y)
[2 3 4 5]
```

`mindspore.Tensor.select(condition, y)`

根据条件判断 Tensor 中的元素的值，来决定输出中的相应元素是从 *self*（如果元素值为 True）还是从 *y*（如果元素值为 False）中选择。

该算法可以被定义为：

$$out_i = \begin{cases} self_i, & \text{if } condition_i \\ y_i, & \text{otherwise} \end{cases}$$

参数:

- **condition** (Tensor[bool]) - 条件 Tensor, 决定选择哪一个元素。维度是: $(x_1, x_2, \dots, x_N, \dots, x_R)$ 。
- **y** (Union[Tensor, int, float]) - 备选的 Tensor 或者数字。
 - 如果 y 是一个 Tensor, 那么 shape 是或者可以被广播为: $(x_1, x_2, \dots, x_N, \dots, x_R)$ 。
 - 如果 y 是 int 或者 float, 那么将会被转化为 int32 或者 float32 类型, 并且被广播为与 *self* 相同的 shape。*self* 和 y 中至少要有一个 Tensor。

返回:

Tensor。与 *condition* 的 shape 相同。

异常:

- **TypeError** - 如果 y 不是 Tensor、int 或者 float。
- **ValueError** - 输入的 shape 不能被广播。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor
>>> # Both input are Tensor
>>> cond = Tensor([True, False])
>>> x = Tensor([2, 3], mindspore.float32)
>>> y = Tensor([1, 2], mindspore.float32)
>>> output = Tensor.select(x, cond, y)
>>> print(output)
[2. 2.]
```

mindspore.Tensor.select_scatter

`mindspore.Tensor.select_scatter(src, axis, index)`

详情请参考 `mindspore.ops.select_scatter()`。

mindspore.Tensor.set_const_arg

`mindspore.Tensor.set_const_arg(const_arg=True)`

指定该 Tensor 在作为网络入参时是否是一个常量。

参数:

- **const_arg** (bool) - Tensor 在作为网络入参时是否是一个常量。默认值: True。

返回:

Tensor, 被指定了是否是一个常量网络入参。

异常:

- **TypeError** - 如果 *const_arg* 不是一个布尔值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1,2,3],[4,5,6]], dtype=np.float32))
>>> x.set_const_arg(True)
```

mindspore.Tensor.sgn

`mindspore.Tensor.sgn()`

详情请参考[mindspore.ops.sgn\(\)](#)。

mindspore.Tensor.shape

property `mindspore.Tensor.shape`

详情请参考[mindspore.ops.shape\(\)](#)。

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> print(x.shape)
(2, 2)
```

mindspore.Tensor.short

`mindspore.Tensor.short()`

将输入 `Tensor` 转换为 `int16` 类型并返回其副本, 与 `self.astype(mstype.int16)` 等价, 当 `Tensor` 中的值为浮点数时, 则会丢弃小数部分, 具体请参考[mindspore.Tensor.astype\(\)](#)。

返回:

`Tensor`, 其数据类型为 `int16`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> x = ms.Tensor(np.array([1,2,3,4,5]), ms.int32)
>>> output = x.short()
>>> output
Tensor(shape=[5], dtype=Int16, value= [1, 2, 3, 4, 5])
```

mindspore.Tensor.sigmoid

`mindspore.Tensor.sigmoid()`

详情请参考[`mindspore.ops.sigmoid\(\)`](#)。

mindspore.Tensor.sign

`mindspore.Tensor.sign()`

详情请参考[`mindspore.ops.sign\(\)`](#)。

mindspore.Tensor.signbit

`mindspore.Tensor.signbit()`

详情请参考[`mindspore.ops.signbit\(\)`](#)。

mindspore.Tensor.sin

`mindspore.Tensor.sin()`

详情请参考[`mindspore.ops.sin\(\)`](#)。

mindspore.Tensor.sinc

`mindspore.Tensor.sinc()`

详情请参考[`mindspore.ops.sinc\(\)`](#)。

mindspore.Tensor.sinh

`mindspore.Tensor.sinh()`

详情请参考[`mindspore.ops.sinh\(\)`](#)。

mindspore.Tensor.size

property `mindspore.Tensor.size`

详情请参考[`mindspore.ops.size\(\)`](#)。

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.size
>>> print(output)
4
```

`mindspore.Tensor.slice_scatter`

`mindspore.Tensor.slice_scatter` (*src*, *axis=0*, *start=None*, *end=None*, *step=1*)

详情请参考[mindspore.ops.slice_scatter\(\)](#)。

`mindspore.Tensor.slogdet`

`mindspore.Tensor.slogdet` ()

详情请参考[mindspore.ops.slogdet\(\)](#)。

`mindspore.Tensor.softmax`

`mindspore.Tensor.softmax` (*axis*, *dtype=None*)

详情请参考[mindspore.ops.softmax\(\)](#)。

`mindspore.Tensor.sort`

`mindspore.Tensor.sort` (*dim=-1*, *descending=False*)

按指定顺序对输入 Tensor 的指定维上的元素进行排序。

警告：目前能良好支持的数据类型有：float16、uint8、int8、int16、int32、int64。如果使用 float32，可能产生精度误差。

参数：

- **dim** (int, 可选) - 指定排序的维。默认值：-1，表示指定最后一维。
- **descending** (bool, 可选) - 按降序还是升序。如果为 True，则元素按降序排列，否则按升序排列。默认值：False。

返回：

- **y1** (Tensor) - 排序后的值，其 shape 和数据类型与输入一致。
- **y2** (Tensor) - 返回值在原输入 Tensor 里对应的索引，数据类型为 int64。

异常：

- **TypeError** - *dim* 不是 int 类型。
- **TypeError** - *descending* 不是 bool 类型。
- **TypeError** - *self* 不是 float16、float32、uint8、int8、int16、int32、int64 或 bfloat16。
- **TypeError** - *stable* 不是 bool 类型。
- **ValueError** - *dim* 不在 [-len(self.shape), len(self.shape)) 范围内。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[8, 2, 1], [5, 9, 3], [4, 6, 7]]), mindspore.float16)
>>> output = x.sort(dim=-1)
>>> # The output below is based on the Ascend platform.
>>> print(output)
(Tensor(shape=[3, 3], dtype=Float16, value=
[[ 1.0000e+00,  2.0000e+00,  8.0000e+00],
[ 3.0000e+00,  5.0000e+00,  9.0000e+00],
[ 4.0000e+00,  6.0000e+00,  7.0000e+00]]), Tensor(shape=[3, 3], dtype=Int64, value=
[[2, 1, 0],
[2, 0, 1],
[0, 1, 2]]))
```

`mindspore.Tensor.sort` (*axis=-1, descending=False*)

按指定顺序对输入 `Tensor` 的指定维上的元素进行排序。

参数：

- **axis** (int, 可选) - 指定排序的轴。默认值：-1，表示指定最后一维。当前 Ascend 后端只支持对最后一维进行排序。
- **descending** (bool, 可选) - 按降序还是升序。如果为 `True`，则元素按降序排列，否则按升序排列。默认值：`False`。

警告：目前能良好支持的数据类型有：`float16`、`uint8`、`int8`、`int16`、`int32`、`int64`。如果使用 `float32`，可能产生精度误差。

返回：

- **y1** (`Tensor`) - 排序后的值，其 `shape` 和数据类型与输入一致。
- **y2** (`Tensor`) - 返回值在原输入 `Tensor` 里对应的索引，数据类型为 `int32`。

异常：

- **TypeError** - *axis* 不是 `int` 类型。
- **TypeError** - *descending* 不是 `bool` 类型。
- **TypeError** - *self* 不是 `float16`、`float32`、`uint8`、`int8`、`int16`、`int32` 或 `int64`。
- **ValueError** - *axis* 不在 `[-len(self.shape), len(self.shape))` 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[8, 2, 1], [5, 9, 3], [4, 6, 7]]), mindspore.float16)
>>> output = x.sort(axis=-1)
>>> # The output below is based on the Ascend platform.
>>> print(output)
```

(下页继续)

(续上页)

```
(Tensor(shape=[3, 3], dtype=Float16, value=
[[ 1.0000e+00,  2.0000e+00,  8.0000e+00],
 [ 3.0000e+00,  5.0000e+00,  9.0000e+00],
 [ 4.0000e+00,  6.0000e+00,  7.0000e+00]]), Tensor(shape=[3, 3], dtype=Int32, value=
[[2, 1, 0],
 [2, 0, 1],
 [0, 1, 2]]))
```

mindspore.Tensor.split

`mindspore.Tensor.split` (*split_size*, *dim=0*)

根据指定的轴将输入 Tensor 切分成块。

参数：

- **split_size** (Union[int, tuple(int), list(int)]) - 如果 *split_size* 是 int 类型，输入 Tensor 将被均匀的切分成块，每块的大小为 *split_size*，若 *tensor.shape[dim]* 不能被 *split_size* 整除，最后一块大小将小于 *split_size*。如果 *split_size* 是个 list 类型，输入 Tensor 将沿 *dim* 轴被切分成 *len(split_size)* 块，大小为 *split_size*。
- **dim** (int, 可选) - 指定分割轴。默认值：0。

返回：

tuple[Tensor]。

异常：

- **TypeError** - 如果 *dim* 不是 int 类型。
- **ValueError** - 如果 *dim* 超出取值范围 $[-\text{tensor.ndim}, \text{tensor.ndim})$ 。
- **TypeError** - *split_size* 中的每个元素不是 int、tuple(int) 或者 list(int)。
- **TypeError** - *split_size* 的数据类型不是 int、tuple(int) 或者 list(int)。
- **ValueError** - *split_size* 的和不等于 *x.shape[dim]*。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = np.arange(9).astype("float32")
>>> output = Tensor.split(Tensor(input_x), 3)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  1.00000000e+00,  2.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 3.00000000e+00,  4.00000000e+00,  5.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 6.00000000e+00,  7.00000000e+00,  8.
↪00000000e+00]))
```

`mindspore.Tensor.split` (*split_size_or_sections*, *axis=0*)

根据指定的轴将输入 Tensor 切分成块。

参数：

- **split_size_or_sections** (Union[int, tuple(int), list(int)]) - 如果 *split_size_or_sections* 是 int 类型, 输入 Tensor 将被均匀的切分成块, 每块的大小为 *split_size_or_sections*, 若 *tensor.shape[axis]* 不能被 *split_size_or_sections* 整除, 最后一块大小将小于 *split_size_or_sections*。如果 *split_size_or_sections* 是个 list 类型, 输入 Tensor 将沿 *axis* 轴被切分成 *len(split_size_or_sections)* 块, 大小为 *split_size_or_sections*。
- **axis** (int, 可选) - 指定分割轴。默认值: 0。

返回:

tuple[Tensor]。

异常:

- **TypeError** - 如果 *axis* 不是 int 类型。
- **ValueError** - 如果 *axis* 超出取值范围 $[-\text{tensor.ndim}, \text{tensor.ndim})$ 。
- **TypeError** - *split_size_or_sections* 中的每个元素不是 int、tuple(int) 或者 list(int)。
- **TypeError** - *split_size_or_sections* 的数据类型不是 int、tuple(int) 或者 list(int)。
- **ValueError** - *split_size_or_sections* 的和不等于 *x.shape[axis]*。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = np.arange(9).astype("float32")
>>> output = Tensor.split(Tensor(input_x), 3)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 3.00000000e+00,  4.00000000e+00,  5.00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 6.00000000e+00,  7.00000000e+00,  8.00000000e+00]))
```

mindspore.Tensor.sqrt

mindspore.Tensor.**sqrt**()

详情请参考[mindspore.ops.sqrt\(\)](#)。

mindspore.Tensor.square

mindspore.Tensor.**square**()

详情请参考[mindspore.ops.square\(\)](#)。

mindspore.Tensor.squeeze

`mindspore.Tensor.squeeze(axis=None)`

详情请参考 `mindspore.ops.squeeze()`。

mindspore.Tensor.std

`mindspore.Tensor.std(axis=None, ddof=0, keepdims=False)`

详情请参考 `mindspore.ops.std()`。

`mindspore.Tensor.std(dim=None, *, correction=1, keepdim=False)`

计算指定维度 `dim` 上的标准差。`dim` 可以是单个维度或维度列表，也可以是 `None`，表示移除所有维度。

标准差 (σ) 计算如下：

$$\sigma = \sqrt{\frac{1}{N - \delta N} \sum_{j=1}^{N-1} (sel f_{ij} - \bar{x}_i)^2}$$

其中 x 表示用来计算标准差的样本集， $\bar{x}$ 表示样本的均值， N 表示样本的数量， δN 则为 `correction` 的值。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **dim** (None, int, tuple(int), 可选) - 用来进行规约计算的维度。默认值为 `None`，所有维度都进行规约计算。

关键字参数：

- **correction** (int, 可选) - 样本大小和样本自由度之间的差异。默认为 Bessel 校正，默认值为 1。
- **keepdim** (bool, 可选) - 是否保留输出 Tensor 的维度。如果为 `True`，则保留缩小的维度，其大小为 1，否则移除维度。默认值为 `False`。

返回：

Tensor，标准差。

假设输入 `self` 的 shape 为 $(x_0, x_1, \dots, x_R)$ ：

- 如果 `dim` 为 `()`，且 `keepdim` 为 `False`，则返回一个零维 Tensor，表示输入 Tensor `self` 中所有元素的标准差。
- 如果 `dim` 为 `int`，如取值为 1，且 `keepdim` 为 `False`，则返回 Tensor 的 shape 为 $(x_0, x_2, \dots, x_R)$ 。
- 如果 `dim` 为 `tuple(int)` 或者 `list(int)`，如取值为 $(1, 2)$ ，且 `keepdim` 为 `False`，则返回 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。

异常：

- **TypeError** - 如果 `self` 不是 Tensor。
- **TypeError** - 如果 `self` 的数据类型不是 `bfloat16`、`float16` 或 `float32`。
- **TypeError** - 如果 `dim` 不是 `None`、`int` 或 `tuple` 类型。
- **TypeError** - 如果 `correction` 不是 `int` 类型。
- **TypeError** - 如果 `keepdim` 不是 `bool` 类型。
- **ValueError** - 如果 `dim` 不在 $[-self.ndim, self.ndim)$ 范围内。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import mint, Tensor
>>> input = Tensor(np.array([[1, 2, 3], [-1, 1, 4]]).astype(np.float32))
>>> output = input.std(dim=1, correction=1, keepdim=False)
>>> print(output)
[1.          2.5166113]
```

mindspore.Tensor.storage_offset

mindspore.Tensor.storage_offset()

以储存元素的个数的形式返回 Tensor 在内存中的偏移量。

返回:

int, 返回 Tensor 在内存中的偏移量。

样例:

```
>>> import mindspore as ms
>>> x = ms.Tensor([1, 2, 3, 4, 5], dtype=ms.float32)
>>> ret = x.storage_offset()
>>> print(ret)
0
```

mindspore.Tensor.stride

mindspore.Tensor.stride(dim=None)

在指定维度 *dim* 中从一个元素跳到下一个元素所必需的步长。当没有参数传入时, 返回所有维度的步长的列表。

参数:

- **dim** (int, 可选) - 指定的维度。默认值: None。

返回:

int, 返回在指定维度下, 从一个元素跳到下一个元素所必需的步长。

异常:

- **TypeError** - *dim* 不是 int。

样例:

```
>>> import mindspore as ms
>>> x = ms.Tensor([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]], dtype=ms.float32)
>>> x.stride()
[5, 1]
```

mindspore.Tensor.strides

property mindspore.Tensor.strides

Tensor 上每个维度跨度的字节元组。

样例：

```
>>> from mindspore import Tensor
>>> from mindspore import dtype as mstype
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]), dtype=mstype.int64)
>>> output = x.strides
>>> print(output)
(16, 8)
```

mindspore.Tensor.sub

mindspore.Tensor.sub(other, *, alpha=1)

对 other 缩放 alpha 后与 input 相减。

$$out_i = self_i - alpha \times other_i$$

说明：

- 当两个输入具有不同的 shape 时，它们的 shape 必须要能广播为一个共同的 shape。
- 两个输入和 alpha 遵循隐式类型转换规则，使数据类型保持一致。

参数：

- other** (Union[Tensor, number.Number, bool]) - 另外一个输入，数据类型为 *number.Number*、*bool* 或者数据类型为 *number* 或 *bool_* 的 Tensor。

关键字参数：

- alpha** (*number.Number*, 可选) - 应用于 other 的缩放因子，默认值为 1。

返回：

Tensor。其 shape 与 *self*、*other* 广播后的 shape 相同，并且数据类型是两个输入和 *alpha* 之间具有更高精度，或位数更多的类型。

异常：

- TypeError** - 如果 other 或者 alpha 的类型不是以下类型：*number.Number*、*bool* 或 *Tensor*。
- TypeError** - 如果 alpha 是浮点类型，但是 *self* 和 *other* 却不是浮点类型。
- TypeError** - 如果 alpha 是 bool 类型，但是 *self* 和 *other* 却不是 bool 类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> y = Tensor(1, mindspore.int32)
>>> output = Tensor.sub(x, y, alpha=0.5)
>>> print(output)
[3.5 4.5 5.5]
>>> # the data type of x is float32, the data type of y is int32,
>>> # alpha is a float, and the output is the data format of higher precision float32.
>>> print(output.dtype)
Float32
```

`mindspore.Tensor.sub(y)`

详情请参考 `mindspore.ops.sub()`。

`mindspore.Tensor.sub_`

`mindspore.Tensor.sub_(other, *, alpha=1)`

详情请参考 `mindspore.mint.sub()`。

`mindspore.Tensor.subtract`

`mindspore.Tensor.subtract(other, *, alpha=1)`

此接口从 2.4 版本开始已被弃用，并将在未来版本中被移除。

`mindspore.Tensor.sum`

`mindspore.Tensor.sum(dim=None, keepdim=False, *, dtype=None)`

计算指定维度上 Tensor 元素的总和。

说明:

- 带有张量类型的 `dim` 仅用于与旧版本兼容，不推荐使用。

参数:

- **dim** (Union[None, int, tuple(int), list(int), Tensor], 可选) - 指定维度，在该维度方向上进行求和运算。默认值: None。如果参数值为 None，会计算输入 Tensor 中所有元素的和。如果 `dim` 为负数，则从最后一维开始往第一维计算。如果 `dim` 为整数元组或列表，会对该元组或列表指定的所有 `dim` 方向上的元素进行求和。必须在 `[-self.ndim, self.ndim)` 范围内。
- **keepdim** (bool, 可选) - 输出 Tensor 是否保留了 `dim`。默认值为 False。如果这个参数为 True，保留这些缩小的尺寸并且长度为 1。如果这个参数为 False，不保留这些尺寸。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 返回 Tensor 所期望的数据类型。默认值: None。

返回:

Tensor。返回输入 Tensor 中指定 `dim` 维度的元素总和。

异常：

- **TypeError** - 如果 *dim* 不是 int, tuple(int), list(int), Tensor 或者 None。
- **ValueError** - 如果 *dim* 没有在 $[-self.ndim, self.ndim)$ 范围中。
- **TypeError** - 如果 *keepdim* 不是 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
...            mstype.float32)
>>> out = Tensor.sum(x)
>>> print(out)
270.0
>>> out = Tensor.sum(x, dim=2)
>>> print(out)
[[ 6. 12. 18.]
 [24. 30. 36.]
 [42. 48. 54.]]
>>> out = Tensor.sum(x, dim=2, keepdim=True)
>>> print(out)
[[[ 6.]
 [12.]
 [18.]]
 [[24.]
 [30.]
 [36.]]
 [[42.]
 [48.]
 [54.]]]
```

`mindspore.Tensor.sum(axis=None, dtype=None, keepdims=False, initial=None)`

返回指定维度上数组元素的总和。

说明：

- 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature* 和 *extobj*。
- Tensor 类型的 *axis* 仅用作兼容旧版本，不推荐使用。

参数：

- **axis** (Union[None, int, tuple(int), list(int), Tensor], 可选) - 指定维度，在该维度方向上进行求和运算。默认值：None。如果参数值为 None，会计算输入数组中所有元素的和。如果 *axis* 为负数，则从最后一维开始往第一维计算。如果 *axis* 为整数元组或列表，会对该元组或列表指定的所有 *axis* 方向上的元素进行求和。
- **dtype** (*mindspore.dtype*, 可选) - 默认值为 None。会覆盖输出 Tensor 的 dtype。

- **keepdims** (bool, 可选) - 如果这个参数为 `True`，被删去的维度保留在结果中，且维度大小设为 1。有了这个选项，结果就可以与输入数组进行正确的广播运算。如果设为默认值，那么 *keepdims* 不会被传递给 `ndarray` 子类的 `sum` 方法。但是任何非默认值都会被传递。如果子类的方法未实现 *keepdims*，则引发异常。默认值：`False`。
- **initial** (scalar, 可选) - 初始化的起始值。默认值：`None`。

返回：

`Tensor`。具有与输入相同 `shape` 的 `Tensor`，删除了指定的 *axis*。如果输入 `Tensor` 是零维数组，或 *axis* 为 `None` 时，返回一个标量。

异常：

- **TypeError** - input 不是 `Tensor`，*axis* 不是整数、整数元组、整数列表或 `Tensor`，*keepdims* 不是整数，或者 *initial* 不是标量。
- **ValueError** - 任意 *axis* 超出范围或存在重复的 *axis*。

参见：

- `mindspore.Tensor.cumsum()`：返回沿给定 *axis* 的元素累加和。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = Tensor(np.array([-1, 0, 1]).astype(np.float32))
>>> print(input_x.sum())
0.0
>>> input_x = Tensor(np.arange(10).reshape(2, 5).astype(np.float32))
>>> print(input_x.sum(axis=1))
[10. 35.]
```

mindspore.Tensor.sum_to_size

`mindspore.Tensor.sum_to_size(*size)`

将原 `Tensor` 按照指定 *size* 进行求和。*size* 必须可以扩展到 `Tensor` 的 `shape`。

参数：

- **size** (Union[tuple(int), int]) - 期望输出 `Tensor` 的 `shape`。

返回：

`Tensor`，根据 *size* 对原 `Tensor` 进行求和的结果。

异常：

- **ValueError** - *size* 不能扩展成原 `Tensor` 的大小。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.random.randn(3, 3, 3, 3, 3, 3), mindspore.float32)
>>> output = x.sum_to_size((1, 3, 1, 3))
>>> print(output.shape)
(1, 3, 1, 3)
```

mindspore.Tensor.svd

`mindspore.Tensor.svd` (*full_matrices=False, compute_uv=True*)

详情请参考 `mindspore.ops.svd()`。

mindspore.Tensor.swapaxes

`mindspore.Tensor.swapaxes` (*axis0, axis1*)

详情请参考 `mindspore.ops.swapaxes()`。

mindspore.Tensor.swapdims

`mindspore.Tensor.swapdims` (*dim0, dim1*)

详情请参考 `mindspore.ops.swapdims()`。

mindspore.Tensor.T

property `mindspore.Tensor.T`

返回转置后的 Tensor。

样例：

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.array([[1, 2], [3, 4]]))
>>> output = x.T
>>> print(output)
[[1 3]
 [2 4]]
```

mindspore.Tensor.take

`mindspore.Tensor.take` (*indices, axis=None, mode='clip'*)

在指定维度上获取 Tensor 中的元素。

参数：

- **indices** (Tensor) - 待提取的值的 shape 为 (*Nj...*) 的索引。
- **axis** (int, 可选) - 在指定维度上选择值。默认情况下，使用展开的输入数组。默认值：None。
- **mode** (str, 可选) - 支持 'raise'、'wrap'、'clip'。

- raise: 抛出错误。
- wrap: 绕接。
- clip: 裁剪到范围。clip 模式意味着所有过大的索引都会被在指定轴方向上指向最后一个元素的索引替换。注：这将禁用具有负数的索引。

默认值: 'clip'。

返回:

Tensor, 索引的结果。

异常:

- **ValueError** - *axis* 超出范围, 或 *mode* 被设置为 'raise'、'wrap' 和 'clip' 以外的值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> a = Tensor(np.array([4, 3, 5, 7, 6, 8]))
>>> indices = Tensor(np.array([0, 1, 4]))
>>> output = a.take(indices)
>>> print(output)
[4 3 6]
```

`mindspore.Tensor.take(index)`

选取给定索引 *index* 处的 *self* 元素。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **index** (LongTensor) - 输入张量的索引张量。

返回:

Tensor, shape 与索引的 shape 相同。

异常:

- **TypeError** - 如果 *index* 的数据类型不是 long。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> input = Tensor([[4, 3, 5], [6, 7, 8]], ms.float32)
>>> index = Tensor([0, 2, 5], ms.int64)
>>> output = input.take(index)
>>> print(output)
[4, 5, 8]
```


mindspore.Tensor.tan

`mindspore.Tensor.tan()`

详情请参考[`mindspore.ops.tan\(\)`](#)。

mindspore.Tensor.tanh

`mindspore.Tensor.tanh()`

详情请参考[`mindspore.ops.tanh\(\)`](#)。

mindspore.Tensor.tensor_split

`mindspore.Tensor.tensor_split(indices_or_sections, axis=0)`

详情请参考[`mindspore.ops.tensor\_split\(\)`](#)。

mindspore.Tensor.tile

`mindspore.Tensor.tile(dims)`

按照给定的次数复制输入 Tensor。

说明: 在 Ascend 平台上, *dims* 参数的个数不大于 8, 当前不支持超过 4 个维度同时做 repeat 的场景。

参数:

- **dims** (tuple[int]) - 指定复制次数的参数, 参数类型为 tuple, 数据类型为整数。如 $(y_1, y_2, \dots, y_S)$ 。只支持常量值。

返回:

Tensor, 具有与 *self* 相同的数据类型。假设 *dims* 的长度为 *d*, *self* 的维度为 *self.dim*, *self* 的 shape 为 $(x_1, x_2, \dots, x_S)$ 。

- 如果 $self.dim = d$, 将其相应位置的 shape 相乘, 输出的 shape 为 $(x_1 * y_1, x_2 * y_2, \dots, x_S * y_S)$ 。
- 如果 $self.dim < d$, 在 *self* 的 shape 的前面填充 1, 直到它们的长度一致。例如将 *self* 的 shape 设置为 $(1, \dots, x_1, x_2, \dots, x_S)$, 然后可以将其相应位置的 shape 相乘, 输出的 shape 为 $(1 * y_1, \dots, x_R * y_R, x_S * y_S)$ 。
- 如果 $self.dim > d$, 在 *dims* 的前面填充 1, 直到它们的长度一致。例如将 *dims* 设置为 $(1, \dots, y_1, y_2, \dots, y_S)$, 然后可以将其相应位置的 shape 相乘, 输出的 shape 为 $(x_1 * 1, \dots, x_R * y_R, x_S * y_S)$ 。

异常:

- **TypeError** - *dims* 不是 tuple 或者其元素并非全部是 int。
- **ValueError** - *dims* 的元素并非全部大于或等于 0。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([[1, 2], [3, 4]]), mindspore.float32)
>>> dims = (2, 3)
>>> output = input.tile(dims)
>>> print(output)
[[1.  2.  1.  2.  1.  2.]
 [3.  4.  3.  4.  3.  4.]
 [1.  2.  1.  2.  1.  2.]
 [3.  4.  3.  4.  3.  4.]]
>>> dims = (2, 3, 2)
>>> output = input.tile(dims)
>>> print(output)
[[[1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]]
 [[1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]]]]

```

`mindspore.Tensor.tile(reps)`

详情请参考[mindspore.ops.tile\(\)](#)，当前接口的参数 *reps* 和详情参考接口中的参数 *dims* 实际为同一参数。

mindspore.Tensor.to

`mindspore.Tensor.to(dtype)`

执行 Tensor 类型的转换。

说明:

- 如果 *self* 张量已经具有所需的 *mindspore.dtype*，则直接返回 *self*。否则，将返回一个具有目标 *mindspore.dtype* 的 *self* 的副本。
- 当将复数转换为布尔类型时，复数的虚部不被考虑。只要实部非零，它就返回 `True`；否则，返回 `False`。

参数:

- **dtype** (`dtype.Number`) - 输出 Tensor 的有效数据类型，只允许常量值。

返回:

Tensor，其数据类型为 *dtype*。

异常:

- **TypeError** - 如果 *dtype* 不是数值类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_np = np.random.randn(2, 3, 4, 5).astype(np.float32)
>>> input_x = Tensor(input_np)
>>> dtype = mindspore.int32
>>> output = input_x.to(dtype)
>>> print(output.dtype)
Int32
```

mindspore.Tensor.to_coo

`mindspore.Tensor.to_coo()`

将常规 Tensor 转为稀疏化的 COOTensor。

说明: 现在只支持二维 Tensor。

返回:

返回一个二维的 COOTensor, 是原稠密 Tensor 的稀疏化表示。其中数据分别为:

- **indices** (Tensor) - 二维整数张量, 表示稀疏张量中 *values* 所处的位置索引。
- **values** (Tensor) - 一维张量, 用来给 *indices* 中的每个元素提供数值。
- **shape** (tuple(int)) - 整数元组, 用来指定稀疏矩阵的稠密形状。

异常:

- **ValueError** - Tensor 的 shape 不是二维。

支持平台:

GPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 0], [-5, 0]]), mindspore.float32)
>>> output = x.to_coo()
>>> print(output.indices, output.values, output.shape)
[[0 0]
 [1 0]] [ 1. -5.] (2, 2)
```

mindspore.Tensor.to_csr

`mindspore.Tensor.to_csr()`

将常规 Tensor 转为稀疏化的 CSRTensor。

说明：现在只支持二维 Tensor。

返回：

返回一个二维的 CSRTensor，是原稠密 Tensor 的稀疏化表示。其中数据分别为：

- **indptr** (Tensor) - 一维整数 Tensor，表示每行非零元素的在 *values* 中存储的起止位置。
- **indices** (Tensor) - 一维整数 Tensor，表示每个元素的列索引值。
- **values** (Tensor) - 一维 Tensor，用来表示索引对应的数值。
- **shape** (tuple(int)) - 整数元组，用来指定稀疏矩阵的稠密形状。

异常：

- **ValueError** - Tensor 的 shape 不是二维。

支持平台：

GPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([[1, 0], [-5, 0]]), mindspore.float32)
>>> output = x.to_csr()
>>> print(output.indptr, output.indices, output.values, output.shape)
[0 1 2] [0 0] [ 1. -5.] (2, 2)
```

mindspore.Tensor.tolist

`mindspore.Tensor.tolist()`

将 Tensor 转为列表，如果输入是 Tensor scalar，将会返回 Python 标量。

返回：

列表或者 Python 标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> x = ms.Tensor([[1, 2, 3], [4, 5, 6]])
>>> out1 = x.tolist()
>>> print(out1)
[[1, 2, 3], [4, 5, 6]]
>>> out2 = x[0][0].tolist()
```

(下页继续)

(续上页)

```
>>> print(out2)
1
```

mindspore.Tensor.topk

`mindspore.Tensor.topk(k, dim=-1, largest=True, sorted=True)`

沿给定维度查找 k 个最大或最小的元素，并返回其值和对应的索引。

警告： 由于不同平台上的内存布局和遍历方法不同：当 `sorted` 设置为 `False` 时，计算结果的显示顺序可能不一致。

如果 `self` 是一维 Tensor，则查找 Tensor 中 k 个最大或最小元素，并且将其值和索引输出为 Tensor。`values[k]` 是 `self` 中 k 个最大元素，同时其索引值为 `indices[k]`。

对于多维矩阵，计算给定维度中最大或最小的 k 个元素，因此：

$$values.shape = indices.shape$$

如果两个比较的元素相同，则优先返回索引值较小的元素。

参数：

- **k** (int) - 指定计算最大或最小元素的数量，必须为常量。
- **dim** (int, 可选) - 需要排序的维度。默认值：-1。
- **largest** (bool, 可选) - 如果设置为 `False`，则会返回 k 个最小元素。默认值：`True`。
- **sorted** (bool, 可选) - 如果设置为 `True`，则获取的元素将按值降序排序；如果设置为 `False`，则不对获取的元素进行排序，默认值：`True`。

返回：

由 `values` 和 `indices` 组成的 tuple。- **values** (Tensor) - 给定维度的每个切片中的 k 最大元素或最小元素。- **indices** (Tensor) - k 最大元素的对应索引。

异常：

- **TypeError** - 如果 `sorted` 不是 bool。
- **TypeError** - 如果 k 不是 int。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> x = ms.Tensor([[0.5368, 0.2447, 0.4302, 0.9673],
...                [0.4388, 0.6525, 0.4685, 0.1868],
...                [0.3563, 0.5152, 0.9675, 0.8230]], dtype=ms.float32)
>>> output = Tensor.topk(x, 2, dim=1)
>>> print(output)
(Tensor(shape=[3, 2], dtype=Float32, value=
[[ 9.67299998e-01,  5.36800027e-01],
 [ 6.52499974e-01,  4.68499988e-01],
```

(下页继续)

(续上页)

```
[ 9.67499971e-01,  8.23000014e-01]], Tensor(shape=[3, 2], dtype=Int32, value=
[[3, 0],
 [1, 2],
 [2, 3]]))
>>> output2 = Tensor.topk(x, 2, dim=1, largest=False)
>>> print(output2)
(Tensor(shape=[3, 2], dtype=Float32, value=
[[ 2.44700000e-01,  4.30200011e-01],
 [ 1.86800003e-01,  4.38800007e-01],
 [ 3.56299996e-01,  5.15200019e-01]]), Tensor(shape=[3, 2], dtype=Int32, value=
[[1, 2],
 [3, 0],
 [0, 1]]))
```

`mindspore.Tensor.topk(k, dim=None, largest=True, sorted=True)`

详情请参考 `mindspore.ops.topk()`。

mindspore.Tensor.trace

`mindspore.Tensor.trace(offset=0, axis1=0, axis2=1, dtype=None)`

在 Tensor 的对角线方向上的总和。

参数：

- **offset** (int, 可选) - 对角线与主对角线的偏移。可以是正值或负值。默认为主对角线。
- **axis1** (int, 可选) - 二维子数组的第一轴，对角线应该从这里开始。默认为第一轴 (0)。
- **axis2** (int, 可选) - 二维子数组的第二轴，对角线应该从这里开始。默认为第二轴。
- **dtype** (mindspore.dtype, 可选) - 默认值为 None。覆盖输出 Tensor 的 dtype。

返回：

Tensor，对角线方向上的总和。

异常：

- **ValueError** - 输入 Tensor 的维度少于 2。

参见：

- `mindspore.Tensor.diagonal()`：返回特定对角线方向上的数据。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.eye(3, dtype=np.float32))
>>> print(x.trace())
3.0
```

mindspore.Tensor.transpose

`mindspore.Tensor.transpose(dim0, dim1)`

通过给定的维度对输入 Tensor 进行转置。

警告：这是一个实验性 API，可能会更改或删除。

参数：

- **dim0** (int) - 指定第一个需要被转置的维度。
- **dim1** (int) - 指定第二个需要被转置的维度。

返回：

转置后的 Tensor，与输入具有相同的数据类型。

异常：

- **TypeError** - 如果 *dim0* 或者 *dim1* 的数据类型不是 int。
- **ValueError** - 如果 *dim0* 或者 *dim1* 的数值超出了范围： $[-ndim, ndim - 1]$ 。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.ones((2,3,4), dtype=np.float32))
>>> output = Tensor.transpose(input, 0, 2)
>>> print(output.shape)
(4, 3, 2)
```

`mindspore.Tensor.transpose(*axes)`

根据指定的排列对输入的 Tensor 进行数据重排。

此函数对于一维数组转置后不产生变化。对于一维数组转为二维列向量，请参照：[mindspore.ops.expand_dims\(\)](#)。对于二维数组可以看做是标准的矩阵转置。对于 n 维数组，根据指定的轴进行排列。如果没有指定轴并且 *a.shape* 为： $(i[0], i[1], \dots, i[n-2], i[n-1])$ ，那么 *a.transpose().shape* 为： $(i[n-1], i[n-2], \dots, i[1], i[0])$ 。

说明：GPU 和 CPU 平台上，如果 *axes* 的元素值为负数，则其实际值为 $axes[i] + rank(self)$ 。

参数：

- **axes** (tuple[int]) - 指定转置的排列。*axes* 中的元素由 *self* 的每个维度的索引组成。*axes* 的长度和 *self* 的 *shape* 相同。只支持常量值。其范围在 $[-rank(self), rank(self))$ 内。

返回：

Tensor。输出 Tensor 的数据类型与输入相同，输出 Tensor 的 *shape* 由 *self* 的 *shape* 和 *axes* 的值决定。

异常：

- **TypeError** - 如果 *axes* 的类型不为 tuple。
- **ValueError** - 如果 *self* 的 *shape* 长度不等于 *axes* 的长度。

- **ValueError** - 如果 *axes* 中存在相同的元素。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input = Tensor(np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]), mindspore.
↳float32)
>>> axes = (0, 2, 1)
>>> output = Tensor.transpose(input, axes)
>>> print(output)
[[[ 1.  4.]
  [ 2.  5.]
  [ 3.  6.]]
 [[ 7. 10.]
  [ 8. 11.]
  [ 9. 12.]]]
```

mindspore.Tensor.triangular_solve

`mindspore.Tensor.triangular_solve(A, upper=True, transpose=False, unitriangular=False)`

详情请参考[`mindspore.mint.triangular\_solve\(\)`](#)。

mindspore.Tensor.tril

`mindspore.Tensor.tril(diagonal=0)`

详情请参考[`mindspore.ops.tril\(\)`](#)。

mindspore.Tensor.triu

`mindspore.Tensor.triu(diagonal=0)`

详情请参考[`mindspore.ops.triu\(\)`](#)。

mindspore.Tensor.true_divide

`mindspore.Tensor.true_divide()`

[`mindspore.Tensor.div\(\)`](#) 中 `rounding_mode = None` 的别名。更多参考详见[`mindspore.ops.div\(\)`](#)。

mindspore.Tensor.trunc

`mindspore.Tensor.trunc()`

详情请参考[mindspore.ops.trunc\(\)](#)。

mindspore.Tensor.type

`mindspore.Tensor.type(dtype=None)`

将 Tensor 的 dtype 转换成 *dtype*。如果 *dtype* 为 None，则返回原类型。

参数：

- **dtype** (`mindspore.dtype`，可选) - 指定输出 Tensor 的数据类型。默认值：None。

返回：

Tensor 或 str。如果 *dtype* 为 None 则返回 str，str 描述了 Tensor 的数据类型。如果 *dtype* 不为 None，则返回 Tensor，返回 Tensor 的 dtype 是 *dtype*。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor([[1.2, 2], [3.4, 4]], dtype=mindspore.float32)
>>> print(x.type())
Float32
>>> print(x.type(dtype=mindspore.int32))
[[1 2]
 [3 4]]
```

mindspore.Tensor.type_as

`mindspore.Tensor.type_as(other)`

将第一个输入的 Tensor 的数据类型转换为第二个输入的 Tensor 的数据类型。

警告： 这是一个实验性 API，后续可能修改或删除。

说明：将复数转换为 bool 类型的时候，不考虑复数的虚部，只要实部不为零，返回 True，否则返回 False。

参数：

- **other** (Tensor) - 数据类型为指定类型的 Tensor，其 shape 为 $(x_0, x_1, \dots, x_R)$ 。

返回：

Tensor，其 shape 与输入 Tensor 相同，即 $(x_0, x_1, \dots, x_R)$ 。

异常：

- **TypeError** - *other* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_np = np.random.randn(2, 3, 4, 5).astype(np.float32)
>>> self = Tensor(input_np)
>>> other_np = np.random.randn(2, 3, 4).astype(np.int32)
>>> other = Tensor(other_np)
>>> output = self.type_as(other)
>>> print(output.dtype)
Int32
>>> print(output.shape)
(2, 3, 4, 5)
```

mindspore.Tensor.unbind

`mindspore.Tensor.unbind(dim=0)`

详情请参考[`mindspore.ops.unbind\(\)`](#)。

mindspore.Tensor.unfold

`mindspore.Tensor.unfold(kernel_size, dilation=1, padding=0, stride=1)`

详情请参考[`mindspore.ops.unfold\(\)`](#)。

警告：这是一个实验性 API，后续可能修改或删除。

mindspore.Tensor.uniform

`mindspore.Tensor.uniform(from_=0., to=1., generator=None)`

在半开区间 $[from_, to)$ 内生成服从均匀分布的随机数。

$$P(x) = \frac{1}{to - from_}$$

参数：

- **from_** (number) - 区间的下界。
- **to** (number) - 区间的上界。
- **generator** (Generator, 可选) - 随机种子。默认值：None。

返回：

Tensor，与输入张量形状相同。

异常：

- **TypeError** - 如果 `from_` 大于 `to`。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> x = mindspore.ops.ones((4, 2))
>>> generator = mindspore.Generator()
>>> generator.manual_seed(100)
>>> output = x.uniform(1., 2., generator)
>>> print(output.shape)
(4, 2)
```

`mindspore.Tensor.uniform_`

`mindspore.Tensor.uniform_(from_=0, to=1, *, generator=None)`

通过在半开区间 $[from_, to)$ 内生成服从均匀分布的随机数来原地更新输入 Tensor。

$$P(x) = \frac{1}{to - from_}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **from_** (Union[number.Number, Tensor], 可选) - 均匀分布的下界，可以是一个标量值或只有单个元素的任意维度的 Tensor，默认值：0。
- **to** (Union[number.Number, Tensor], 可选) - 均匀分布的上界，可以是一个标量值或只有单个元素的任意维度的 Tensor，默认值：1。

关键字参数：

- **generator** (`mindspore.Generator`, 可选) - 伪随机数生成器。默认值：None，使用默认伪随机数生成器。

返回：

返回输入 tensor。

异常：

- **TypeError** - `from_` 或 `to` 既不是 number 也不是 Tensor。
- **TypeError** - `from_` 或 `to` 为 Tensor 类型，但数据类型不是 bool、int8、int16、int32、int64、uint8、float32、float64 之一。
- **ValueError** - `from_` 或 `to` 为 Tensor 类型，但是有多个元素。
- **RuntimeError** - 如果 `from_` 大于 `to`。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> x = mindspore.ops.ones((4, 2))
>>> generator = mindspore.Generator()
>>> generator.manual_seed(100)
>>> output = x.uniform_(1., 2., generator=generator)
>>> print(output.shape)
(4, 2)
```

mindspore.Tensor.unique

`mindspore.Tensor.unique` (*sorted=True, return_inverse=False, return_counts=False, dim=None*)

对 *self* 中的元素去重。

在 *return_inverse=True* 时, 会返回一个索引 Tensor, 包含 *self* 中的元素在输出 Tensor 中的索引。

在 *return_counts=True* 时, 会返回一个 Tensor, 表示输出元素在 *self* 中的个数。

参数:

- **sorted** (bool, 可选) - 输出是否需要升序排序。默认值: `True`。
- **return_inverse** (bool, 可选) - 是否输出 *self* 在 *output* 上对应的 *index*。默认值: `False`。
- **return_counts** (bool, 可选) - 是否输出 *output* 中元素的数量。默认值: `False`。
- **dim** (int, 可选) - 做去重操作的维度。当 *dim* 为 `None` 时, 对展开的 *self* 做去重操作, 否则, 将给定维度的 Tensor 视为一个元素做去重操作。默认值: `None`。

返回:

输出为一个 Tensor, 或者以下一个或几个 Tensor 的集合: (*output*, *inverse_indices*, *counts*)。

- **output** (Tensor) - 与 *self* 数据类型相同的 Tensor, 包含 *self* 中去重后的元素。
- **inverse_indices** (Tensor, 可选) - 当 *return_inverse=True* 时返回, 表示 *self* 中的元素在输出 Tensor 中的索引。当 *dim=None* 时, *shape* 和 *self* 相同; 否则, *shape* 为 *self.shape[dim]*。
- **counts** (Tensor, 可选) - 当 *return_counts=True* 时返回, 表示输出 Tensor 中元素在 *self* 中的数量。当 *dim=None* 时, *shape* 和 *output* 相同; 否则, *shape* 是 *output.shape[dim]*。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> x = Tensor(np.array([1, 2, 5, 2]), mindspore.int32)
>>> output = x.unique(return_inverse=True)
>>> print(output)
(Tensor(shape=[3], dtype=Int32, value= [1, 2, 5]), Tensor(shape=[4], dtype=Int64, value= [0, 1, 2, 1]))
>>> y = output[0]
>>> print(y)
[1 2 5]
>>> idx = output[1]
>>> print(idx)
[0 1 2 1]
```

mindspore.Tensor.unique_consecutive

`mindspore.Tensor.unique_consecutive (return_inverse=False, return_counts=False, dim=None)`

详情请参考[mindspore.ops.unique_consecutive\(\)](#)。

mindspore.Tensor.unsorted_segment_max

`mindspore.Tensor.unsorted_segment_max (segment_ids, num_segments)`

详情请参考[mindspore.ops.unsorted_segment_max\(\)](#)。

mindspore.Tensor.unsorted_segment_min

`mindspore.Tensor.unsorted_segment_min (segment_ids, num_segments)`

详情请参考[mindspore.ops.unsorted_segment_min\(\)](#)。

mindspore.Tensor.unsorted_segment_prod

`mindspore.Tensor.unsorted_segment_prod (segment_ids, num_segments)`

详情请参考[mindspore.ops.unsorted_segment_prod\(\)](#)。

mindspore.Tensor.unsqueeze

`mindspore.Tensor.unsqueeze (dim)`

详情请参考[mindspore.ops.unsqueeze\(\)](#)。

mindspore.Tensor.var

`mindspore.Tensor.var (axis=None, ddof=0, keepdims=False)`

在指定维度上的方差。

方差是平均值的平方偏差的平均值，即： $var = mean(abs(x - x.mean()) ** 2)$ 。

返回方差值，默认情况下计算展开 Tensor 的方差，否则在指定维度上计算。

说明：不支持 NumPy 参数 *dtype*、*out* 和 *where*。

参数：

- **axis** (Union[None, int, tuple(int)], 可选) - 维度，在指定维度上计算方差。其默认值是展开 Tensor 的方差。默认值：None。
- **ddof** (int, 可选) - δ 自由度。默认值：0。计算中使用的除数是 $N - ddof$ ，其中 N 表示元素的数量。
- **keepdims** (bool, 可选) - 是否保留输出 Tensor 的维度。如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值：False。

返回：

含有方差值的 Tensor。

异常:

- **TypeError** - 如果 *axis* 不是 `None`, `int` 或 `tuple` 类型。
- **TypeError** - 如果 *ddof* 不是 `int` 类型。
- **TypeError** - 如果 *keepdims* 不是 `bool` 类型。
- **ValueError** - 如果 *axis* 不在 `[-self.ndim, self.ndim)` 范围内。

参见:

- `mindspore.Tensor.mean()`: 通过对 `Tensor` 中的所有元素求平均值来减少 `Tensor` 的维数。
- `mindspore.Tensor.std()`: 计算沿指定轴的标准差。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> input_x = Tensor(np.array([1., 2., 3., 4.], np.float32))
>>> output = input_x.var()
>>> print(output)
1.25
```

`Tensor.var(dim=None, *, correction=1, keepdim=False) → Tensor`

计算指定维度 *dim* 上的方差。*dim* 可以是单个维度、维度列表, 也可以是 `None`, 表示移除所有维度。

方差 (δ^2) 计算如下:

$$\delta^2 = \frac{1}{\max(0, N - \delta N)} \sum_{i=0}^{N-1} (x_i - \bar{x})^2$$

其中 *x* 表示用来计算方差的样本集, $\bar{x}$ 表示样本的均值, *N* 表示样本的数量, δN 则为 *correction* 的值。

参数:

- **dim** (`None`, `int`, `tuple(int)`, 可选) - 用来进行规约计算的维度。默认值为 `None`, 所有维度都进行规约计算。

关键字参数:

- **correction** (`int`, 可选) - 样本大小和样本自由度之间的差异。默认为 **Bessel** 校正, 默认值为 1。
- **keepdim** (`bool`, 可选) - 是否保留输出 `Tensor` 的维度。如果为 `True`, 则保留缩小的维度, 其大小为 1, 否则移除维度。默认值为 `False`。

返回:

`Tensor`, 方差。假设 *self Tensor* 的 *shape* 为 $(x_0, x_1, \dots, x_R)$:

- 如果 *dim* 为 `()`, 且 *keepdim* 为 `False`, 则返回一个零维 `Tensor`, 表示 *self Tensor* 中所有元素的方差。
- 如果 *dim* 为 `int`, 如取值为 1, 且 *keepdim* 为 `False`, 则返回 `Tensor` 的 *shape* 为 $(x_0, x_2, \dots, x_R)$ 。
- 如果 *dim* 为 `tuple(int)` 或者 `list(int)`, 如取值为 $(1, 2)$, 且 *keepdim* 为 `False`, 则返回 `Tensor` 的 *shape* 为 $(x_0, x_3, \dots, x_R)$ 。

异常:

- **TypeError** - 如果 *dim* 不是 None, int, list 或 tuple 类型。
- **TypeError** - 如果 *correction* 不是 int 类型。
- **TypeError** - 如果 *keepdim* 不是 bool 类型。
- **ValueError** - 如果 *dim* 不在 $[-self.ndim, self.ndim)$ 范围内。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = Tensor([[8, 2, 1], [5, 9, 3], [4, 6, 7]], mindspore.float32)
>>> output = input_x.var(dim=0, correction=1, keepdim=True)
>>> print(output)
[[ 4.333333, 12.333333, 9.333333]]
```

mindspore.Tensor.view

`mindspore.Tensor.view(*shape)`

根据输入 *shape* 重新创建一个 Tensor, 与原 Tensor 数据相同。该方法与 `reshape` 方法相同, 都是依靠底层 `reshape` 算子实现的。

参数:

- **shape** (Union[tuple(int), int]) - 输出 Tensor 的维度。

返回:

Tensor, 具有与入参 *shape* 相同的维度。

样例:

```
>>> from mindspore import Tensor
>>> import numpy as np
>>> a = Tensor(np.array([[1, 2, 3], [2, 3, 4]], dtype=np.float32))
>>> output = a.view((3, 2))
>>> print(output)
[[1. 2.]
 [3. 2.]
 [3. 4.]]
```

mindspore.Tensor.view_as

`mindspore.Tensor.view_as(other)`

根据 *other* 的 *shape* 改变原 *self* Tensor 的 *shape*。

参数:

- **other** (Tensor) - 返回 Tensor 的 *shape* 与 *other* 的 *shape* 一致。

返回:

Tensor, 和 *other* 具有相同的 *shape*。

异常:

- **TypeError** - *other* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor
>>> from mindspore import dtype as mstype
>>> a = Tensor([[1, 2, 3], [2, 3, 4]], mstype.float32)
>>> b = Tensor([1, 1, 1, 1, 1, 1], mstype.float32)
>>> output = a.view_as(b)
>>> print(output)
[1. 2. 3. 2. 3. 4.]
```

mindspore.Tensor.vsplit

`mindspore.Tensor.vsplit (indices_or_sections)`

详情请参考[`mindspore.ops.vsplit\(\)`](#)。

mindspore.Tensor.where

`mindspore.Tensor.where (condition, y)`

详情请参考[`mindspore.ops.where\(\)`](#)。

mindspore.Tensor.xdivy

`mindspore.Tensor.xdivy (y)`

详情请参考[`mindspore.ops.xdivy\(\)`](#)。

mindspore.Tensor.xlogy

`mindspore.Tensor.xlogy (other)`

详情请参考[`mindspore.ops.xlogy\(\)`](#)。当前接口的参数 *y* 和参考接口中的参数 *other* 实际为同一参数。

mindspore.Tensor.zero_

`mindspore.Tensor.zero_()`

返回填充 0 的 Tensor。

警告： 这是一个实验性 API，后续可能修改或删除。

返回：

返回一个 Tensor，该 Tensor 填充为零。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> x = Tensor(np.array([2, 2]))
>>> output = x.zero_()
>>> print(output)
[0 0]
```

mindspore.tensor

`mindspore.tensor(input_data=None, dtype=None, shape=None, init=None, const_arg=False)`

此接口用于在 `Cell.construct()` 或者 `@jit` 装饰的函数内，创建一个新的 Tensor 对象。

在图模式下，MindSpore 可以在运行时依据 `dtype` 参数来动态创建新 Tensor。

详情请参考教程 [创建和使用 Tensor](#)。

此接口与 Tensor 类的区别为内部增加了 `Annotation` 指示当前创建的 Tensor 的类型，与 Tensor 类相比能够防止 AnyType 的产生。

参数和返回值与 Tensor 类完全一致。另参考：[mindspore.Tensor](#)。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import jit, tensor
>>> @jit
... def func(x):
...     return tensor(x.asnumpy(), dtype=ms.float32)
>>> x = tensor([1, 2, 3])
>>> y = func(x)
>>> print(y)
[1. 2. 3.]
```

mindspore.COOTensor

`class mindspore.COOTensor(indices=None, values=None, shape=None, coo_tensor=None)`

用来表示某一 Tensor 在给定索引上非零元素的集合，其中索引 (`indices`) 指示了每一个非零元素的位置。

对一个稠密 Tensor *dense* 来说，它对应的 COOTensor(`indices`, `values`, `shape`)，满足 `dense[indices[i]] = values[i]`。

如果 `indices` 是 `[[0, 1], [1, 2]]`，`values` 是 `[1, 2]`，`shape` 是 `(3, 4)`，那么它对应的稠密 Tensor 如下：

```
[[0, 1, 0, 0],
 [0, 0, 2, 0],
 [0, 0, 0, 0]]
```

COOTensor 的算术运算包括：加 (+)、减 (-)、乘 (*)、除 (/)。详细的算术运算支持请参考 [运算符](#)。

警告：

- 这是一个实验性 API，后续可能修改或删除。
- 使用 PyNative 模式，需要添加“export MS_PYNATIVE_CONFIG_STATIC_SHAPE=1”。
- 目前 COOTensor 中相同索引的值不会进行合并。如果索引中包含界外值，则得出未定义结果。

参数：

- **indices** (Tensor) - shape 为 $(N, ndims)$ 的二维整数 Tensor，其中 N 和 $ndims$ 分别表示稀疏 Tensor 中 *values* 的数量和 COOTensor 维度的数量。目前 $ndims$ 只能为 2。请确保 *indices* 的值在所给 *shape* 范围内。支持的数据类型为 int16、int32 和 int64。默认值：None。
- **values** (Tensor) - shape 为 (N) 的一维 Tensor，用来给 *indices* 中的每个元素提供数值。默认值：None。
- **shape** (tuple(int)) - shape 为 $(ndims)$ 的整数元组，用来指定稀疏矩阵的稠密 *shape*。默认值：None。
- **coo_tensor** (COOTensor) - COOTensor 对象，用来初始化新的 COOTensor。默认值：None。

返回：

COOTensor，由 *indices*、*values* 和 *shape* 组成。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> print(x.values)
[1. 2.]
>>> print(x.indices)
[[0 1]
 [1 2]]
>>> print(x.shape)
(3, 4)
```

abs()

对所有非零元素取绝对值，并返回新的 COOTensor。

返回：

COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1, 2], [1, 0, 2]], dtype=ms.int32)
>>> values = Tensor([1, -5, -4], dtype=ms.float32)
>>> shape = (3, 3)
>>> coo_tensor = COOTensor(indices.transpose(), values, shape)
>>> res = coo_tensor.abs()
>>> print(res.values)
[1. 5. 4.]
```

add (*other*: COOTensor, *thresh*: Tensor)

与另一个 COOTensor 相加, 并返回新的 COOTensor。

参数:

- **other** (COOTensor) - 另一个操作数, 与当前操作数相加。
- **thresh** (Tensor) - 零维, 表示阈值, 用来决定 COOTensor.add 结果中的 *indice/value* 对是否出现。如果结果中 *value* 的数据类型为实数, 则 *thresh* 的数据类型应该与它的数据类型一致。如果结果中 *value* 小于 *thresh*, 它将会被丢掉。

返回:

COOTensor, 为两个 COOTensor 相加后的结果。

异常:

- **ValueError** - 如果操作数 (本 COOTensor/other) 的 *indices* 的维度不等于 2。
- **ValueError** - 如果操作数 (本 COOTensor/other) 的 *values* 的维度不等于 1。
- **ValueError** - 如果操作数 (本 COOTensor/other) 的 *shape* 的维度不等于 1。
- **ValueError** - 如果 *thresh* 的维度不等于 0。
- **TypeError** - 如果操作数 (本 COOTensor/other) 的 *indices* 的数据类型不为 int64。
- **TypeError** - 如果操作数 (本 COOTensor/other) 的 *shape* 的数据类型不为 int64。
- **ValueError** - 如果操作数 (本 COOTensor/other) 的 *indices* 的长度不等于它的 *values* 的长度。
- **TypeError** - 如果操作数 (本 COOTensor/other) 的 *values* 的数据类型不为 int8/int16/int32/int64/float32/float64/complex64/complex128 中的任何一个。
- **TypeError** - 如果 *thresh* 的数据类型不为 int8/int16/int32/int64/float32/float64 中的任何一个。
- **TypeError** - 如果操作数 (本 COOTensor) 的 *indices* 数据类型不等于 other 的 *indices* 数据类型。
- **TypeError** - 如果操作数 (本 COOTensor) 的 *values* 数据类型不等于 other 的 *values* 数据类型。
- **TypeError** - 如果操作数 (本 COOTensor) 的 *shape* 数据类型不等于 other 的 *shape* 数据类型。
- **TypeError** - 如果操作数 (本 COOTensor/other) 的 *values* 的数据类型与 *thresh* 数据类型不匹配。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import Tensor, COOTensor
>>> from mindspore import dtype as mstype
>>> indic0 = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values0 = Tensor([1, 2], dtype=mstype.int32)
>>> shape0 = (3, 4)
>>> input0 = COOTensor(indic0, values0, shape0)
>>> indic1 = Tensor([[0, 0], [1, 1]], dtype=mstype.int64)
>>> values1 = Tensor([3, 4], dtype=mstype.int32)
>>> shape1 = (3, 4)
>>> input1 = COOTensor(indic1, values1, shape1)
>>> thres = Tensor(0, dtype=mstype.int32)
>>> out = input0.add(input1, thres)
>>> print(out)
COOTensor(shape=[3, 4], dtype=Int32, indices=Tensor(shape=[4, 2], dtype=Int64, value=
[[0 0]
```

(下页继续)

(续上页)

```
[0 1]
[1 1]
[1 2]]), values=Tensor(shape=[4], dtype=Int32, value=[3 1 4 2]))
```

astype(dtype: mstype)

返回指定数据类型的 COOTensor。

参数：

- **dtype** (Union[mindspore.dtype, numpy.dtype, str]) - 指定数据类型。

返回：

COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (3, 4)
>>> coo_tensor = COOTensor(indices, values, shape)
>>> print(coo_tensor.astype(ms.float64).dtype)
Float64
```

coalesce()

合并 COOTensor 中相同索引的值。

返回：

COOTensor。

支持平台：

GPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> x_indices = Tensor([[0, 0, 1], [1, 1, 2]], dtype=ms.int64)
>>> x_values = Tensor([1, 5, 4], dtype=ms.float32)
>>> x_shape = (3, 3)
>>> coo_tensor = COOTensor(x_indices.transpose(), x_values, x_shape)
>>> res = coo_tensor.coalesce()
>>> print(res)
COOTensor(shape=[3, 3], dtype=Float32, indices=Tensor(shape=[2, 2], dtype=Int64,
value=[[0 1] [1 2]]), values=Tensor(shape=[2], dtype=Float32, value=[6.00000000e+00,
↪4.00000000e+00]))
```

property dtype

返回 COOTensor 数据类型 (*mindspore.dtype*)。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (3, 4)
>>> coo_tensor = COOTensor(indices, values, shape)
>>> print(coo_tensor.dtype)
Float32
```

property indices

返回 COOTensor 的索引值。

property itemsize

返回每个非零元素所占字节数。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float64)
>>> shape = (3, 4)
>>> coo_tensor = COOTensor(indices, values, shape)
>>> print(coo_tensor.itemsize)
8
```

property ndim

返回稀疏矩阵的稠密维度。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> coo_tensor = COOTensor(indices, values, (3, 4))
>>> print(coo_tensor.ndim)
2
```

property shape

返回稀疏矩阵的稠密 shape。

property size

返回稀疏矩阵非零元素值数量。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1, 2], [1, 0, 2]], dtype=ms.int32)
>>> values = Tensor([1, 5, 4], dtype=ms.float32)
>>> shape = (3, 3)
>>> coo_tensor = COOTensor(indices.transpose(), values, shape)
>>> print(coo_tensor.size)
3
```

to_csr()

将 COOTensor 转换为 CSRTensor。

说明：如果运行后端是 CPU，那么仅支持在安装了 LLVM12.0.1 的机器运行。

返回：

CSRTensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.int32)
>>> shape = (3, 4)
>>> coo_tensor = COOTensor(indices, values, shape)
>>> print(coo_tensor.to_csr())
CSRTensor(shape=[3, 4], dtype=Int32, indptr=Tensor(shape=[4], dtype=Int32, value=[0 1 2 2]),
indices=Tensor(shape=[2], dtype=Int32, value=[1 2]), values=Tensor(shape=[2], dtype=Int32, value=[1 2]))
```

to_dense()

将 COOTensor 转换为稠密 Tensor。

返回：

Tensor。

支持平台：

GPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1, 2], [1, 0, 2]], dtype=ms.int32)
>>> values = Tensor([1, 5, 4], dtype=ms.float32)
>>> shape = (3, 3)
>>> coo_tensor = COOTensor(indices.transpose(), values, shape)
>>> print(coo_tensor.to_dense())
[[0. 1. 0.]
 [5. 0. 0.]
 [0. 0. 4.]]
```

to_tuple()

将 COOTensor 的索引、非零元素，以及 shape 信息作为 tuple 返回。

返回：

tuple(Tensor, Tensor, tuple(int))。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (3, 4)
>>> coo_tensor = COOTensor(indices, values, shape)
>>> print(coo_tensor.to_tuple())
(Tensor(shape=[2, 2], dtype=Int32, value=
[[0, 1],
 [1, 2]]), Tensor(shape=[2], dtype=Float32, value= [ 1.00000000e+00,  2.00000000e+00]),
↵ (3, 4))
```

property values

返回 COOTensor 的非零元素值。

mindspore.CSRTensor

class mindspore.CSRTensor (*indptr=None, indices=None, values=None, shape=None, csr_tensor=None*)

用来表示某一 Tensor 在给定索引上非零元素的集合，其中行索引由 *indptr* 表示，列索引由 *indices* 表示，非零值由 *values* 表示。

如果 *indptr* 是 [0, 2, 5, 6]，*indices* 是 [0, 3, 1, 2, 4, 2]，*values* 是 [1., 2., 3., 4., 5., 6.]，*shape* 是 (3, 5)，那么它对应的稠密 Tensor 如下：

```
[[1., 0., 0., 2., 0.],
 [0., 3., 4., 0., 5.],
 [0., 0., 6., 0., 0.]]
```

indptr 的长度应为 *shape[0]+1*，其中的元素应为单调递增或相等的，最大值应等于张量中所有非零元素的总个数；*indices* 和 *values* 的长度应等于张量中所有非零元素的总个数。具体地，通过 *indptr* 拿到每一行非零元素的查询索引，然后到 *indices* 中根据该行的查询索引确定该行非零元素所在列，到 *values* 中确定该行非零元素的取值。

前例中，*indptr* 为 [0, 2, 5, 6]，表示查询索引 [0, 2) 为第 0 行的数据，查询索引 [2, 5) 为第 1 行的数据，查询索引 [5, 6) 为第 2 行的数据。例如：

- 稠密张量第 0 行非零元素所在的列位置由 *indices* 中的第 [0, 2) 个元素（即 [0, 3]）给出，实际值由 *values* 中的第 [0, 2) 个元素（即 [1., 2.]）给出。
- 稠密张量第 1 行非零元素所在的列位置由 *indices* 中的第 [2, 5) 个元素（即 [1, 2, 4]）给出，实际值由 *values* 中的第 [2, 5) 个元素（即 [3., 4., 5.]）给出。
- 稠密张量第 2 行非零元素所在的列位置由 *indices* 中的第 [5, 6) 个元素（即 [2]）给出，实际值由 *values* 中的第 [5, 6) 个元素（即 [6.]）给出。

CSRTensor 的算术运算包括：加 (+)、减 (-)、乘 (*)、除 (/)。详细的算术运算支持请参考 [运算符](#)。

警告：

- 这是一个实验性 API，后续可能修改或删除。
- 使用 PyNative 模式，需要添加“export MS_PYNATIVE_CONFIG_STATIC_SHAPE=1”。
- 如果 *indptr* 或 *indices* 的值不合法，行为将没有定义。不合法的值包括当 *values* 或 *indices* 的长度超出了 *indptr* 所指定的取值范围，以及当 *indices* 在同一行中出现重复的列。

参数：

- **indptr** (Tensor) - shape 为 (M) 的一维整数 Tensor, 其中 M 等于 $shape[0] + 1$, 表示每行非零元素在 *values* 中存储的起止位置。默认值: None。支持的数据类型为 int16、int32 和 int64。
- **indices** (Tensor) - shape 为 (N) 的一维整数 Tensor, 其中 N 等于非零元素数量, 表示每个元素的列索引值。默认值: None。支持的数据类型为 int16、int32 和 int64。
- **values** (Tensor) - Tensor, *values* 的零维长度必须与 *indices* 的零维长度相等 ($values.shape[0] == indices.shape[0]$)。 *values* 用来表示索引对应的数值。默认值: None。
- **shape** (tuple(int)) - shape 为 ($ndims$) 的整数元组, 用来指定稀疏矩阵的稠密 shape。 $shape[0]$ 表示行数, 因此必须和 $M - 1$ 值相等。默认值: None。
- **csr_tensor** (CSRTensor) - CSRTensor 对象, 用来初始化新的 CSRTensor。 *values* 的特征维度需要和 *csr_tensor* 的特征维度匹配 ($values.shape[1:] == csr\_tensor.shape[2:]$)。默认值: None。

输出:

CSRTensor, 稠密 shape 取决于传入的 *shape*, 数据类型由 *values* 决定。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> # initialize a csr_tensor with indptr, indices, values and shape
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> # access a data member of CSRTensor
>>> print(indptr == csr_tensor.indptr)
[ True True True]
```

abs()

对所有非零元素取绝对值, 并返回新的 CSRTensor。

返回:

CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([-1, -2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.abs().values)
[1. 2.]
```

add (*b*: CSRTensor, *alpha*: Tensor, *beta*: Tensor)

两个 CSRTensor 求和: $C = \alpha * a + \beta * b$ 。

参数:

- **b** (CSRTensor) - 稀疏 CSRTensor。

- **alpha** (Tensor) - 稠密 Tensor, shape 必须可以广播给 self。
- **beta** (Tensor) - 稠密 Tensor, shape 必须可以广播给 *b*。

返回:

CSRTensor, 求和。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import Tensor, CSRTensor
>>> import mindspore.common.dtype as mstype
>>> indptr = Tensor([0, 1, 2], dtype=mstype.int32)
>>> indices = Tensor([0, 1], dtype=mstype.int32)
>>> values_a = Tensor([2, 1], dtype=mstype.float32)
>>> values_b = Tensor([1, 2], dtype=mstype.float32)
>>> dense_shape = (2, 4)
>>> alpha = Tensor(1, mstype.float32)
>>> beta = Tensor(1, mstype.float32)
>>> a = CSRTensor(indptr, indices, values_a, dense_shape)
>>> b = CSRTensor(indptr, indices, values_b, dense_shape)
>>> print(a.add(b, alpha, beta))
CSRTensor(shape=[2, 4], dtype=Float32,
          indptr=Tensor(shape=[3], dtype=Int32, value=[0 1 2]),
          indices=Tensor(shape=[2], dtype=Int32, value=[0 1]),
          values=Tensor(shape=[2], dtype=Float32, value=[ 3.00000000e+00  3.
->00000000e+00]))
```

astype (*dtype: mstype*)

返回指定数据类型的 CSRTensor。

参数:

- **dtype** (Union[mindspore.dtype, numpy.dtype, str]) - 指定数据类型。

返回:

CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.astype(ms.float64).dtype)
Float64
```

property dtype

返回稀疏矩阵非零元素值数据类型 (*mindspore.dtype*)。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.dtype)
Float32
```

property indices

返回 CSRTensor 的列索引值。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.indices)
[0 1]
```

property indptr

返回 CSRTensor 的行偏移量。

property itemsize

返回每个非零元素所占字节数。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float64)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.itemsize)
8
```

mm (matrix: Union[[Tensor](#), [CSRTensor](#)])

返回 CSRTensor 右乘稀疏矩阵或稠密矩阵的矩阵乘法运算结果。

shape 为 $[M, N]$ 的 CSRTensor，需要适配 shape 为 $[N, K]$ 的稠密矩阵或稀疏矩阵，得到结果为 $[M, K]$ 的稠密矩阵或稀疏矩阵。

说明：

- 若右矩阵为 Tensor，则仅支持安装了 LLVM12.0.1 及以上版本的 CPU 后端或 GPU 后端。
- 若右矩阵为 CSRTensor，则仅支持 GPU 后端。

参数：

- **matrix** (Tensor or CSRTensor) - shape 为 $[N, K]$ 的二维矩阵，其中 N 等于 CSRTensor 的列数。

返回:

Tensor 或者 CSRTensor。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import Tensor, CSRTensor
>>> from mindspore import dtype as mstype
>>> indptr = Tensor([0, 1, 2], dtype=mstype.int32)
>>> indices = Tensor([0, 1], dtype=mstype.int32)
>>> values = Tensor([2, 1], dtype=mstype.float32)
>>> dense_shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, dense_shape)
>>> dense_matrix = Tensor([[1., 2.], [1, 2.], [1, 2.], [1., 2.]], dtype=mstype.float32)
>>> print(csr_tensor.mm(dense_matrix))
[[2. 4.]
 [1. 2.]]
```

mv (*dense_vector*: Tensor)

返回 CSRTensor 右乘稠密矩阵的矩阵乘法运算结果。shape 为 $[M, N]$ 的 CSRTensor, 需要适配 shape 为 $[N, 1]$ 的稠密向量, 得到结果为 $[M, 1]$ 的稠密向量。

说明: 如果运行后端是 CPU, 那么仅支持在安装了 LLVM12.0.1 的机器运行。

参数:

- **dense_vector** (Tensor) - shape 为 $[N, 1]$ 的二维 Tensor, 其中 N 等于 CSRTensor 的列数。

返回:

Tensor。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import Tensor, CSRTensor
>>> from mindspore import dtype as mstype
>>> indptr = Tensor([0, 1, 2], dtype=mstype.int32)
>>> indices = Tensor([0, 1], dtype=mstype.int32)
>>> values = Tensor([2, 1], dtype=mstype.float32)
>>> dense_shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, dense_shape)
>>> dense = Tensor([[1], [1], [1], [1]], dtype=mstype.float32)
>>> print(csr_tensor.mv(dense))
[[2.]
 [1.]]
```

property ndim

返回稀疏矩阵的稠密维度。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.ndim)
2
```

property shape

返回稀疏矩阵的稠密 shape。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.shape)
(2, 4)
```

property size

返回稀疏矩阵非零元素值数量。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.size)
2
```

sum (*axis: int*)

对 CSRTensor 的某个轴求和。

说明：如果运行后端是 CPU，那么仅支持在安装了 LLVM12.0.1 的机器运行。

参数：

- **axis** (int) - 求和轴。

返回：

Tensor。

支持平台：

GPU CPU

样例：

```
>>> from mindspore import Tensor, CSRTensor
>>> from mindspore import dtype as mstype
>>> indptr = Tensor([0, 1, 2], dtype=mstype.int32)
>>> indices = Tensor([0, 1], dtype=mstype.int32)
>>> values = Tensor([2, 1], dtype=mstype.float32)
>>> dense_shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, dense_shape)
>>> print(csr_tensor.sum(1))
[[2.]
 [1.]]
```

to_coo()

将 CSRTensor 转换为 COOTensor。

说明：如果运行后端是 CPU，那么仅支持在安装了 LLVM12.0.1 的机器运行。

返回：

COOTensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.int32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.to_coo())
COOTensor(shape=[2, 4], dtype=Int32, indices=Tensor(shape=[2, 2], dtype=Int32, value=
[[0 0]
 [1 1]]), values=Tensor(shape=[2], dtype=Int32, value=[1 2]))
```

to_dense()

将 CSRTensor 转换为稠密 Tensor。

返回：

Tensor。

支持平台：

GPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
```

(下页继续)

(续上页)

```
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.to_dense())
[[1. 0. 0. 0.]
 [0. 2. 0. 0.]]
```

to_tuple()

将 CSRTensor 的行偏移量、列索引、非零元素，以及 shape 信息作为 tuple 返回。

返回：

tuple(Tensor, Tensor, Tensor, tuple(int))。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.to_tuple())
(Tensor(shape=[3], dtype=Int32, value= [0, 1, 2]), Tensor(shape=[2], dtype=Int32, value=
[0, 1]),
Tensor(shape=[2], dtype=Float32, value= [ 1.00000000e+00,  2.00000000e+00]), (2, 4))
```

property values

返回 CSRTensor 的非零元素值。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2], dtype=ms.int32)
>>> indices = Tensor([0, 1], dtype=ms.int32)
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (2, 4)
>>> csr_tensor = CSRTensor(indptr, indices, values, shape)
>>> print(csr_tensor.values)
[1. 2.]
```

mindspore.RowTensor

class mindspore.RowTensor (indices=None, values=None, shape=None, row_tensor=None)

用来表示一组指定索引的 Tensor 切片的稀疏表示。

若 RowTensor 的 values 的 shape 为 $(d_0, d_1, \dots, d_n)$ ，则该 RowTensor 用于表示一个有着 shape 为 $(l_0, d_1, \dots, d_n)$ 的更大的稠密 Tensor 的子集，其中 d_i 为 RowTensor 第 i 轴的 size， l_0 为稠密 Tensor 在第 0 轴的 size，并且 $l_0 > d_0$ 。

其中，参数 indices 用于指定 RowTensor 从该稠密 Tensor 的第一维度的哪些位置来进行切片，即参数 indices 和 values 满足以下关系： $dense[indices[i], :, :, \dots] = values[i, :, :, \dots]$ 。

如果 indices 是 [0]，values 是 [[1, 2]]，shape 是 (3, 2)，那么它对应的稠密 Tensor 如下：

```
[[1, 2],
 [0, 0],
 [0, 0]]
```

警告:

- 这是一个实验性 API，后续可能修改或删除。
- 使用 PyNative 模式，需要添加“export MS_PYNATIVE_CONFIG_STATIC_SHAPE=1”。

参数:

- **indices** (Tensor) - shape 为 (d_0) 的一维整数 Tensor。默认值: None。
- **values** (Tensor) - shape 为 $(d_0, d_1, \dots, d_n)$ 中任意类型的 Tensor。默认值: None。
- **shape** (tuple(int)) - 包含相应稠密 Tensor shape 的整数元组。默认值: None。
- **row_tensor** (RowTensor) - RowTensor 对象，用来初始化新的 RowTensor。默认值: None。

返回:

RowTensor，由 *indices*、*values* 和 *shape* 组成。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, RowTensor
>>> indices = Tensor([0])
>>> values = Tensor([[1, 2]], dtype=ms.float32)
>>> shape = (3, 2)
>>> x = RowTensor(indices, values, shape)
>>> print(x.values)
[[1. 2.]]
>>> print(x.indices)
[0]
>>> print(x.dense_shape)
(3, 2)
```

mindspore.SparseTensor

class mindspore.SparseTensor(*indices, values, shape*)

用来表示某一张量在给定索引上非零元素的集合。

SparseTensor 只能在 *Cell* 的构造方法中使用。

说明: 此接口从 1.7 版本开始弃用，并计划在将来移除。请使用 *mindspore.COOTensor*。

对于稠密张量，其 *SparseTensor(indices, values, shape)* 具有 *dense[indices[i]] = values[i]*。

参数:

- **indices** (Tensor) - shape 为 $(N, ndims)$ 的二维整数张量，其中 N 和 $ndims$ 分别表示稀疏张量中 *values* 的数量和 *SparseTensor* 维度的数量。

- **values** (Tensor) - shape 为 (N) 的一维张量, 其内部可以为任何数据类型, 用来给 *indices* 中的每个元素提供数值。
- **shape** (tuple(int)) - shape 为 $(ndims)$ 的整数元组, 用来指定稀疏矩阵的稠密形状。

返回:

SparseTensor, 由 *indices*、*values* 和 *shape* 组成。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, SparseTensor
>>> indices = Tensor([[0, 1], [1, 2]])
>>> values = Tensor([1, 2], dtype=ms.float32)
>>> shape = (3, 4)
>>> x = SparseTensor(indices, values, shape)
>>> print(x.values)
[1. 2.]
>>> print(x.indices)
[[0 1]
 [1 2]]
>>> print(x.shape)
(3, 4)
```

property indices

返回 SparseTensor 的索引值。

property shape

返回 SparseTensor 的稠密 shape。

property values

返回 SparseTensor 的非零元素值。

mindspore.is_tensor

`mindspore.is_tensor(obj)`

判断输入对象是否为 *mindspore.Tensor*。

参数:

- **obj** (Object) - 输入对象。

返回:

Bool, 如果 *obj* 是 *mindspore.Tensor*, 返回 True, 否则返回 False。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> a = mindspore.tensor([1.9, 2.2, 3.1])
>>> mindspore.ops.is_tensor(a)
True
```


mindspore.from_numpy

`mindspore.from_numpy(array)`

将 Numpy 数组转换为张量。

当数据为非 C 连续时，数据会被拷贝成 C 连续数据后再创建张量，否则会通过不复制数据的方式将 Numpy 数组转换为张量。

参数：

- **array** (numpy.array) - 输入数组。

返回：

与输入的张量具有相同的数据类型的 Tensor。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.array([1, 2])
>>> output = ms.from_numpy(x)
>>> print(output)
[1 2]
```

1.1.2 参数

`mindspore.Parameter`

Parameter 是 *Tensor* 的子类，当它们被绑定为 *Cell* 的属性时，会自动添加到其参数列表中，并且可以通过 *Cell* 的某些方法获取，例如 `cell.get_parameters()`。

`mindspore.ParameterTuple`

继承于 *tuple*，用于管理多个 *Parameter*。

mindspore.Parameter

class `mindspore.Parameter` (*default_input*, *name=None*, *requires_grad=True*, *layerwise_parallel=False*,
parallel_optimizer=True, *storage_format=""*, *device=None*)

Parameter 是 *Tensor* 的子类，当它们被绑定为 *Cell* 的属性时，会自动添加到其参数列表中，并且可以通过 *Cell* 的某些方法获取，例如 `cell.get_parameters()`。

说明：

- 并行场景下存在参数的形状发生变化的情况，用户可以调用 *Parameter* 的 *init_data* 方法得到原始数据。
- 如果网络中存在需要部分输入为 *Parameter* 的算子，则不允许这部分输入的 *Parameter* 进行数据类型转换。
- 每一个 *Parameter* 使用唯一的名字可以帮助后续的操作和更新。如果有两个或多个 *Parameter* 在同一个网络中使用了相同的名字，将会在定义时提示使用唯一的名字。
- *Parameter* 直接打印时无法查看到里面实际包含的值，需要使用 *Parameter.asnumpy()* 方法来查看实际的值。

参数：

- **default_input** (Union[*Tensor*, int, float, numpy.ndarray, list]) - 初始化参数的输入值。

- **name** (str) - 参数的名称。默认值: None。如果一个网络中存在两个及以上相同名称的 *Parameter* 对象, 在定义时将提示设置一个特有的名称。

- 如果一个 *Parameter* 未命名, 默认的名字就是变量名。例如, *param_a* 的名字是 *name_a*, *param_b* 的名字是 *param_b*。

```
self.param_a = Parameter(Tensor([1], ms.float32), name="name_a")
self.param_b = Parameter(Tensor([2], ms.float32))
```

- 如果在 list 或 tuple 中的 *Parameter* 未命名, 将会提供一个唯一值。例如, 以下 *Parameter* 的名字是 **Parameter\$1** and **Parameter\$2**。

```
self.param_list = [Parameter(Tensor([3], ms.float32)),
                   Parameter(Tensor([4], ms.float32))]
```

- 如果 *Parameter* 已命名, 并且不同 *Parameter* 间有重复名称, 将会抛出类似“its name 'name_a' already exists.”的异常。

```
self.param_a = Parameter(Tensor([1], ms.float32), name="name_a")
self.param_tuple = (Parameter(Tensor([5], ms.float32), name="name_a"),
                    Parameter(Tensor([6], ms.float32)))
```

- 如果一个 *Parameter* 多次出现在 list 或 tuple 中, 只检查一次它的名字。例如, 以下代码将不会抛出异常。

```
self.param_a = Parameter(Tensor([1], ms.float32), name="name_a")
self.param_tuple = (self.param_a, self.param_a)
```

- **requires_grad** (bool) - 是否需要微分求梯度。默认值: True。
- **layerwise_parallel** (bool) - 在数据/混合并行模式下, *layerwise_parallel* 配置为 True 时, 参数广播和梯度聚合时会过滤掉该 *Parameter*。默认值: False。
- **parallel_optimizer** (bool) - 用于在并行模式下, 区分该参数是否进行优化器切分。仅在 *mindspore.parallel.auto_parallel.AutoParallel.hsdp()* 启用优化器并行时有效。默认值: True。
- **storage_format** (str) - 仅限 Ascend, 用于指定权重加载到设备的格式。默认不改变格式, 可选值为: "FRACTAL_NZ"、"NC1HWC0"、"FRACTAL_Z" 等。默认值: ""。
- **device** (str) - 仅限 Ascend, 用于指定存储 *Parameter* 的设备。默认情况下, *Parameter* 将在计算时存储在设备上。当 device 被指定为 "CPU" 时, *Parameter* 将在需要使用时加载到设备上, 并在使用后卸载至 CPU。仅当 *mindspore.set_context()* 中的 *memory_offload* 配置为 "ON", *jit_level* 配置为非 "O2", *memory_optimize_level* 配置为 "O0" 时生效。可以通过指定 device 为 "CPU" 节省显存。

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Parameter, Tensor, ops, nn
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.matmul = ops.MatMul()
...         self.weight = Parameter(Tensor(np.ones((1, 2)), mindspore.float32), name="w",
->requires_grad=True)
...
...     def construct(self, x):
...         out = self.matmul(self.weight, x)
```

(下页继续)

(续上页)

```

...         return out
>>> net = Net()
>>> x = Tensor(np.ones((2, 1)), mindspore.float32)
>>> print(net(x))
[[2.]]
>>> net.weight.set_data(Tensor(np.zeros((1, 2)), mindspore.float32))
>>> print(net(x))
[[0.]]

```

add_pipeline_stage(stage)

为参数添加 pipeline_stage。

参数:

- **stage** (int) - 参数的 pipeline_stage。

异常:

- **TypeError** - 如果 *stage* 不是正整数。

property cache_enable

表示该参数是否开启缓存功能。

样例:

```

>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.cache_enable=True
>>> x.cache_enable
True

```

property cache_shape

如果使用缓存, 则返回对应参数的缓存 shape。

样例:

```

>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.cache_enable=True
>>> x.cache_shape=[1, 2]
>>> x.cache_shape
[1, 2]

```

clone(init='same')

克隆参数。

参数:

- **init** (Union[*Tensor*, *str*, *numbers.Number*]) - 初始化参数的形状和数据类型。如果 *init* 是 *Tensor* 或 *numbers.Number*, 则克隆一个具有相同数值、形状和数据类型的新参数。如果 *init* 是 *str*, 则 *init* 将继承 *Initializer* 模块中对应的同名的类。例如, 如果 *init* 是 'same', 则克隆一个具有相同数据、形状和数据类型的新参数。默认值: 'same'。

返回:

Parameter, 返回克隆的新参数。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> y = x.clone()
```

property comm_fusion

获取此参数的通信算子的融合类型 (int)。

在 *AutoParallel(Cell)* 使能的并行模式下, 一些用于参数或梯度聚合的通信算子将自动插入。*comm_fusion* 的值必须大于等于 0。当 *comm_fusion* 为 0 时, 算子不会融合在一起。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.comm_fusion = 3
>>> x.comm_fusion
3
```

copy()

拷贝参数。

返回:

Parameter, 返回拷贝的新参数。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> y = x.copy()
```

property data

返回参数对象。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([[1, 2], [3, 4]], dtype=np.float32)), name="param")
>>> x.data
Parameter (name=param, shape=(2, 2), dtype=Float32, requires_grad=True)
```

init_data (layout=None, set_sliced=False)

初始化参数的数据。

参数:

- **layout** (Union[None, tuple]) - 参数的 layout 信息。layout[dev_mat, tensor_map, slice_shape, filed_size, uniform_split, opt_shard_group]。默认值: None。仅在 *SEMI_AUTO_PARALLEL* 或 *AUTO_PARALLEL* 模式下, layout 不是 None。
 - **dev_mat** (list(int)) - 该参数的设备矩阵。
 - **tensor_map** (list(int)) - 该参数的张量映射。
 - **slice_shape** (list(int)) - 该参数的切片 shape。
 - **filed_size** (int) - 该权重的行数。

- **uniform_split** (bool) - 该参数是否进行均匀切分。
- **opt_shard_group** (str) - 该参数进行优化器切分时的 group。
- **set_sliced** (bool) - 参数初始化时被设定为分片，则为 True。默认值：False。

返回：

初始化数据后的 *Parameter*。如果当前 *Parameter* 已初始化，则更新 *Parameter* 数据。

异常：

- **RuntimeError** - 参数使用 *Initializer* 模块进行初始化，初始化后并行模式发生更改。
- **ValueError** - *layout* 长度小于 6。
- **TypeError** - *layout* 不是元组。

样例：

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([[1, 2], [3, 4]], dtype=np.float32)), name="param")
>>> x.init_data()
```

property init_data

用于调用 *init_data* 后，获取当前的 *Parameter*。

如果 *self* 是没有数据的 *Parameter*，则默认返回为 None；在调用 *init_data* 方法对 *Parameter* 初始化数据后，当前 *Parameter* 会被记录在此属性中。

样例：

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.init_data()
```

property key

用于获取当前 *Parameter* 的唯一 key 值。

样例：

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.key = 2
>>> x.key
2
```

property layerwise_parallel

获取此参数的逐层并行状态 (bool)。

在 *DATA_PARALLEL* 和 *HYBRID_PARALLEL* 模式下，如果 *layerwise_parallel* 为 True，则广播和 gradients 通信将不会应用到参数。

样例：

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.layerwise_parallel = True
```

(下页继续)

(续上页)

```
>>> x.layerwise_parallel
True
```

property name

获取参数的名称。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.name = "param1"
>>> x.name
'param1'
```

property parallel_optimizer

获取此参数的优化器并行状态 (bool)。

用于在 *AutoParallel(Cell)* 使能的并行模式下, 过滤权重切分操作。当在 *mindspore.parallel.auto_parallel.AutoParallel.hsdp()* 中启用优化器并行时, 它才有效。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.parallel_optimizer = True
>>> x.parallel_optimizer
True
```

property parallel_optimizer_comm_recompute

获取此参数的优化器并行通信重计算状态 (bool)。

在 *AutoParallel(Cell)* 使能的并行模式下, 当使用并行优化器时, 会自动插入一些 *mindspore.ops.AllGather* 算子, 用于参数聚合。它用于控制这些 *mindspore.ops.AllGather* 算子的重计算属性。

说明:

- 仅支持 *Graph* 模式。
- 建议使用 *cell.recompute(parallel_optimizer_comm_recompute=True/False)* 去配置由优化器并行生成的 *mindspore.ops.AllGather* 算子, 而不是直接使用该接口。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.parallel_optimizer_comm_recompute = True
>>> x.parallel_optimizer_comm_recompute
True
```

register_hook (hook_fn)

详情请参考 *mindspore.Tensor.register_hook()*。

property requires_grad

表示该参数是否需要梯度进行更新。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.requires_grad = True
>>> x.requires_grad
True
```

set_data (*data*, *slice_shape=False*)

设置参数数据。

参数:

- **data** (Union[*Tensor*, int, float]) - 新数据。
- **slice_shape** (bool) - 如果 *slice_shape* 设为 True, 则不检查 *data* 和当前参数 *shape* 的一致性。默认值: False。当 *slice_shape* 设为 True 时, 如果两个 *shape* 不一致, 会抛出 *ValueError*。

返回:

完成数据设置的新参数。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([[1, 2], [3, 4]], dtype=np.float32)), name="param")
>>> x.set_data(Tensor(np.array([[6, 6], [6, 6]], dtype=np.float32)))
Parameter (name=param, shape=(2, 2), dtype=Float32, requires_grad=True)
```

set_param_ps (*init_in_server=False*)

表示可训练参数是否由参数服务器更新, 以及可训练参数是否在服务器上初始化。

说明: 仅当运行的任务处于参数服务器模式下有效。只支持在图模式下调用。

参数:

- **init_in_server** (bool) - 表示参数服务器更新的可训练参数是否在服务器上初始化。默认值: False。

property sliced

获取参数的切片状态。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.sliced = True
>>> x.sliced
True
```

property unique

表示参数是否唯一。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x.unique = True
>>> x.unique
True
```

value()

返回参数的值。

样例:

```
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([1, 2], dtype=np.float32)), name="param")
>>> x_value = x.value()
>>> print(x_value)
[1.  2.]
```

mindspore.ParameterTuple

class mindspore.ParameterTuple

继承于 tuple，用于管理多个 Parameter。

说明：该类把网络参数存储到参数元组集合中。

样例:

```
>>> from mindspore import Tensor, Parameter, ParameterTuple
>>> import numpy as np
>>> x = Parameter(Tensor(np.array([[1, 2], [3, 4]], dtype=np.float32)), name="param")
>>> y = Parameter(Tensor(np.array([[5, 6], [7, 8]], dtype=np.float32)), name="param1")
>>> pt = ParameterTuple([x, y])
>>> pt1 = pt.clone(prefix="new")
```

clone (*prefix*, *init*='same')

逐个对 ParameterTuple 中的 Parameter 进行克隆，生成新的 ParameterTuple。

参数:

- **prefix** (str) - Parameter 的 namespace，此前缀将会被添加到 ParameterTuple 中的 Parameter 的 name 属性中。
- **init** (Union[*Tensor*, str, numbers.Number]) - 对 ParameterTuple 中 Parameter 的 shape 和类型进行克隆，并根据传入的 *init* 设置数值。默认值: 'same'。
 - 如果 *init* 是 *Tensor*，则新参数的数值与该 *Tensor* 相同。
 - 如果 *init* 是 *str*，则按照 *Initializer* 模块中对应的同名的初始化方法设置新参数的数值。若 *init* 的值为 'same'，则新参数的数值与原 Parameter 相同。
 - 如果 *init* 是 *numbers.Number*，则设置新参数的数值为该值。

返回:

新的参数元组。

1.1.3 数据类型

<code>mindspore.dtype</code>	MindSpore 的数据类型。
<code>mindspore.dtype_to_nptype</code>	将 MindSpore 数据类型转换成 NumPy 数据类型。
<code>mindspore.dtype_to_pytype</code>	将 MindSpore 数据类型转换为 Python 数据类型。
<code>mindspore.pytype_to_dtype</code>	将 Python 数据类型转换为 MindSpore 数据类型。
<code>mindspore.get_py_obj_dtype</code>	获取与 Python 数据类型对应的 MindSpore 数据类型。
<code>mindspore.QuantDtype</code>	MindSpore 量化数据类型枚举类, 包含 <i>INT1 ~ INT16</i> , <i>UINT1 ~ UINT16</i> 。
<code>mindspore.common.np_dtype</code>	MindSpore 的 Numpy 数据类型。

mindspore.dtype

class mindspore.dtype

MindSpore 的数据类型。

`dtype` 的实际路径为 `/mindspore/common/dtype.py`。运行以下命令导入环境：

```
from mindspore import dtype as mstype
```

基础数据类型

MindSpore 支持以下基础数据类型：

定义	描述
<code>mindspore.int8</code> , <code>mindspore.byte</code>	8 位整型数
<code>mindspore.int16</code> , <code>mindspore.short</code>	16 位整型数
<code>mindspore.int32</code> , <code>mindspore.intc</code>	32 位整型数
<code>mindspore.int64</code> , <code>mindspore.intp</code>	64 位整型数
<code>mindspore.uint8</code> , <code>mindspore.ubyte</code>	无符号 8 位整型数
<code>mindspore.uint16</code> , <code>mindspore.ushort</code>	无符号 16 位整型数
<code>mindspore.uint32</code> , <code>mindspore.uintc</code>	无符号 32 位整型数
<code>mindspore.uint64</code> , <code>mindspore.uintp</code>	无符号 64 位整型数
<code>mindspore.float16</code> , <code>mindspore.half</code>	16 位浮点数
<code>mindspore.float32</code> , <code>mindspore.single</code>	32 位浮点数
<code>mindspore.float64</code> , <code>mindspore.double</code>	64 位浮点数
<code>mindspore.bfloat16</code>	16 位脑浮点数
<code>mindspore.complex64</code>	64 位复数
<code>mindspore.complex128</code>	128 位复数

其他类型

除基础数据类型以外的其他数据类型，请参照以下表格。

类型	描述
Tensor	MindSpore 中的张量类型。数据格式采用 NCHW。详情请参考 tensor 。
bool_	布尔型，值为 True 或者 False。
int_	整数标量。
uint	无符号整数标量。
float_	浮点标量。
complex	复数标量。
number	数值型，包括 int_、uint、float_、complex 和 bool_。
list_	由 tensor 构造的列表，例如 List[T0,T1,...,Tn]，其中元素 Ti 可以是不同的类型。
tuple_	由 tensor 构造的元组，例如 Tuple[T0,T1,...,Tn]，其中元素 Ti 可以是不同的类型。
function	函数类型。两种返回方式：当 function 不是 None 时，直接返回 function；当 function 为 None 时，返回 function(参数：List[T0,T1,...,Tn]，返回值：T)。
type_type	类型的类型定义。
type_none	没有匹配的返回类型，对应 Python 中的 type(None)。
symbolic_key	在 env_type 中用作变量的键的变量的值。
env_type	用于存储函数的自由变量的梯度，其中键是自由变量节点的 symbolic_key，值是梯度。

类型转换规则

当算子的部分输入要求目标类型一致时，会根据类型转换规则，自动进行类型提升。如果这些输入存在不同大小和类别 (其中 complex > float > int > bool) 的类型，它们将会被提升为具有足够大小和类别的类型。

Tensor 与 Tensor 的类型转换规则请参考以下表格。表格里首行和首列均表示输入 Tensor 的类型，表格中对应位置表示输出 Tensor 的类型，- 表示不进行类型提升。

为方便描述，表格中使用 bool_ 指代 mindspore.bool_，使用 int8 指代 mindspore.int8，以此类推。

Tensor 与 Tensor	bool	int8	int16	int32	int64	uint8	uint16	uint32	uint64	float16	bfloat16	float32	float64	complex64	complex128
bool	bool	int8	int16	int32	int64	uint8	uint16	uint32	uint64	float16	bfloat16	float32	float64	complex64	complex128
int8	int8	int8	int16	int32	int64	int16	-	-	-	float16	bfloat16	float32	float64	complex64	complex128
int16	int16	int16	int16	int32	int64	int16	-	-	-	float16	bfloat16	float32	float64	complex64	complex128
int32	int32	int32	int32	int32	int64	int32	-	-	-	float16	bfloat16	float32	float64	complex64	complex128
int64	int64	int64	int64	int64	int64	int64	-	-	-	float16	bfloat16	float32	float64	complex64	complex128
uint8	uint8	int16	int16	int32	int64	uint8	-	-	-	float16	bfloat16	float32	float64	complex64	complex128
uint16	uint16	-	-	-	-	uint16	-	-	-	-	-	-	-	-	-
uint32	uint32	-	-	-	-	-	-	uint32	-	-	-	-	-	-	-
uint64	uint64	-	-	-	-	-	-	-	uint64	-	-	-	-	-	-
float16	float16	float16	float16	float16	float16	float16	-	-	-	float16	bfloat16	float32	float64	complex64	complex128
bfloat16	bfloat16	float16	float16	float16	float16	float16	-	-	-	float16	bfloat16	float32	float64	complex64	complex128
float32	float32	float32	float32	float32	float32	float32	-	-	-	float32	float32	float32	float64	complex64	complex128
float64	float64	float64	float64	float64	float64	float64	-	-	-	float64	float64	float64	float64	complex128	complex128
complex64	complex64	complex64	complex64	complex64	complex64	complex64	-	-	-	complex64	complex64	complex64	complex64	complex128	complex128
complex128	complex128	complex128	complex128	complex128	complex128	complex128	-	-	-	complex128	complex128	complex128	complex128	complex128	complex128

Number 与 Tensor 的类型转换规则请参考以下表格。表格里首行表示输入 Number 的类型，首列表示输入 Tensor 的类型。表格中对应位置表示输出 Tensor 的类型，- 表示不进行类型提升。

Number 与 Tensor	bool	int	float
bool	bool	int64	float32
int8	int8	int8	float32
int16	int16	int16	float32
int32	int32	int32	float32
int64	int64	int64	float32
uint8	uint8	uint8	float32
uint16	uint16	-	-
uint32	uint32	-	-
uint64	uint64	-	-
float16	float16	float16	float16
bfloat16	bfloat16	bfloat16	bfloat16
float32	float32	float32	float32
float64	float64	float64	float64
complex64	complex64	complex64	complex64
complex128	complex128	complex128	complex128

mindspore.dtype_to_nptype

`mindspore.dtype_to_nptype(type_)`

将 MindSpore 数据类型转换成 NumPy 数据类型。

参数:

- **type_** (mindspore.dtype) - MindSpore 中的 dtype。

返回:

NumPy 的数据类型。

样例:

```
>>> import mindspore as ms
>>> ms.dtype_to_nptype(ms.int8)
<class 'numpy.int8'>
```

mindspore.dtype_to_pytype

`mindspore.dtype_to_pytype(type_)`

将 MindSpore 数据类型转换为 Python 数据类型。

参数:

- **type_** (mindspore.dtype) - MindSpore 中的 dtype。

返回:

Python 的数据类型。

样例:

```
>>> import mindspore as ms
>>> out = ms.dtype_to_pytype(ms.bool_)
>>> print(out)
<class 'bool'>
```

mindspore.pytype_to_dtype

`mindspore.pytype_to_dtype(obj)`

将 Python 数据类型转换为 MindSpore 数据类型。

参数:

- **obj** (type) - Python 数据对象。

返回:

MindSpore 的数据类型。

异常:

- **NotImplementedError** - Python 类型无法转换为 MindSpore 类型。

样例:

```
>>> import mindspore as ms
>>> out = ms.pytype_to_dtype(bool)
>>> print(out)
Bool
```

`mindspore.get_py_obj_dtype`

`mindspore.get_py_obj_dtype(obj)`

获取与 Python 数据类型对应的 MindSpore 数据类型。

参数：

- `obj (type)` - Python 数据对象，或在 Python 环境中定义的变量。

返回：

MindSpore 的数据类型。

样例：

```
>>> import mindspore as ms
>>> ms.get_py_obj_dtype(1)
mindspore.int64
```

`mindspore.QuantDtype`

class `mindspore.QuantDtype`

MindSpore 量化数据类型枚举类，包含 `INT1 ~ INT16`，`UINT1 ~ UINT16`。

`QuantDtype` 定义在 `dtype.py` 文件下。运行以下命令导入环境：

```
from mindspore import QuantDtype
```

教程样例：

- 昇思金箍棒量化感知训练时配置算法

value()

获取当前 `QuantDtype` 的值。该接口当前主要用于序列化或反序列化 `QuantDtype`。

返回：

`int`，表示当前 `QuantDtype` 的值。

样例：

```
>>> from mindspore import QuantDtype
>>> print(QuantDtype.INT8.value())
7
>>> print(QuantDtype.UINT16.value())
115
```

mindspore.common.np_dtype

class mindspore.common.np_dtype

MindSpore 的 Numpy 数据类型。

np_dtype 的实际路径为 */mindspore/common/np_dtype.py*。运行以下命令导入环境：

```
from mindspore.common import np_dtype
```

- 数值型

定义	描述
bfloat16	Numpy 下的 bfloat16 数据类型。该类型仅用于构造 bfloat16 类型的 Tensor，不保证 Numpy 下的完整运算能力。仅当运行时的 Numpy 版本不小于编译时的 Numpy 版本时生效。

1.2 运行环境

<i>mindspore.set_device</i>	设置运行环境的设备类型和设备 ID。
<i>mindspore.get_current_device</i>	获取当前运行环境中的设备类型以及设备 ID。
<i>mindspore.set_deterministic</i>	是否开启确定性计算。
<i>mindspore.set_context</i>	设置运行环境的 context，此接口将在后续版本中废弃，参数相关功能将通过新 API 接口提供。
<i>mindspore.get_context</i>	根据输入 key 获取 context 中的属性值。
<i>mindspore.set _auto_parallel_context</i>	配置自动并行，当前 CPU 仅支持数据并行。
<i>mindspore.get _auto_parallel_context</i>	根据 key 获取自动并行的配置。
<i>mindspore.reset _auto_parallel_context</i>	重置自动并行的配置为默认值。
<i>mindspore.ParallelMode</i>	并行模式。
<i>mindspore.set_ps_context</i>	设置参数服务器训练模式的上下文。
<i>mindspore.get_ps_context</i>	根据 key 获取参数服务器训练模式上下文中的属性值。
<i>mindspore.reset_ps_context</i>	将参数服务器训练模式上下文中的属性重置为默认值。
<i>mindspore.set_algo_parameters</i>	设置并行策略搜索算法中的参数。
<i>mindspore.get_algo_parameters</i>	获取算法参数配置属性。
<i>mindspore.reset_algo_parameters</i>	重置算法参数属性。
<i>mindspore.set_offload_context</i>	配置异构训练详细参数，来调整 offload 策略。
<i>mindspore.get_offload_context</i>	获取 offload 配置参数。

1.2.1 mindspore.set_device

mindspore.set_device (*device_target*, *device_id=None*)
设置运行环境的设备类型和设备 ID。

说明：

- 建议在调用接口 *mindspore.communication.init()* 前设置 *device_target* 和 *device_id*。

参数：

- **device_target** (str) - 要运行的目标设备，仅支持 "Ascend"、"GPU" 和 "CPU"。
- **device_id** (int, 可选) - 目标设备的 ID，值必须在 [0, device_num_per_host-1] 范围中，其中 device_num_per_host 指主机上的设备总数。默认值：None。框架将根据场景按需设置不同的默认值：如果是单卡场景，则设置为 0；如果是 msrun 启动的分布式场景，则会自动协商可用的 device_id 值；如果是其他启动方式的分布式场景，则设置为 0。

样例：

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
```

1.2.2 mindspore.get_current_device

mindspore.get_current_device()

获取当前运行环境中的设备类型以及设备 ID。

样例：

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
>>> ms.get_current_device()
('Ascend', 1)
>>> ms.get_current_device().device_target
'Ascend'
>>> ms.get_current_device().device_id
1
```

1.2.3 mindspore.set_deterministic

mindspore.set_deterministic(deterministic)

是否开启确定性计算。

当开启确定性计算功能时，算子在相同的硬件和输入下，多次执行将产生相同的输出。但启用确定性计算往往导致算子执行变慢。

在分布式场景下，建议用户在调用接口 `mindspore.communication.init()` 前设置确定性计算，以保证使能全局通信域上的通信算子确定性。

框架默认不开启确定性计算。

参数：

- **deterministic** (bool) - 是否开启确定性计算。

样例：

```
>>> import mindspore as ms
>>> ms.set_deterministic(True)
```

1.2.4 mindspore.set_context

`mindspore.set_context(**kwargs)`

设置运行环境的 context，此接口将在后续版本中废弃，参数相关功能将通过新 API 接口提供。

参数：

- **mode** (int) - GRAPH_MODE（用 0 表示）或 PYNATIVE_MODE（用 1 表示）。默认 PYNATIVE_MODE。
- **device_id** (int) - 目标设备的 ID，默认 0。此参数将在后续版本中废弃，请使用接口 `mindspore.set_device()` 代替。
- **device_target** (str) - 程序运行的目标设备，支持 "Ascend"、"GPU" 和 "CPU"。此参数将在后续版本中废弃，请使用接口 `mindspore.set_device()` 代替。
- **deterministic** (str) - 算子确定性计算，默认 "OFF"。此参数将在后续版本中废弃，请使用接口 `mindspore.set_deterministic()` 代替。
- **max_call_depth** (int) - 函数调用最大深度，默认 1000。此参数将在后续版本中废弃，请使用接口 `mindspore.set_recursion_limit()` 代替。
- **variable_memory_max_size** (str) - 此参数将在后续版本中废弃，请使用接口 `mindspore.runtime.set_memory()` 代替。
- **mempool_block_size** (str) - 设置设备内存池的块大小，默认 "1GB"。此参数将在后续版本中废弃，请使用接口 `mindspore.runtime.set_memory()` 代替。
- **memory_optimize_level** (str) - 内存优化级别，默认 "O0"。此参数将在后续版本中废弃，请使用接口 `mindspore.runtime.set_memory()` 代替。
- **max_device_memory** (str) - 设置设备可用的最大内存，默认 "1024GB"。此参数将在后续版本中废弃，请使用接口 `mindspore.runtime.set_memory()` 代替。
- **pynative_synchronize** (bool) - 是否启动设备同步执行，默认 False。此参数将在后续版本中废弃，请使用接口 `mindspore.runtime.launch_blocking()` 代替。
- **compile_cache_path** (str) - 保存编译缓存的路径，默认 "."。此参数将在后续版本中废弃，请使用环境变量 `MS_COMPILER_CACHE_PATH` 代替。
- **inter_op_parallel_num** (int) - 算子间并行数控制，默认 0。此参数将在后续版本中废弃。请使用接口 `mindspore.runtime.dispatch_threads_num()` 代替。
- **memory_offload** (str) - 是否开启 Offload 功能，默认 "OFF"。此参数将在后续版本中废弃，请使用接口 `mindspore.nn.Cell.offload()` 代替。
- **disable_format_transform** (bool) - 是否取消 NCHW 到 NHWC 的自动格式转换功能，默认 False。此参数将在后续版本中废弃，请使用接口 `mindspore.jit()` 相关参数代替。
- **jit_syntax_level** (int) - 设置 jit 语法支持级别，默认 LAX。此参数将在后续版本中废弃，请使用接口 `mindspore.jit()` 相关参数代替。
- **jit_config** (dict) - 设置全局编译选项。此参数将在后续版本中废弃，请使用接口 `mindspore.jit()` 相关参数代替。
- **exec_order** (str) - 算子执行时的排序方法，此参数将在后续版本中废弃，请使用接口 `mindspore.jit()` 相关参数代替。
- **op_timeout** (int) - 设置一个算子的最大执行时间。默认 900。此参数将在后续版本中废弃，请使用接口 `mindspore.device_context.ascend.op_debug.execute_timeout()` 代替。
- **aoe_tune_mode** (str) - AOE 调优。此参数将在后续版本中废弃，请使用接口 `mindspore.device_context.ascend.op_tuning.aoe_tune_mode()` 代替。

- **aoe_config** (dict) - aoe 专用参数。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.ascend.op_tuning.aoe_job_type()` 代替。
- **runtime_num_threads** (int) - 运行时 actor 和 CPU 算子核使用的线程池线程数, 默认 30。此参数将在后续版本中废弃。请使用接口 `mindspore.device_context.cpu.op_tuning.threads_num()` 代替。
- **save_graphs** (bool 或 int) - 表示是否保存中间编译图。默认 0。此参数将在后续版本中废弃, 请使用环境变量 `MS_DEV_SAVE_GRAPHS` 代替。
- **save_graphs_path** (str) - 表示保存计算图的路径。默认 "."。此参数将在后续版本中废弃, 请使用环境变量 `MS_DEV_SAVE_GRAPHS_PATH` 代替。
- **precompile_only** (bool) - 是否仅预编译网络, 默认 False。此参数将在后续版本中废弃, 请使用环境变量 `MS_DEV_PRECOMPILE_ONLY` 代替。
- **enable_compile_cache** (bool) - 是否加载或者保存图编译缓存, 默认 False。此参数将在后续版本中废弃, 请使用环境变量 `MS_COMPILER_CACHE_ENABLE` 代替。
- **ascend_config** (dict) - 设置 Ascend 硬件平台参数。
 - **precision_mode** (str): 混合精度模式设置。此参数将在后续版本中废弃, 默认 "force_fp16"。请使用接口 `mindspore.device_context.ascend.op_precision.precision_mode()` 代替。
 - **jit_compile** (bool): 是否在线编译。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.ascend.op_tuning.op_compile()` 代替。
 - **matmul_allow_hf32** (bool): 是否为 Matmul 类算子使能 FP32 转换为 HF32, 默认 False。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.ascend.op_precision.matmul_allow_hf32()` 代替。
 - **conv_allow_hf32** (bool): 是否为 Conv 类算子使能 FP32 转换为 HF32, 默认 True。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.ascend.op_precision.conv_allow_hf32()` 代替。
 - **op_precision_mode** (str): 算子精度模式配置文件路径。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.ascend.op_precision.op_precision_mode()` 代替。
 - **op_debug_option** (str): Ascend 算子调试配置。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.ascend.op_debug.debug_option()` 代替。
 - **ge_options** (dict): 设置 CANN 的 options 配置项。此参数将在后续版本中废弃, 请使用接口 `mindspore.jit()` 代替。
 - **atomic_clean_policy** (int): 清理网络中 atomic 算子占用的内存的策略。默认 1, 不集中清理。0 表示集中清理。此参数将在后续版本中废弃, 请使用接口 `mindspore.jit()` 相关参数代替。
 - **exception_dump** (str): 开启 Ascend 算子异常 dump。默认 "2"。此参数已废弃, 请使用接口 `mindspore.device_context.ascend.op_debug.aclinit_config()` 代替。
 - **host_scheduling_max_threshold** (int): 控制根图是否使用动态 shape 调度的最大阈值, 默认 0。此参数将在后续版本中废弃, 请使用接口 `mindspore.jit()` 相关参数代替。
 - **parallel_speed_up_json_path** (Union[str, None]): 并行加速配置文件, 此参数将在后续版本中废弃, 请使用接口 `mindspore.parallel.auto_parallel.AutoParallel.transformer_opt()` 代替。
 - **hccl_watchdog** (bool): 开启一个线程监控集合通信故障。默认 True。
- **gpu_config** (dict) - 设置 GPU 硬件平台专用参数, 默认不设置。
 - **conv_fprop_algo** (str): 指定 cuDNN 的卷积前向算法, 默认 "normal"。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.gpu.op_tuning.conv_fprop_algo()` 代替。

- **conv_dgrad_algo** (str): 指定 cuDNN 的卷积输入数据的反向算法, 默认 "normal"。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.gpu.op_tuning.conv_dgrad_algo()` 代替。
- **conv_wgrad_algo** (str): 指定 cuDNN 的卷积输入卷积核的反向算法, 默认 "normal"。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.gpu.op_tuning.conv_wgrad_algo()` 代替。
- **conv_allow_tf32** (bool): 是否开启卷积在 cuDNN 下的 TF32 张量核计算, 默认 True。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.gpu.op_precision.conv_allow_tf32()` 代替。
- **matmul_allow_tf32** (bool): 是否开启矩阵乘在 CUBLAS 下的 TF32 张量核计算, 默认 False。此参数将在后续版本中废弃, 请使用接口 `mindspore.device_context.gpu.op_precision.matmul_allow_tf32()` 代替。
- **print_file_path** (str) - 此参数将在后续版本中废弃。
- **env_config_path** (str) - 此参数将在后续版本中废弃。
- **debug_level** (int) - 此参数将在后续版本中废弃。
- **reserve_class_name_in_scope** (bool) - 此参数将在后续版本中废弃。
- **check_bprop** (bool) - 此参数将在后续版本中废弃。
- **enable_reduce_precision** (bool) - 此参数将在后续版本中废弃。
- **grad_for_scalar** (bool) - 此参数将在后续版本中废弃。
- **support_binary** (bool) - 是否支持在图模式下运行 .pyc 或 .so。

样例:

```
>>> import mindspore as ms
>>> ms.set_context(mode=ms.PYNATIVE_MODE)
>>> ms.set_context(precompile_only=True)
>>> ms.set_context(device_target="Ascend")
>>> ms.set_context(device_id=0)
>>> ms.set_context(save_graphs=True, save_graphs_path="./model.ms")
>>> ms.set_context(enable_reduce_precision=True)
>>> ms.set_context(reserve_class_name_in_scope=True)
>>> ms.set_context(variable_memory_max_size="6GB")
>>> ms.set_context(aoe_tune_mode="online")
>>> ms.set_context(aoe_config={"job_type": "2"})
>>> ms.set_context(check_bprop=True)
>>> ms.set_context(max_device_memory="3.5GB")
>>> ms.set_context(mempool_block_size="1GB")
>>> ms.set_context(print_file_path="print.pb")
>>> ms.set_context(max_call_depth=80)
>>> ms.set_context(env_config_path="./env_config.json")
>>> ms.set_context(grad_for_scalar=True)
>>> ms.set_context(enable_compile_cache=True, compile_cache_path="./cache.ms")
>>> ms.set_context(pynative_synchronize=True)
>>> ms.set_context(runtime_num_threads=10)
>>> ms.set_context(inter_op_parallel_num=4)
>>> ms.set_context(disable_format_transform=True)
>>> ms.set_context(memory_optimize_level='00')
>>> ms.set_context(memory_offload='ON')
>>> ms.set_context(deterministic='ON')
>>> ms.set_context(ascend_config={"precision_mode": "force_fp16", "jit_compile": True,
...                               "atomic_clean_policy": 1, "op_precision_mode": "./op_precision_config_file
↵"
```

(下页继续)

(续上页)

```

...         "op_debug_option": "oom",
...         "ge_options": {"global": {"ge.opSelectImplmode": "high_precision"},
...                        "session": {"ge.exec.atomicCleanPolicy": "0"}}})
>>> ms.set_context(jit_syntax_level=ms.STRICT)
>>> ms.set_context(debug_level=ms.context.DEBUG)
>>> ms.set_context(gpu_config={"conv_fprop_algo": "performance", "conv_allow_tf32": True,
...                           "matmul_allow_tf32": True})
>>> ms.set_context(jit_config={"jit_level": "00"})
>>> ms.set_context(exec_order="bfs")

```

1.2.5 mindspore.get_context

`mindspore.get_context(attr_key)`

根据输入 key 获取 context 中的属性值。如果该 key 没有设置，则会获取它们这些的默认值。

参数：

- **attr_key** (str) - 属性的 key。

返回：

Object，表示给定属性 key 的值。

异常：

- **ValueError** - 输入 key 不是 context 中的属性。

样例：

```

>>> import mindspore as ms
>>> ms.get_context("device_target")
>>> ms.get_context("device_id")

```

1.2.6 mindspore.set_auto_parallel_context

`mindspore.set_auto_parallel_context(**kwargs)`

配置自动并行，当前 CPU 仅支持数据并行。

说明： 全局并行配置，该接口将在未来版本废弃，请使用接口 `mindspore.parallel.auto_parallel.AutoParallel` 代替。

某些配置适用于特定的并行模式，有关详细信息，请参见下表：

Common	AUTO_PARALLEL
device_num	gradient_fp32_sync
global_rank	loss_repeated_mean
gradients_mean	search_mode
parallel_mode	parameter_broadcast
all_reduce_fusion_config	strategy_ckpt_load_file
enable_parallel_optimizer	strategy_ckpt_save_file
parallel_optimizer_config	dataset_strategy
enable_alltoall	pipeline_stages
pipeline_config	auto_parallel_search_mode
force_fp32_communication	pipeline_result_broadcast
	comm_fusion
	strategy_ckpt_config
	group_ckpt_save_file
	auto_pipeline
	dump_local_norm
	dump_local_norm_path
	dump_device_local_norm

参数：

- **device_num** (int) - 表示可用设备的编号，必须在 [1,4096] 范围中。默认值：1。
- **global_rank** (int) - 表示全局 RANK 的 ID，必须在 [0,4095] 范围中。默认值：0。
- **gradients_mean** (bool) - 表示是否在梯度的 AllReduce 后执行平均算子。stand_alone 不支持 gradients_mean。默认值：False。
- **gradient_fp32_sync** (bool) - 在 FP32 中运行梯度的 AllReduce。stand_alone、data_parallel 和 hybrid_parallel 不支持 gradient_fp32_sync。默认值：True。
- **loss_repeated_mean** (bool) - 表示在重复计算时，是否向后执行均值操作符。默认值：True。
- **parallel_mode** (str) - 有五种并行模式，分别是 stand_alone、data_parallel、hybrid_parallel、semi_auto_parallel 和 auto_parallel。默认值：stand_alone。
 - stand_alone：单卡模式。
 - data_parallel：数据并行模式。
 - hybrid_parallel：手动实现数据并行和模型并行。
 - semi_auto_parallel：半自动并行模式。
 - auto_parallel：自动并行模式。
- **search_mode** (str) - 表示有三种策略搜索模式，分别是 recursive_programming，sharding_propagation 和 dynamic_programming (不推荐使用)。只有在 auto_parallel 模式下生效。默认值：recursive_programming。
 - recursive_programming：表示双递归搜索模式。为了获取最优性能，建议用户设置 batch size 大于等于设备数与多副本并行数的乘积。

- `sharding_propagation`: 表示从已配置算子的切分策略传播到所有算子。目前不支持动态 `shape`。
- `dynamic_programming`: 表示动态规划搜索模式。
- **`auto_parallel_search_mode` (str)** - `search_mode` 参数的兼容接口。将在后续的版本中删除。
- **`parameter_broadcast` (bool)** - 表示在训练前是否广播参数。在训练之前, 为了使所有设备的网络初始化参数值相同, 请将设备 0 上的参数广播到其他设备。不同并行模式下的参数广播不同。在 `data_parallel` 模式下, 除 `layerwise_parallel` 属性为 `True` 的参数外, 所有参数都会被广播。在 `hybrid_parallel`、`semi_auto_parallel` 和 `auto_parallel` 模式下, 分段参数不参与广播。默认值: `False`。
- **`strategy_ckpt_load_file` (str)** - 表示用于加载并行策略 `checkpoint` 的路径。目前不建议使用该参数, 建议使用 `strategy_ckpt_config` 来替代它。默认值: `''`。
- **`strategy_ckpt_save_file` (str)** - 表示用于保存并行策略 `checkpoint` 的路径。目前不建议使用该参数, 建议使用 `strategy_ckpt_config` 来替代它。默认值: `''`。
- **`full_batch` (bool)** - 如果在 `auto_parallel` 模式下加载整个 `batch` 数据集, 则此参数应设置为 `True`。默认值: `False`。目前不建议使用该接口, 建议使用 `dataset_strategy` 来替换它。
- **`dataset_strategy` (Union[str, tuple])** - 表示数据集分片策略。默认值: `data_parallel`。
`dataset_strategy="data_parallel"` 等价于 `full_batch=False`, `dataset_strategy="full_batch"` 等价于 `full_batch=True`。
对于在静态图模式下执行并且通过模型并列策略加载到网络的数据集分片策略, 如 `ds_stra ((1, 8)、(1, 8))`, 需要使用 `set_auto_parallel_context(dataset_strategy=ds_stra)`。数据集分片策略不受当前配置的并行模式影响。
`dataset_strategy` 同时也支持配置元组, 元组中每个元素都是 `Layout`。
- **`enable_parallel_optimizer` (bool)** - 这是一个开发中的特性, 它可以为数据并行训练对权重更新计算进行分片, 以节省时间和内存。目前, 自动和半自动并行模式支持 `Ascend` 和 `GPU` 中的所有优化器。数据并行模式仅支持 `Ascend` 中的 `Lamb` 和 `AdamWeightDecay`。默认值: `False`。
- **`enable_alltoall` (bool)** - 允许在通信期间生成 `AllToAll` 通信算子的开关。如果其值为 `False`, 则将由 `AllGather`、`Split` 和 `Concat` 等通信算子的组合来代替 `AllToAll`。默认值: `False`。
- **`force_fp32_communication` (bool)** - 通信期间 `reduce` 类算子 (`AllReduce`、`ReduceScatter`) 是否强制使用 `fp32` 数据类型进行通信的开关。`True` 为开启开关。默认值: `False`。
- **`all_reduce_fusion_config` (list)** - 通过参数索引设置 `AllReduce` 融合策略。仅支持 `ReduceOp.SUM` 和 `HCCL_WORLD_GROUP/NCCL_WORLD_GROUP`。没有默认值。如果不设置, 则关闭算子融合。
- **`pipeline_stages` (int)** - 设置 `pipeline` 并行的阶段信息。表明设备如何单独分布在 `pipeline` 上。所有的设备将被划分为 `pipeline_stages` 个阶段。默认值: `1`。
- **`pipeline_result_broadcast` (bool)** - 表示 `pipeline` 并行推理时, 最后一个 `stage` 的结果是否广播给其余 `stage`。默认值: `False`。
- **`auto_pipeline` (bool)** - 自动设置流水线阶段数。其值将在 `1` 和输入的 `pipeline_stages` 之间选择。本功能需要将 `parallel_mode` 设置成自动并行 `auto_parallel` 并将 `search_mode` 设置成双递归算法 `recursive_programming`。默认值: `False`。
- **`pipeline_config` (dict)** - 用于设置开启 `pipeline` 并行后的行为配置。目前支持如下关键字:
 - `pipeline_interleave`(bool): 表示是否开启 `interleave`。
 - `pipeline_scheduler`(str): 表示 `pipeline` 并行使用的调度策略。当前仅支持 `gpipe/1f1b/seqpipe/seqvpp/seqsmartvpp`。当应用 `seqsmartvpp` 时, 流水线并行必须是偶数。
- **`parallel_optimizer_config` (dict)** - 用于开启优化器并行后的行为配置。仅在 `enable_parallel_optimizer=True` 的时候生效。目前支持如下关键字:
 - `gradient_accumulation_shard`(bool): 请使用 `optimizer_level: level2` 替换此配置。设置累加梯度变量是否在数据并行维度上进行切分。开启后, 将进一步减小模型的显存占用, 但是会在反向计算梯度时引入额外的通信算子 (`ReduceScatter`)。此配置仅在流水线并行训练和梯度累加模式下生效。默认值: `True`。

- `parallel_optimizer_threshold(int)`: 设置参数切分的阈值。占用内存小于该阈值的参数不做切分。占用内存大小 = $\text{shape}[0] * \dots * \text{shape}[n] * \text{size}(\text{dtype})$ 。该阈值非负。单位: KB。默认值: 64。
- `optimizer_weight_shard_size(int)`: 设置指定优化器权重切分通信域的大小。只有当启用优化器并行时生效。数值范围可以是 $(0, \text{device\_num}]$, 若同时开启流水线并行, 数值范围则为 $(0, \text{device\_num}/\text{stage}]$ 。如果参数的数据并行通信域大小不能被 `optimizer_weight_shard_size` 整除, 那么指定的优化器权重切分通信域大小就不会生效。默认值为 -1, 表示优化器权重切片通信域大小是每个参数的数据并行通信域大小。
- `optimizer_level(str, optional)`: `optimizer_level` 配置用于指定优化器切分的切分级别。需要注意的是, 静态图的优化器并行实现与动态图比如 `megatron` 不一致, 但是显存优化效果相同。当 `optimizer_level=level1` 时, 对权重与优化器状态进行切分。当 `optimizer_level=level2` 时, 对权重、优化器状态以及梯度进行切分。当 `optimizer_level=level3` 时, 对权重、优化器状态、梯度进行切分, 并且在反向开始前会对权重额外展开一次 `allgather` 通信, 以释放前向 `allgather` 的显存。它必须是 `[level1、level2、level3]` 中的一个。默认值: `level1`。
- **comm_fusion (dict)** - 用于设置通信算子的融合配置。可以同一类型的通信算子按梯度张量的大小或者顺序分块传输。输入格式为 {"通信类型": {"mode":str, "config": None int 或者 list}}, 每种通信算子的融合配置有两个键: "mode" 和 "config"。支持以下通信类型的融合类型和配置:
 - `openstate`: 是否开启通信融合功能。通过 `True` 或 `False` 来开启或关闭通信融合功能。默认值: `True`。
 - `allreduce`: 进行 `AllReduce` 算子的通信融合。"mode" 包含 "auto"、"size" 和 "index"。在 "auto" 模式下, 融合梯度变量的大小, 默认值阈值为 "64"MB, "config" 对应的值为 `None`。在 "size" 模式下, 需要用户在 `config` 的字典中指定梯度大小阈值, 这个值必须大于 "0"MB。在 "mode" 为 "index" 时, 它与 "all_reduce_fusion_config" 相同, 用户需要给 "config" 传入一个列表, 里面每个值表示梯度的索引。
 - `allgather`: 进行 `AllGather` 算子的通信融合。"mode" 包含 "auto"、"size"。"auto" 和 "size" 模式的配置方式与 `AllReduce` 相同。
 - `reducescatter`: 进行 `ReduceScatter` 算子的通信融合。"mode" 包含 "auto"、"size"。"auto" 和 "size" 模式的配置方式与 `AllReduce` 相同。
- **strategy_ckpt_config (dict)** - 用于设置并行策略文件的配置。包含 `strategy_ckpt_load_file` 和 `strategy_ckpt_save_file` 两个参数的功能, 建议使用此参数替代这两个参数。它包含以下配置:
 - `load_file(str)`: 加载并行切分策略的路径。如果文件扩展名为 `.json`, 文件以 `json` 格式加载。否则, 文件以 `ProtoBuf` 格式加载。默认值: ""。
 - `save_file(str)`: 保存并行切分策略的路径。如果文件扩展名为 `.json`, 文件以 `json` 格式保存。否则, 文件以 `ProtoBuf` 格式保存。默认值: ""。
 - `only_trainable_params(bool)`: 仅保存/加载可训练参数的策略信息。默认值: `True`。
- **group_ckpt_save_file (str)** - 在自动/半自动并行场景下, 指定图编译过程中所创建 `group` 的保存路径。
- **dump_local_norm (bool)** - 在自动/半自动并行场景下, 指定是否打印 `local norm` 值。
- **dump_local_norm_path (str)** - 在自动/半自动并行场景下, 指定 `local norm` 值的保存路径。默认值: ""。
- **dump_device_local_norm (bool)** - 在自动/半自动并行场景下, 指定是否打印 `device local norm` 值。

异常:

- **ValueError** - 输入 `key` 不是自动并行上下文中的属性。

样例:

```
>>> import mindspore as ms
>>> ms.set_auto_parallel_context(device_num=8)
>>> ms.set_auto_parallel_context(global_rank=0)
>>> ms.set_auto_parallel_context(gradients_mean=True)
```

(下页继续)

(续上页)

```

>>> ms.set_auto_parallel_context (gradient_fp32_sync=False)
>>> ms.set_auto_parallel_context (parallel_mode="auto_parallel")
>>> ms.set_auto_parallel_context (search_mode="recursive_programming")
>>> ms.set_auto_parallel_context (auto_parallel_search_mode="recursive_programming")
>>> ms.set_auto_parallel_context (parameter_broadcast=False)
>>> ms.set_auto_parallel_context (strategy_ckpt_load_file="./strategy_stage1.ckpt")
>>> ms.set_auto_parallel_context (strategy_ckpt_save_file="./strategy_stage1.ckpt")
>>> ms.set_auto_parallel_context (dataset_strategy=((1, 8), (1, 8)))
>>> ms.set_auto_parallel_context (enable_parallel_optimizer=False)
>>> ms.set_auto_parallel_context (enable_alltoall=False)
>>> ms.set_auto_parallel_context (all_reduce_fusion_config=[8, 160])
>>> ms.set_auto_parallel_context (pipeline_stages=2)
>>> ms.set_auto_parallel_context (pipeline_stages=2, pipeline_result_broadcast=True)
>>> parallel_config = {"gradient_accumulation_shard": True, "parallel_optimizer_threshold": 24,
...                    "optimizer_weight_shard_size": 2, "optimizer_level": "level3"}
>>> ms.set_auto_parallel_context (parallel_optimizer_config=parallel_config, enable_parallel_optimizer=True)
>>> config = {"allreduce": {"mode": "size", "config": 32}, "allgather": {"mode": "size", "config": 32}}
>>> ms.set_auto_parallel_context (comm_fusion=config)
>>> stra_ckpt_dict = {"load_file": "./stra0.ckpt", "save_file": "./stra1.ckpt", "only_trainable_params": False}
>>> ms.set_auto_parallel_context (strategy_ckpt_config=stra_ckpt_dict)

```

1.2.7 mindspore.get_auto_parallel_context

`mindspore.get_auto_parallel_context (attr_key)`

根据 `key` 获取自动并行的配置。

说明：该接口即将废弃。

参数：

- `attr_key` (str) - 配置的关键字。

返回：

根据 `key` 返回配置的值。

异常：

- `ValueError` - 输入的关键字不在自动并行的配置列表中。

样例：

```

>>> import mindspore as ms
>>> parallel_mode = ms.get_auto_parallel_context ("parallel_mode")
>>> dataset_strategy = ms.get_auto_parallel_context ("dataset_strategy")

```

1.2.8 mindspore.reset_auto_parallel_context

`mindspore.reset_auto_parallel_context()`

重置自动并行的配置为默认值。该接口将在未来版本废弃，请使用接口`mindspore.parallel.auto_parallel.AutoParallel`代替。

- `device_num`: 1。
- `global_rank`: 0。
- `gradients_mean`: `False`。
- `gradient_fp32_sync`: `True`。
- `parallel_mode`: `'stand_alone'`。
- `search_mode`: `'recursive_programming'`。
- `auto_parallel_search_mode`: `'recursive_programming'`。
- `parameter_broadcast`: `False`。
- `strategy_ckpt_load_file`: ""。
- `strategy_ckpt_save_file`: ""。
- `full_batch`: `False`。
- `enable_parallel_optimizer`: `False`。
- `force_fp32_communication`: `False`。
- `enable_alltoall`: `False`。
- `pipeline_stages`: 1。
- `pipeline_result_broadcast`: `False`。
- `fusion_threshold`: 64。
- `auto_pipeline`: `False`。
- `dump_local_norm`: `False`。
- `dump_local_norm_path`: ""。
- `dump_device_local_norm`: `False`。

样例：

```
>>> import mindspore as ms
>>> ms.reset_auto_parallel_context()
```

1.2.9 mindspore.ParallelMode

`class mindspore.ParallelMode`

并行模式。

有五种并行模式，分别是 `STAND_ALONE`、`DATA_PARALLEL`、`HYBRID_PARALLEL`、`SEMI_AUTO_PARALLEL` 和 `AUTO_PARALLEL`。默认值：`STAND_ALONE`。

- `STAND_ALONE`：单卡模式。
- `DATA_PARALLEL`：数据并行模式。

- HYBRID_PARALLEL：手动实现数据并行和模型并行。
- SEMI_AUTO_PARALLEL：半自动并行模式。
- AUTO_PARALLEL：自动并行模式。

MODE_LIST：所有支持的并行模式列表。

1.2.10 mindspore.set_ps_context

`mindspore.set_ps_context(**kwargs)`

设置参数服务器训练模式的上下文。

说明：参数服务器训练模式只在图模式下支持。

需要给参数服务器训练模式设置其他的环境变量。这些环境变量如下所示：

- MS_SERVER_NUM：表示参数服务器数量。
- MS_WORKER_NUM：表示工作进程数量。
- MS_SCHED_HOST：表示调度器 IP 地址。
- MS_SCHED_PORT：表示调度器开启的监听端口。
- MS_ROLE：表示进程角色，角色列表如下：
 - MS_SCHED：表示调度器。
 - MS_WORKER：表示工作进程。
 - MS_PSERVER/MS_SERVER：表示参数服务器。

参数：

- **enable_ps** (bool) - 表示是否启用参数服务器训练模式。只有在 `enable_ps` 设置为 `True` 后，环境变量才会生效。默认值：`False`。
- **config_file_path** (str) - 配置文件路径，用于容灾恢复等，目前参数服务器训练模式仅支持 `Server` 容灾。默认值：`''`。
- **scheduler_manage_port** (int) - 调度器 HTTP 端口，对外开放用于接收和处理用户扩容/缩容等请求。默认值：`11202`。
- **enable_ssl** (bool) - 设置是否打开 SSL 认证。默认值：`False`。
- **client_password** (str) - 用于解密客户端证书密钥的密码。默认值：`''`。
- **server_password** (str) - 用于解密服务端证书密钥的密码。默认值：`''`。

异常：

- **ValueError** - 输入 `key` 不是参数服务器训练模式上下文中的属性。

样例：

```
>>> import mindspore as ms
>>> ms.set_ps_context(enable_ps=True, enable_ssl=True, client_password='123456', server_
    ↳password='123456')
```

1.2.11 mindspore.get_ps_context

`mindspore.get_ps_context(attr_key)`

根据 `key` 获取参数服务器训练模式上下文中的属性值。

参数:

- `attr_key` (str) - 属性的 `key`。
 - `enable_ps` (bool, 可选): 表示是否启用参数服务器训练模式。默认值: `False`。
 - `config_file_path` (str, 可选): 配置文件路径, 用于容灾恢复等。默认值: `''`。
 - `scheduler_manage_port` (int, 可选): 调度器 HTTP 端口, 对外开放用于接收和处理用户扩容/缩容等请求。默认值: `11202`。
 - `enable_ssl` (bool, 可选): 设置是否打开 SSL 认证。默认值: `False`。
 - `client_password` (str, 可选): 用于解密客户端证书密钥的密码。默认值: `''`。
 - `server_password` (str, 可选): 用于解密服务端证书密钥的密码。默认值: `''`。

返回:

根据 `key` 返回属性值。

异常:

- **ValueError** - 输入 `key` 不是参数服务器训练模式上下文中的属性。

样例:

```
>>> import mindspore as ms
>>> ms.get_ps_context("enable_ps")
```

1.2.12 mindspore.reset_ps_context

`mindspore.reset_ps_context()`

将参数服务器训练模式上下文中的属性重置为默认值。各字段的含义及其默认值见 `mindspore.set_ps_context()` 接口。

样例:

```
>>> import mindspore as ms
>>> ms.reset_ps_context()
```

1.2.13 mindspore.set_algo_parameters

`mindspore.set_algo_parameters(**kwargs)`

设置并行策略搜索算法中的参数。

说明: 属性名称为必填项。此接口仅在 `AUTO_PARALLEL` 模式下工作。

参数:

- **fully_use_devices** (bool) - 表示是否仅搜索完全使用所有可用设备的策略。默认值: `False`。例如, 如果有 8 个可用设备, 当该参数设为 `True` 时, 策略 (4, 1) 将不包括在 ReLU 的候选策略中, 因为策略 (4, 1) 仅使用 4 个设备。
- **elementwise_op_strategy_follow** (bool) - 表示 `elementwise` 算子是否具有与后续算子一样的策略, `elementwise` 算子是指对输入张量逐元素应用一个函数变换的算子, 如 `Add`、`ReLU` 等。默认值: `False`。例如, `Add` 的输出给了 `ReLU`, 如果该参数设置为 `True`, 则算法搜索的策略可以保证这两个算子的策略是一致的, 例如, `ReLU` 的策略 (8, 1) 和 `Add` 的策略 ((8, 1), (8, 1))。
- **enable_algo_approxi** (bool) - 表示是否在算法中启用近似。默认值: `False`。由于大型 DNN 模型的并行搜索策略有较大的解空间, 该算法在这种情况下耗时较长。为了缓解这种情况, 如果该参数设置为 `True`, 则会进行近似丢弃一些候选策略, 以便缩小解空间。
- **algo_approxi_epsilon** (float) - 表示近似算法中使用的 `epsilon` 值。默认值: 0.1 此值描述了近似程度。例如, 一个算子的候选策略数量为 `S`, 如果 `enable_algo_approxi` 为 `True`, 则剩余策略的大小为 `min{S, 1/epsilon}`。
- **tensor_slice_align_enable** (bool) - 表示是否检查 `MatMul` 的 `tensor` 切片的 `shape`。默认值: `False` 受某些硬件的属性限制, 只有 `shape` 较大的 `MatMul` 内核才能显示出优势。如果该参数为 `True`, 则检查 `MatMul` 的切片 `shape` 以阻断不规则的 `shape`。
- **tensor_slice_align_size** (int) - 表示 `MatMul` 的最小 `tensor` 切片的 `shape`, 该值必须在 [1,1024] 范围内。默认值: 16。如果 `tensor_slice_align_enable` 设为 `True`, 则 `MatMul tensor` 的最后维度的切片大小应该是该值的倍数。

异常:

- **ValueError** - 无法识别传入的关键字。

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.dataset as ds
>>> from mindspore import nn, ops, train
>>> from mindspore.communication import init
>>> from mindspore.common.initializer import initializer
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> ms.set_auto_parallel_context(parallel_mode=ms.ParallelMode.AUTO_PARALLEL,
...                             search_mode="sharding_propagation")
>>> init()
>>> ms.set_algo_parameters(fully_use_devices=True)
>>> ms.set_algo_parameters(elementwise_op_strategy_follow=True)
>>> ms.set_algo_parameters(enable_algo_approxi=True)
>>> ms.set_algo_parameters(algo_approxi_epsilon=0.2)
>>> ms.set_algo_parameters(tensor_slice_align_enable=True)
>>> ms.set_algo_parameters(tensor_slice_align_size=8)
>>>
>>> # Define the network structure.
>>> class Dense(nn.Cell):
...     def __init__(self, in_channels, out_channels):
...         super().__init__()
...         self.weight = ms.Parameter(initializer("normal", [in_channels, out_channels], ms.
float32))
...         self.bias = ms.Parameter(initializer("normal", [out_channels], ms.float32))
```

(下页继续)

(续上页)

```

...     self.matmul = ops.MatMul()
...     self.add = ops.Add()
...
...     def construct(self, x):
...         x = self.matmul(x, self.weight)
...         x = self.add(x, self.bias)
...         return x
>>>
>>> class FFN(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.flatten = ops.Flatten()
...         self.dense1 = Dense(28*28, 64)
...         self.relu = ops.ReLU()
...         self.dense2 = Dense(64, 10)
...
...     def construct(self, x):
...         x = self.flatten(x)
...         x = self.dense1(x)
...         x = self.relu(x)
...         x = self.dense2(x)
...         return x
>>> net = FFN()
>>> net.dense1.matmul.shard(((2, 1), (1, 2)))
>>>
>>> # Create dataset.
>>> step_per_epoch = 16
>>> def get_dataset(*inputs):
...     def generate():
...         for _ in range(step_per_epoch):
...             yield inputs
...     return generate
>>>
>>> input_data = np.random.rand(1, 28, 28).astype(np.float32)
>>> label_data = np.random.rand(1).astype(np.int32)
>>> fake_dataset = get_dataset(input_data, label_data)
>>> dataset = ds.GeneratorDataset(fake_dataset, ["input", "label"])
>>> # Train network.
>>> optimizer = nn.Momentum(net.trainable_params(), 1e-3, 0.1)
>>> loss_fn = nn.CrossEntropyLoss()
>>> loss_cb = train.LossMonitor()
>>> model = ms.Model(network=net, loss_fn=loss_fn, optimizer=optimizer)
>>> model.train(epoch=2, train_dataset=dataset, callbacks=[loss_cb])

```

1.2.14 mindspore.get_algo_parameters

`mindspore.get_algo_parameters(attr_key)`
 获取算法参数配置属性。

说明：属性名称为必填项。此接口仅在 AUTO_PARALLEL 模式下工作。

参数：

- **attr_key** (str) - 属性的 key。key 包括“fully_use_devices”、“elementwise_op_strategy_follow”、“enable_algo_approxi”、“algo_approxi_epsilon”、“tensor_slice_align_enable” 和“tensor_slice_align_size”。对应属性的含义详见 `mindspore.set_algo_parameters()`。

返回：

根据 key 返回属性值。

异常：

- **ValueError** - 无法识别传入的关键字。

样例：

```
>>> import mindspore as ms
>>> ms.get_algo_parameters("fully_use_devices")
False
```

1.2.15 mindspore.reset_algo_parameters

`mindspore.reset_algo_parameters()`

重置算法参数属性。

说明：此接口仅在 AUTO_PARALLEL 模式下工作。

重置后，属性值为：

- fully_use_devices: False
- elementwise_op_strategy_follow: False
- enable_algo_approxi: False
- algo_approxi_epsilon: 0.1
- tensor_slice_align_enable: False
- tensor_slice_align_size: 16

样例：

```
>>> import mindspore as ms
>>> ms.reset_algo_parameters()
```

1.2.16 mindspore.set_offload_context

`mindspore.set_offload_context(offload_config)`

配置异构训练详细参数，来调整 offload 策略。此接口将被弃用，将在后续版本中删除。

说明：offload 配置只有在通过 `mindspore.set_context(memory_offload="ON")` 开启 offload 功能才会被使用，并且 `memory_optimize_level` 必须设置为 O0。在 Ascend 硬件平台上，图编译等级必须为 O0。

参数：

- **offload_config** (dict) - 输入格式为 {"offload_path": "./offload", "offload_cpu_size": "512GB", "hbm_ratio": 0.9}。支持以下参数配置：
 - **offload_path**: offload 到磁盘上的路径, 支持相对路径, 默认值: `"./offload"`。
 - **offload_cpu_size**: 设置可用于 offload 到 host 侧总内存大小, 格式只支持"xxGB"形式字符串。
 - **offload_disk_size**: 设置可用于 offload 到磁盘的大小, 格式只支持"xxGB"形式字符串。
 - **hbm_ratio**: 策略相关参数, 策略在用户设置的最大显存基础上能够使用的比例, 小数, 值范围 (0,1], 默认值: 1.0。
 - **cpu_ratio**: 策略相关参数, 策略在用户设置的最大 host 侧内存基础上能够使用的比例, 小数, 值范围 (0,1], 默认值: 1.0。
 - **enable_pinned_mem**: 是否开启 Pinned Memory, 开启后可加速片上内存-DDR 之间的拷贝, 会影响系统虚拟内存。Bool 类型。默认值: True。
 - **enable_aio**: 是否开启 AIO, 开启后可加速 DDR-NVME 之间的拷贝。Bool 类型。默认值: True。
 - **aio_block_size**: AIO 中 blocksize 值, 格式只支持"xxGB"形式字符串。
 - **aio_queue_size**: AIO 中 depth 值, 刷新磁盘等待队列最大值, 取值需为整数。
 - **offload_param**: 参数初始位置, 进行设置时只支持"disk"或"cpu", 用户不设置默认为 ""。
 - **offload_checkpoint**: 重计算点 offload 位置, 只有开启重计算才有效, 进行设置时只支持"disk"或"cpu", 用户不设置默认为 ""。
 - **auto_offload**: 是否自动进行 offload 策略生成, True 时生成自动策略, False 时配合 offload_param 生成强制 offload 策略, 一般取值为 True, 默认值: True。
 - **host_mem_block_size**: host 侧内存池 block 块的大小, 格式只支持"xxGB"形式字符串。通过调整块大小, 可以减少内存碎片的产生。

异常:

- **ValueError** - 输入 key 不是 offload_config 中的属性。

样例:

```
>>> from mindspore import context
>>> context.set_offload_context(offload_config={"offload_param": "cpu"})
```

1.2.17 mindspore.get_offload_context

`mindspore.get_offload_context()`

获取 offload 配置参数。通过接口 `mindspore.set_offload_context()` 进行配置。如果用户未设置, 则获取到默认配置。此接口将被弃用, 将在后续版本中删除。

返回:

dict, 异构训练 offload 详细配置参数。

样例:

```
>>> from mindspore import context
>>> offload_config = context.get_offload_context()
```

1.3 随机种子

`mindspore.set_seed`

设置全局种子。

`mindspore.get_seed`

获取随机种子。

1.3.1 mindspore.set_seed

`mindspore.set_seed(seed)`

设置全局种子。

说明:

- 全局种子可用于 `numpy.random`, `mindspore.common.Initializer` 以及 `mindspore.nn.probability.distribution`。
- 如果没有设置全局种子, 这些包将会各自使用自己的种子, `numpy.random` 和 `mindspore.common.Initializer` 将会随机选择种子值, `mindspore.nn.probability.distribution` 将会使用零作为种子值。
- `numpy.random.seed()` 设置的种子仅能被 `numpy.random` 使用, 而该 API 设置的种子也可被 `numpy.random` 使用, 因此推荐使用该 API 设置所有的种子。
- 在 `semi_auto_parallel/auto_parallel` 模式下, 使用 `set_seed` 时, 同一节点具有相同形状和相同切分策略的权重将被初始化为相同的结果, 否则, 将被初始化为不同的结果。

参数:

- **seed** (int) - 设置的全局种子。

异常:

- **ValueError** - 种子值非法 (小于 0)。
- **TypeError** - 种子值非整型数。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, set_seed, Parameter, ops
>>> from mindspore.common.initializer import initializer
>>> # Note: (1) Please make sure the code is running in PYNATIVE MODE;
>>> # (2) Because Composite-level ops need parameters to be Tensors, for below examples,
>>> # when using ops.uniform operator, minval and maxval are initialised as:
>>> minval = Tensor(1.0, ms.float32)
>>> maxval = Tensor(2.0, ms.float32)
>>>
>>> # 1. If global seed is not set, numpy.random and initializer will choose a random seed:
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A1
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A2
>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W1
>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W2
>>> # Rerun the program will get different results:
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A3
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A4
```

(下页继续)

(续上页)

```

>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W3
>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W4
>>>
>>> # 2. If global seed is set, numpy.random and initializer will use it:
>>> set_seed(1234)
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A1
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A2
>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W1
>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W2
>>> # Rerun the program will get the same results:
>>> set_seed(1234)
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A1
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A2
>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W1
>>> w1 = Parameter(initializer("uniform", [2, 2], ms.float32), name="w1") # W2
>>>
>>> # 3. If neither global seed nor op seed is set, mindspore.ops.function.random_func and
>>> # mindspore.nn.probability.distribution will choose a random seed:
>>> c1 = ops.uniform((1, 4), minval, maxval) # C1
>>> c2 = ops.uniform((1, 4), minval, maxval) # C2
>>> # Rerun the program will get different results:
>>> c1 = ops.uniform((1, 4), minval, maxval) # C3
>>> c2 = ops.uniform((1, 4), minval, maxval) # C4
>>>
>>> # 4. If global seed is set, but op seed is not set, mindspore.ops.function.random_func
↳and
>>> # mindspore.nn.probability.distribution will calculate a seed according to global seed
↳and
>>> # default op seed. Each call will change the default op seed, thus each call get
↳different
>>> # results.
>>> set_seed(1234)
>>> c1 = ops.uniform((1, 4), minval, maxval) # C1
>>> c2 = ops.uniform((1, 4), minval, maxval) # C2
>>> # Rerun the program will get the same results:
>>> set_seed(1234)
>>> c1 = ops.uniform((1, 4), minval, maxval) # C1
>>> c2 = ops.uniform((1, 4), minval, maxval) # C2
>>>
>>> # 5. If both global seed and op seed are set, mindspore.ops.function.random_func and
>>> # mindspore.nn.probability.distribution will calculate a seed according to global seed
↳and
>>> # op seed counter. Each call will change the op seed counter, thus each call get
↳different
>>> # results.
>>> set_seed(1234)
>>> c1 = ops.uniform((1, 4), minval, maxval, seed=2) # C1
>>> c2 = ops.uniform((1, 4), minval, maxval, seed=2) # C2
>>> # Rerun the program will get the same results:
>>> set_seed(1234)
>>> c1 = ops.uniform((1, 4), minval, maxval, seed=2) # C1
>>> c2 = ops.uniform((1, 4), minval, maxval, seed=2) # C2
>>>
>>> # 6. If op seed is set but global seed is not set, 0 will be used as global seed. Then
>>> # mindspore.ops.function.random_func and mindspore.nn.probability.distribution act as in
>>> # condition 5.

```

(下页继续)

(续上页)

```
>>> c1 = ops.uniform((1, 4), minval, maxval, seed=2) # C1
>>> c2 = ops.uniform((1, 4), minval, maxval, seed=2) # C2
>>> # Rerun the program will get the different results:
>>> c1 = ops.uniform((1, 4), minval, maxval, seed=2) # C1
>>> c2 = ops.uniform((1, 4), minval, maxval, seed=2) # C2
>>>
>>> # 7. Recall set_seed() in the program will reset numpy seed and op seed counter of
>>> # mindspore.ops.function.random_func and mindspore.nn.probability.distribution.
>>> set_seed(1234)
>>> np_1 = np.random.normal(0, 1, [1]).astype(np.float32) # A1
>>> c1 = ops.uniform((1, 4), minval, maxval, seed=2) # C1
>>> set_seed(1234)
>>> np_2 = np.random.normal(0, 1, [1]).astype(np.float32) # still get A1
>>> c2 = ops.uniform((1, 4), minval, maxval, seed=2) # still get C1
```

1.3.2 mindspore.get_seed

`mindspore.get_seed()`
获取随机种子。

返回：
int，随机种子。

样例：

```
>>> import mindspore as ms
>>> ms.set_seed(1234)
>>> seed = ms.get_seed()
>>> print (seed)
1234
```

1.4 随机状态管理

<code>mindspore.get_rng_state</code>	获取默认生成器状态。
<code>mindspore.Generator</code>	管理随机数状态的生成器，为随机函数提供 <code>seed</code> 和 <code>offset</code> 。
<code>mindspore.initial_seed</code>	返回默认生成器的初始种子。
<code>mindspore.manual_seed</code>	设置默认生成器种子。
<code>mindspore.seed</code>	生成可作为默认生成器种子的随机种子。
<code>mindspore.set_rng_state</code>	设置默认生成器状态。

1.4.1 mindspore.get_rng_state

`mindspore.get_rng_state()`

获取默认生成器状态。

返回:

包含默认生成器的 `seed` 和 `offset` 的 `tuple`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import get_rng_state
>>> state = get_rng_state()
```

1.4.2 mindspore.Generator

class `mindspore.Generator`

管理随机数状态的生成器，为随机函数提供 `seed` 和 `offset`。当 `seed` 和 `offset` 固定时，随机函数产生相同的随机序列。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import Generator
>>> generator = Generator()
>>> generator.manual_seed(5)
>>> print(generator.initial_seed())
5
>>> state = generator.get_state()
>>> generator.seed()
>>> generator.set_state(state)
>>> print(generator.initial_seed())
5
```

get_state()

获取生成器状态。

返回:

Tensor，生成器的状态。

initial_seed()

返回生成器的初始种子。

返回:

生成器的初始化种子。

manual_seed(seed)

设置生成器种子。

参数:

- **seed** (int) - 生成器的种子。

返回：

生成器自身。

seed ()

生成可作为生成器种子的随机种子。

返回：

int 类型，随机生成的种子。

set_state (state)

设置生成器状态。

参数：

- **state** (tensor) - 生成器的状态。

1.4.3 mindspore.initial_seed

mindspore.initial_seed ()

返回默认生成器的初始种子。

返回：

默认生成器的初始化种子。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import manual_seed, initial_seed
>>> manual_seed(14)
>>> print(initial_seed())
14
```

1.4.4 mindspore.manual_seed

mindspore.manual_seed (seed)

设置默认生成器种子。

参数：

- **seed** (int) - 默认生成器的种子。

返回：

默认生成器自身。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import manual_seed, initial_seed
>>> manual_seed(13)
>>> print(initial_seed())
13
```

1.4.5 mindspore.seed

`mindspore.seed()`

生成可作为默认生成器种子的随机种子。

返回:

`int` 类型, 随机生成的种子。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import seed
>>> print(seed())
1663920602
```

1.4.6 mindspore.set_rng_state

`mindspore.set_rng_state(state)`

设置默认生成器状态。

参数:

- **state** (Tensor) - 默认生成器的状态。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import set_rng_state, get_rng_state
>>> state = get_rng_state()
>>> set_rng_state(state)
```

1.5 序列化

`mindspore.async_ckpt_thread_status`

获取异步保存 `checkpoint` 文件线程的状态。

`mindspore.check_checkpoint`

检查 `checkpoint` 文件是否合法。

`mindspore.ckpt_to_safetensors`

将 MindSpore 的 `checkpoint` 文件转换为 `safetensors` 格式并保存到 `save_path`。

`mindspore.convert_model`

将 MindIR 模型转化为其他格式的模型文件。

下页继续

表 8 – 续上页

<code>mindspore.export</code>	将 MindSpore 网络模型导出为指定格式的文件。
<code>mindspore.get_ckpt_path_with_strategy</code>	从当前卡的所有存在备份关系的 <code>checkpoint</code> 文件中找到可用的 <code>checkpoint</code> 文件。
<code>mindspore.load</code>	加载 MindIR 文件。
<code>mindspore.load_checkpoint</code>	加载 <code>checkpoint</code> 文件。
<code>mindspore.load_checkpoint_async</code>	异步加载 <code>checkpoint</code> 文件。
<code>mindspore.load_mindir</code>	加载 MindIR 文件。
<code>mindspore.load_param_into_net</code>	将参数加载到网络中，返回网络中没有被加载的参数列表。
<code>mindspore.parse_print</code>	解析由 <code>mindspore.ops.Print</code> 生成的数据文件。
<code>mindspore.safetensors_to_ckpt</code>	将 <code>safetensors</code> 文件转换为 MindSpore 的 <code>checkpoint</code> 格式并保存到 <code>save_path</code> 。
<code>mindspore.save_checkpoint</code>	将网络权重保存到 <code>checkpoint</code> 文件中。
<code>mindspore.save_mindir</code>	保存 MindIR 文件。

1.5.1 mindspore.async_ckpt_thread_status

`mindspore.async_ckpt_thread_status()`
获取异步保存 `checkpoint` 文件线程的状态。

说明：这个接口在 2.5 版本废弃，并且会在未来版本移除。

在执行异步保存 `checkpoint` 时，判断异步线程是否执行完毕。
返回：

Bool: True，异步保存 `checkpoint` 线程正在运行。False，异步保存 `checkpoint` 线程未运行。

样例：

```
>>> import mindspore as ms
>>> ms.async_ckpt_thread_status()
False
```

1.5.2 mindspore.check_checkpoint

`mindspore.check_checkpoint(ckpt_file_name)`
检查 `checkpoint` 文件是否合法。

说明：这个接口在 2.5 版本废弃，并且会在未来版本移除。

参数：
• `ckpt_file_name` (str) - `checkpoint` 的文件名称。

返回：
bool，`checkpoint` 文件是否合法。

样例：

```
>>> import mindspore as ms
>>> ckpt_file_name = "./checkpoint/LeNet5-1_32.ckpt"
>>> check_result = ms.check_checkpoint(ckpt_file_name)
>>> print(check_result)
True
```

1.5.3 mindspore.ckpt_to_safetensors

`mindspore.ckpt_to_safetensors` (*file_path*, *save_path=None*, *name_map=None*, *file_name_regex=None*, *processes_num=1*)

将 MindSpore 的 checkpoint 文件转换为 safetensors 格式并保存到 *save_path*。safetensors 是 Huggingface 推出的一种可靠、易移植的机器学习模型存储格式，用于安全地存储 Tensor，而且存储速度较快（零拷贝）。

说明：多进程设置数量与主机规模有关，不推荐设置太大，否则容易导致卡死。safetensors 格式不支持 enc 校验功能，若 ckpt 为开启 enc 校验保存，执行转换时会报错。safetensors 格式暂不支持 crc 校验功能，若 ckpt 包含 crc 校验信息，转换为 safetensors 过后 crc 校验信息会丢失。

参数：

- **file_path** (str) - 包含 checkpoint 文件的目录路径或单个 checkpoint 文件 (.ckpt) 的路径。
- **save_path** (str, 可选) - 保存 safetensors 文件的目录路径。默认值：None。
- **name_map** (dict, 可选) - 映射原始参数名到新参数名的字典。默认值：None。
- **file_name_regex** (str, 可选) - 用于匹配需要转换的文件的正则表达式。默认值：None。
- **processes_num** (int, 可选) - 控制并行处理的进程数量。默认值：1。

异常：

- **ValueError** - 如果输入路径无效，或 *save_path* 不是目录，或 *file_path* 不以 '.ckpt' 结尾。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> ms.ckpt_to_safetensors("./ckpt_save_path")
>>> ms.ckpt_to_safetensors("./ckpt_save_path/rank0/checkpoint_0.ckpt")
>>> ms.ckpt_to_safetensors(file_path="./ckpt_save_path/rank0/checkpoint_0.ckpt", save_path="./
↪new_path/")
>>> namemap = {"lin.weight": "new_name"}
>>> ms.ckpt_to_safetensors("./ckpt_save_path/rank0/checkpoint_0.ckpt", "./new_path/", ↪
↪namemap)
```

1.5.4 mindspore.convert_model

`mindspore.convert_model(mindir_file, convert_file, file_format)`

将 MindIR 模型转化为其他格式的模型文件。当前版本仅支持转化成 ONNX 模型。

说明：这个接口在 2.5 版本废弃，并且会在未来版本移除。

参数：

- **mindir_file** (str) - MindIR 模型文件名称。
- **convert_file** (str) - 转化后的模型文件名称。
- **file_format** (str) - 需要转化的文件格式，当前版本仅支持"ONNX"。

异常：

- **TypeError** - *mindir_file* 参数不是 str 类型。
- **TypeError** - *convert_file* 参数不是 str 类型。
- **ValueError** - *file_format* 参数的值不是"ONNX"。

样例：

```
>>> import mindspore as ms
>>> ms.convert_model("lenet.mindir", "lenet.onnx", "ONNX")
```

1.5.5 mindspore.export

`mindspore.export(net, *inputs, file_name, file_format, **kwargs)`

将 MindSpore 网络模型导出为指定格式的文件。

说明：

- 当导出文件格式为 AIR、ONNX 时，单个 Tensor 的大小不能超过 2GB。
 - 当 *file_name* 没有后缀时，系统会根据 *file_format* 自动添加后缀。
 - 现已支持将 `mindspore.jit()` 修饰的函数导出成 MINDIR 格式文件。
 - 当导出 `mindspore.jit()` 修饰的函数时，函数内不能包含有类属性参与的计算。
 - AIR 格式已弃用，将被删除。请使用其他格式或使用 MindSpore Lite 进行离线推理。
-

参数：

- **net** (Union[Cell, function]) - MindSpore 网络结构。
- **inputs** (Union[Tensor, Dataset, List, Tuple, Number, Bool]) - 网络的输入，如果网络有多个输入，需要一同传入。当传入的类型为 *Dataset* 时，将会把数据预处理行为同步保存起来。需要手动调整 *batch* 的大小，当前仅支持获取 *Dataset* 的 *image* 列。
- **file_name** (str) - 导出模型的文件名称。
- **file_format** (str) - MindSpore 目前支持导出"AIR"，"ONNX" 和"MINDIR" 格式的模型。
 - **AIR** - Ascend Intermediate Representation。一种 Ascend 模型的中间表示格式。推荐的输出文件后缀是".air"。

- **ONNX** - Open Neural Network eXchange。一种针对机器学习所设计的开放式的文件格式。推荐的输出文件后缀是".onnx"。
- **MINDIR** - MindSpore Native Intermediate Representation for Anf。一种 MindSpore 模型的中间表示格式。推荐的输出文件后缀是".mindir"。MINDIR 格式不支持带有字典属性的算子导出。
- **kwargs** (dict) - 配置选项字典。
 - **enc_key** (byte) - 用于加密的字节类型密钥，有效长度为 16、24 或者 32。
 - **enc_mode** (Union[str, function]) - 指定加密模式，当设置 *enc_key* 时启用。
 - * 对于'AIR'和'ONNX'格式的模型，当前仅支持自定义加密导出。
 - * 对于'MINDIR'格式的模型，支持的加密选项有：'AES-GCM'，'AES-CBC'，'SM4-CBC' 和用户自定义加密算法。默认值：'AES-GCM'。
 - * 关于使用自定义加密导出的详情，请查看 [教程](#)。
 - **dataset** (Dataset) - 指定数据集的预处理方法，用于将数据集的预处理导入 MindIR。
 - **custom_func** (function) - 用户自定义的导出策略的函数。该函数会在网络导出时，对模型使用该函数进行自定义处理。需要注意，当前仅支持对 *format* 为 *MindIR* 的文件使用 *custom_func*，且自定义函数仅接受一个代表 *MindIR* 文件 *Proto* 对象的入参。当使用 *custom_func* 对模型进行修改时，需要保证修改后模型的正确性，否则可能导致模型加载失败或功能错误。默认值：None。

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> input_tensor = Tensor(np.ones([1, 1, 32, 32]).astype(np.float32))
>>> ms.export(net, input_tensor, file_name='lenet', file_format='MINDIR')
>>>
>>> # Export model in MindIR format and modified the model info using custom_func
>>> # The custom_func only support one input representing the Proto object of the model
>>> # And custom_func does not support return value
>>> def _custom_func(mindir_model):
...     mindir_model.producer_name = "test11111"
...     mindir_model.producer_version = "11.0"
...     mindir_model.user_info["version"] = "11.0"
>>> ms.export(net, input_tensor, file_name="lenet", file_format='MINDIR', custom_func=_
↵custom_func)
```

教程样例：

- 保存与加载 - 保存和加载 MindIR

1.5.6 mindspore.get_ckpt_path_with_strategy

`mindspore.get_ckpt_path_with_strategy (cur_ckpt_path, cur_strategy_path)`

从当前卡的所有存在备份关系的 checkpoint 文件中找到可用的 checkpoint 文件。

该接口假定文件名中存在 `rank_{rank_id}` 的字符串用于区分不同的 checkpoint 文件。如果不存在该字符串并且传入路径可用, 则返回传入路径, 否则返回 `None`。

说明:

- 这个 API 必须在集群初始完成后调用, 因为接口内部需要获取集群信息。

参数:

- `cur_ckpt_path` (str) - 当前卡需要的 checkpoint 的文件路径。
- `cur_strategy_path` (str) - 当前卡的 strategy 文件。

返回:

- str, 如果找到可用 checkpoint 文件, 返回该路径。
- None, 如果未找到可用 checkpoint 文件, 返回 None。

样例:

```
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore import get_ckpt_path_with_strategy
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> ms.set_auto_parallel_context(parallel_mode=ms.ParallelMode.DATA_PARALLEL, gradients_
↳mean=True)
>>> init()
>>> ckpt_file= "./rank_5/iteration-1_40.ckpt"
>>> strategy_file = "./src_pipeline_strategys/src_strategy_5.ckpt"
>>> ckpt_file_new = get_ckpt_path_with_strategy(ckpt_file, strategy_file)
>>> print(ckpt_file_new)
```

1.5.7 mindspore.load

`mindspore.load (file_name, **kwargs)`

加载 MindIR 文件。

返回一个可以由 `GraphCell` 执行的对象, 更多细节参见类 `mindspore.nn.GraphCell`。

参数:

- `file_name` (str) - MindIR 文件的全路径名。
- `kwargs` (dict) - 配置项字典。
 - `dec_key` (bytes) - 用于解密的字节类型密钥。有效长度为 16、24 或 32。
 - `dec_mode` (Union[str, function], 可选) - 指定解密模式, 设置 `dec_key` 时生效。
 - * 可选项: 'AES-GCM'、'SM4-CBC'、'AES-CBC' 或自定义解密函数。默认值: 'AES-GCM'。
 - * 关于使用自定义解密加载的详情, 请查看 [教程](#)。

返回:

GraphCell，一个可以由 *GraphCell* 构成的可执行的编译图。

异常：

- **NotImplementedError** - 动态结构混淆已不再支持。
- **ValueError** - MindIR 文件名不存在或 *file_name* 不是 `string` 类型。
- **RuntimeError** - 解析 MindIR 文件失败。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> from mindspore import context
>>> context.set_context(mode=context.GRAPH_MODE)
>>>
>>> net = nn.Conv2d(1, 1, kernel_size=3, weight_init="ones")
>>> input_tensor = Tensor(np.ones([1, 1, 3, 3]).astype(np.float32))
>>> ms.export(net, input_tensor, file_name="net", file_format="MINDIR")
>>> graph = ms.load("net.mindir")
>>> net = nn.GraphCell(graph)
>>> output = net(input_tensor)
>>> print(output)
[[[4. 6. 4.]
  [6. 9. 6.]
  [4. 6. 4.]]]]
```

教程样例：

- [保存与加载 - 保存和加载 MindIR](#)

1.5.8 mindspore.load_checkpoint

`mindspore.load_checkpoint` (*ckpt_file_name*, *net=None*, *strict_load=False*, *filter_prefix=None*, *dec_key=None*, *dec_mode='AES-GCM'*, *specify_prefix=None*, *choice_func=None*, *crc_check=False*, *remove_redundancy=False*, *format='ckpt'*)

加载 checkpoint 文件。

说明：

- *specify_prefix* 和 *filter_prefix* 参数已被弃用，推荐使用 *choice_func* 代替。*specify_prefix* 和 *filter_prefix* 的功能相互之间没有影响，并且使用这两个参数中的任何一个都将覆盖 *choice_func*。
- 如果没有参数被成功加载，将会报 **ValueError**。
- 当加载去冗余的 checkpoint 时，网络应该是编译过的。

参数：

- **ckpt_file_name** (str) - checkpoint 的文件名称。
- **net** (Cell, 可选) - 加载 checkpoint 参数的网络。默认值：None。
- **strict_load** (bool, 可选) - 是否将严格加载参数到网络中。如果是 `False`，它将根据相同的后缀名将参数字典中的参数加载到网络中，并会在精度不匹配时，进行强制精度转换，比如将 `float32` 转换为 `float16`。默认值：`False`。

- **filter_prefix** (Union[str, list[str], tuple[str]], 可选) - 废弃 (请参考参数 *choice_func*)。以 *filter_prefix* 开头的参数将不会被加载。默认值: None。
- **dec_key** (Union[None, bytes], 可选) - 用于解密的字节类型密钥, 如果值为 None, 则不需要解密。默认值: None。
- **dec_mode** (str, 可选) - 该参数仅当 *dec_key* 不为 None 时有效。指定解密模式, 目前支持 "AES-GCM", "AES-CBC" 和 "SM4-CBC"。默认值: "AES-GCM"。
- **specify_prefix** (Union[str, list[str], tuple[str]], 可选) - 废弃 (请参考参数 *choice_func*)。以 *specify_prefix* 开头的参数将会被加载。默认值: None。
- **choice_func** (Union[None, function], 可选) - 函数的输入值为字符串类型的 **Parameter** 名称, 并且返回值是一个布尔值。如果返回 True, 则匹配自定义条件的 **Parameter** 将被加载。如果返回 False, 则匹配自定义条件的 **Parameter** 将被删除。默认值: None。
- **crc_check** (bool, 可选) - 是否在加载 checkpoint 时进行 crc32 校验。默认值: False。
- **remove_redundancy** (bool, 可选) - 是否开启加载去冗余保存的 checkpoint。去冗余是指去除数据并行模式下的冗余数据。默认值: False, 不开启去冗余加载。
- **format** (str, 可选) - 输入文件的格式, 可以是 "ckpt" 或 "safetensors"。默认值: "ckpt"。

返回:

字典, key 是参数名称, value 是 **Parameter** 类型。当使用 *mindspore.save_checkpoint()* 的 *append_dict* 参数和 *mindspore.train.CheckpointConfig* 的 *append_info* 参数保存 checkpoint, *append_dict* 和 *append_info* 是 dict 类型, 且它们的值是 string 时, 加载 checkpoint 得到的返回值是 string 类型, 其他情况返回值均是 **Parameter** 类型。

异常:

- **ValueError** - checkpoint 文件格式不正确。
- **ValueError** - 没有一个参数被成功加载。
- **TypeError** - *specify_prefix* 或者 *filter_prefix* 的数据类型不正确。

样例:

```
>>> import mindspore as ms
>>>
>>> ckpt_file_name = "./checkpoint/LeNet5-1_32.ckpt"
>>> param_dict = ms.load_checkpoint(ckpt_file_name,
...                               choice_func=lambda x: x.startswith("conv") and not x.
->startswith("conv1"))
>>> print(param_dict["conv2.weight"])
Parameter (name=conv2.weight, shape=(16, 6, 5, 5), dtype=Float32, requires_grad=True)
>>> def func(param_name):
...     whether_load = False
...     if param_name.startswith("conv"):
...         whether_load = True
...     if param_name.startswith("conv1"):
...         whether_load = False
...     return whether_load
>>> param_dict1 = ms.load_checkpoint(ckpt_file_name, choice_func=func)
>>> print(param_dict1["conv2.weight"])
Parameter (name=conv2.weight, shape=(16, 6, 5, 5), dtype=Float32, requires_grad=True)
>>> def func(param_name):
...     whether_load = False
...     if param_name.startswith("conv1"):
...         whether_load = True
```

(下页继续)

(续上页)

```

...     return whether_load
>>> param_dict2 = ms.load_checkpoint(ckpt_file_name, choice_func=func)
>>> print(param_dict2)
{'conv1.weight': Parameter (name=conv1.weight, shape=(6, 1, 5, 5), dtype=Float32, requires_
    grad=True)}

```

教程样例：

- 保存与加载 - 保存和加载模型权重

1.5.9 mindspore.load_checkpoint_async

`mindspore.load_checkpoint_async(ckpt_file_name, net=None, strict_load=False, filter_prefix=None, dec_key=None, dec_mode='AES-GCM', specify_prefix=None, choice_func=None)`

异步加载 checkpoint 文件。

警告：这是一个实验性 API，后续可能修改或删除。

说明：

- `specify_prefix` 和 `filter_prefix` 的功能相互之间没有影响。
- 如果发现没有参数被成功加载，将会报 `ValueError`。
- `specify_prefix` 和 `filter_prefix` 参数已被弃用，推荐使用 `choice_func` 代替。并且使用这两个参数中的任何一个都将覆盖 `choice_func`。

参数：

- **ckpt_file_name** (str) - checkpoint 的文件名称。文件名后缀必须是 `ckpt` 或 `safetensors`。
- **net** (Cell, 可选) - 加载 checkpoint 参数的网络。默认值：None。
- **strict_load** (bool, 可选) - 是否将严格加载参数到网络中。如果是 `False`，它将根据相同的后缀名将参数字典中的参数加载到网络中，并会在精度不匹配时，进行强制精度转换，比如将 `float32` 转换为 `float16`。默认值：`False`。
- **filter_prefix** (Union[str, list[str], tuple[str]], 可选) - 废弃（请参考参数 `choice_func`）。以 `filter_prefix` 开头的参数将不会被加载。默认值：None。
- **dec_key** (Union[None, bytes], 可选) - 用于解密的字节类型密钥，如果值为 `None`，则不需要解密。默认值：None。
- **dec_mode** (str, 可选) - 该参数仅当 `dec_key` 不为 `None` 时有效。指定解密模式，目前支持 "AES-GCM"，"AES-CBC" 和 "SM4-CBC"。默认值："AES-GCM"。
- **specify_prefix** (Union[str, list[str], tuple[str]], 可选) - 废弃（请参考参数 `choice_func`）。以 `specify_prefix` 开头的参数将会被加载。默认值：None。
- **choice_func** (Union[None, function], 可选) - 函数的输入值为字符串类型的 `Parameter` 名称，并且返回值是一个布尔值。如果返回 `True`，则匹配自定义条件的 `Parameter` 将被加载。如果返回 `False`，则匹配自定义条件的 `Parameter` 将被删除。默认值：None。

返回：

自定义的内部类，调用其 `result` 方法可以得到 `mindspore.load_checkpoint()` 返回的结果。

异常：

- **ValueError** - checkpoint 文件格式不正确。
- **ValueError** - 没有一个参数被成功加载。
- **TypeError** - `specify_prefix` 或者 `filter_prefix` 的数据类型不正确。

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.train import Model
>>> from mindspore.amp import FixedLossScaleManager
>>> from mindspore import context
>>> from mindspore import load_checkpoint_async
>>> from mindspore import load_param_into_net
>>> mindspore.set_device(device_target="Ascend")
>>> context.set_context(mode=context.GRAPH_MODE)
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> ckpt_file = "./checkpoint/LeNet5-1_32.ckpt"
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
>>> loss_scale_manager = FixedLossScaleManager()
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim, metrics=None,
...               loss_scale_manager=loss_scale_manager)
>>> pd_future = load_checkpoint_async(ckpt_file)
>>> model.build(train_dataset=dataset, epoch=2)
>>> param_dict = pd_future.result()
>>> load_param_into_net(net, param_dict)
>>> model.train(2, dataset)
>>> print("param dict len: ", len(param_dict), flush=True)
```

1.5.10 mindspore.load_mindir

`mindspore.load_mindir(file_name)`

加载 MindIR 文件。

说明：这个接口在 2.5 版本废弃，并且会在未来版本移除。

参数：

- **file_name** (str) - MindIR 文件的全路径名。

返回：

ModelProto，一个 MindIR proto 对象。

异常：

- **ValueError** - 文件不存在或文件名格式不对。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> md = ms.load_mindir("test.mindir")
```

1.5.11 mindspore.load_param_into_net

`mindspore.load_param_into_net(net, parameter_dict, strict_load=False, remove_redundancy=False)`

将参数加载到网络中，返回网络中没有被加载的参数列表。

说明：当加载去冗余的参数字典时，网络应该是编译过的。

参数:

- **net** (Cell) - 将要加载参数的网络。
- **parameter_dict** (dict) - 加载 checkpoint 文件得到的字典。
- **strict_load** (bool, 可选) - 是否将参数严格加载到网络中。如果为 `False`，它将以相同的后缀名将参数字典中的参数加载到网络中，并会在精度不匹配时，进行精度转换，比如将 `float32` 转换为 `float16`。默认值：`False`。
- **remove_redundancy** (bool, 可选) - 是否开启加载去冗余保存的 checkpoint。去冗余是指去除数据并行模式下的冗余数据。默认值：`False`，即不开启去冗余加载。

返回:

- **param_not_load** (List)，网络中没有被加载的参数。
- **ckpt_not_load** (List)，checkpoint 文件中没有被加载的参数。

异常:

- **TypeError** - 如果参数不是 `Cell`，或者 `parameter_dict` 不是 `Parameter` 类型的字典。

样例:

```
>>> import mindspore as ms
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> ckpt_file_name = "./checkpoint/LeNet5-1_32.ckpt"
>>> param_dict = ms.load_checkpoint(ckpt_file_name, filter_prefix="conv1")
>>> param_not_load, _ = ms.load_param_into_net(net, param_dict)
>>> print(param_not_load)
['conv1.weight']
```

教程样例:

- 保存与加载 - 保存和加载模型权重

1.5.12 mindspore.parse_print

`mindspore.parse_print(print_file_name)`
解析由 `mindspore.ops.Print` 生成的数据文件。

说明：这个接口在 2.5 版本废弃，并且会在未来版本移除。

参数：

- **print_file_name** (str) - 需要解析的文件名。

返回：

List, 由 Tensor 组成的 list。

异常：

- **ValueError** - 指定的文件不存在或为空。
- **RuntimeError** - 解析文件失败。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, Tensor, ops
>>> ms.set_context(mode=ms.GRAPH_MODE, print_file_path='log.data')
>>> class PrintInputTensor(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.print = ops.Print()
...
...     def construct(self, input_pra):
...         self.print('print:', input_pra)
...         return input_pra
>>> x = np.array([[1, 2, 3, 4], [5, 6, 7, 8]]).astype(np.float32)
>>> input_pra = Tensor(x)
>>> net = PrintInputTensor()
>>> net(input_pra)
>>>
>>> data = ms.parse_print('./log.data')
>>> print(data)
['print:', Tensor(shape=[2, 4], dtype=Float32, value=
[[ 1.00000000e+00,  2.00000000e+00,  3.00000000e+00,  4.00000000e+00],
[ 5.00000000e+00,  6.00000000e+00,  7.00000000e+00,  8.00000000e+00]])]
```

1.5.13 mindspore.safetensors_to_ckpt

`mindspore.safetensors_to_ckpt(file_path, save_path=None, name_map=None, file_name_regex=None, processes_num=1)`

将 `safetensors` 文件转换为 MindSpore 的 `checkpoint` 格式并保存到 `save_path`。`safetensors` 是 Huggingface 推出的一种可靠、易移植的机器学习模型存储格式，用于安全地存储 `Tensor`，而且存储速度较快（零拷贝）。

说明：多进程设置数量与主机规模有关，不推荐设置太大，否则容易导致卡死。

参数：

- **file_path** (str) - 包含 safetensors 文件的目录路径或单个 safetensors 文件 (.safetensors) 的路径。
- **save_path** (str, 可选) - 保存 checkpoint 文件的目录路径。默认值：None。
- **name_map** (dict, 可选) - 映射原始参数名到新参数名的字典。默认值：None。
- **file_name_regex** (str, 可选) - 用于匹配需要转换的文件的正则表达式。默认值：None。
- **processes_num** (int, 可选) - 控制并行处理的进程数量。默认值：1。

异常：

- **ValueError** - 如果输入路径无效，save_path 不是目录，或 file_path 不以'.safetensors' 结尾。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> ms.safetensors_to_ckpt("./safetensors_save_path")
>>> ms.safetensors_to_ckpt("./safetensors_save_path/rank0/checkpoint_0.safetensors")
>>> ms.safetensors_to_ckpt("./safetensors_save_path/rank0/checkpoint_0.safetensors", "./new_
↳path/")
>>> namemap = {"lin.weight": "new_name"}
>>> ms.safetensors_to_ckpt("./safetensors_save_path/rank0/checkpoint_0.safetensors", "./new_
↳path/", namemap)
```

1.5.14 mindspore.save_checkpoint

`mindspore.save_checkpoint` (*save_obj*, *ckpt_file_name*, *integrated_save=True*, *async_save=False*, *append_dict=None*, *enc_key=None*, *enc_mode='AES-GCM'*, *choice_func=None*, *crc_check=False*, *format='ckpt'*, ***kwargs*)

将网络权重保存到 checkpoint 文件中。

说明： *enc_mode* 和 *crc_check* 参数互斥，不能同时配置。

参数：

- **save_obj** (Union[Cell, list, dict]) - 待保存的对象。数据类型可为 `mindspore.nn.Cell`、list 或 dict。
 - 若为 list，可以是 `Cell.trainable_params()` 的返回值，或元素为 dict 的列表（如 `[{"name": param_name, "data": param_data}, ...]`，*param_name* 的类型必须是 str，*param_data* 的类型必须是 Parameter 或者 Tensor）。
 - 若为 dict，可以是 `mindspore.load_checkpoint()` 的返回值。
- **ckpt_file_name** (str) - checkpoint 文件名称。如果文件已存在，将会覆盖原有文件。
- **integrated_save** (bool) - 在并行场景下是否合并保存拆分的 Tensor。默认值：True。
- **async_save** (Union[bool, str], 可选) - 是否使用异步方式保存 checkpoint 文件或 safetensors 文件，True 时默认使用异步线程；如果是 str 类型，选择异步保存方式，可以是 "process" 或 "thread"。默认值：False。
- **append_dict** (dict) - 需要保存的其他信息。dict 的键必须为 str 类型，dict 的值类型必须是 int、float、bool、string、Parameter 或 Tensor 类型。默认值：None。

- **enc_key** (Union[None, bytes]) - 用于加密的字节类型密钥。如果值为 None，那么不需要加密。默认值: None。
- **enc_mode** (str) - 该参数在 *enc_key* 不为 None 时有效, 指定加密模式, 目前仅支持 "AES-GCM", "AES-CBC" 和 "SM4-CBC"。默认值: "AES-GCM"。
- **choice_func** (function) - 用于自定义控制保存参数的函数。函数的输入值为字符串类型的 *Parameter* 名称, 并且返回值是一个布尔值。默认值: None。
 - 如果返回 True, 则匹配自定义条件的 *Parameter* 将被保存。
 - 如果返回 False, 则未匹配自定义条件的 *Parameter* 不会被保存。
- **crc_check** (bool) - 是否在保存 checkpoint 时进行 crc32 校验, 并把计算结果写到文件中。默认值: False。
- **format** (str) - 输出文件的格式, 可以是 "ckpt" 或 "safetensors"。默认值: "ckpt"。
- **kwargs** (dict) - 配置选项字典。

异常:

- **TypeError** - 如果参数 *save_obj* 类型不为 *mindspore.nn.Cell*、list 或者 dict。
- **TypeError** - 如果参数 *integrated_save* 不是 bool 类型。
- **TypeError** - 如果参数 *ckpt_file_name* 不是字符串类型。
- **TypeError** - 如果参数 *async_save* 不是 bool 类型或者字符串类型。
- **ValueError** - 如果参数 *async_save* 是字符串类型, 但不是 "process" 或者 "thread"。

样例:

```
>>> import mindspore as ms
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> ms.save_checkpoint(net, "./lenet.ckpt",
...                     choice_func=lambda x: x.startswith("conv") and not x.startswith("conv1")
...                     ↪)
>>> param_dict1 = ms.load_checkpoint("./lenet.ckpt")
>>> print(param_dict1)
{'conv2.weight': Parameter (name=conv2.weight, shape=(16, 6, 5, 5), dtype=Float32, requires_
↪grad=True)}
>>> params_list = net.trainable_params()
>>> ms.save_checkpoint(params_list, "./lenet_list.ckpt",
...                     choice_func=lambda x: x.startswith("conv") and not x.startswith("conv2")
...                     ↪)
>>> param_dict2 = ms.load_checkpoint("./lenet_list.ckpt")
>>> print(param_dict2)
{'conv1.weight': Parameter (name=conv1.weight, shape=(6, 1, 5, 5), dtype=Float32, requires_
↪grad=True)}
>>> ms.save_checkpoint(param_dict2, "./lenet_dict.ckpt")
>>> param_dict3 = ms.load_checkpoint("./lenet_dict.ckpt")
>>> print(param_dict3)
{'conv1.weight': Parameter (name=conv1.weight, shape=(6, 1, 5, 5), dtype=Float32, requires_
↪grad=True)}
```

教程样例:

- 保存与加载 - 保存和加载模型权重

1.5.15 mindspore.save_mindir

`mindspore.save_mindir(model, file_name)`
保存 MindIR 文件。

说明：这个接口在 2.5 版本废弃，并且会在未来版本移除。

参数：

- **model** (ModelProto) - MindIR model。
- **file_name** (str) - MindIR 文件的全路径名。

异常：

- **TypeError** - 参数 *model* 不是 ModelProto 对象。
- **ValueError** - 文件路径不存在或文件名格式不对。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> md = ms.load_mindir("test.mindir")
>>> md.user_info["version"]="pangu v100"
>>> ms.save_mindir(md, "test_new.mindir")
>>> md_new = ms.load_mindir("test_new.mindir")
>>> md_new.user_info
```

1.6 自动微分

<code>mindspore.grad</code>	生成求导函数，用于计算给定函数的梯度。
<code>mindspore.value_and_grad</code>	生成求导函数，用于计算给定函数的正向计算结果和梯度。
<code>mindspore.get_grad</code>	当 <code>mindspore.grad()</code> 或 <code>mindspore.value_and_grad()</code> 的 <code>return_ids</code> 参数设置为 <code>True</code> 时，将 <code>mindspore.grad</code> 的返回值或 <code>mindspore.value_and_grad</code> 的第二个返回值作为 <code>gradients</code> ，再根据 <code>identifier</code> 在 <code>gradients</code> 中找到对应的梯度值。
<code>mindspore.jacfwd</code>	通过前向模式计算给定网络的 Jacobian 矩阵，对应 前向模式自动微分。
<code>mindspore.jacrev</code>	通过反向模式计算给定网络的 Jacobian 矩阵，对应 反向模式自动微分。
<code>mindspore.jvp</code>	计算给定网络的雅可比向量积 (Jacobian-vector product, JVP)。
<code>mindspore.vjp</code>	计算给定网络的向量雅可比积 (vector-jacobian-product, VJP)。

1.6.1 mindspore.grad

`mindspore.grad(fn, grad_position=0, weights=None, has_aux=False, return_ids=False)`

生成求导函数，用于计算给定函数的梯度。

函数求导包含以下三种场景：

1. 对输入求导，此时 `grad_position` 非 `None`，而 `weights` 是 `None`；
2. 对网络变量求导，此时 `grad_position` 是 `None`，而 `weights` 非 `None`；
3. 同时对输入和网络变量求导，此时 `grad_position` 和 `weights` 都非 `None`。

参数：

- **fn** (`Union[Cell, Function]`) - 待求导的函数或网络。
- **grad_position** (`Union[NoneType, int, tuple[int]]`) - 指定求导输入位置的索引。
 - 若为 `int` 类型，表示对单个输入求导；
 - 若为 `tuple` 类型，表示对 `tuple` 内索引的位置求导，其中索引从 0 开始；
 - 若是 `None`，表示不对输入求导，这种场景下，`weights` 非 `None`。默认值：0。
- **weights** (`Union[ParameterTuple, Parameter, list[Parameter]]`) - 训练网络中需要返回梯度的网络变量。一般可通过 `weights = net.trainable_params()` 获取。默认值：`None`。
- **has_aux** (`bool`) - 是否返回辅助参数的标志。若为 `True`，`fn` 输出数量必须超过一个，其中只有 `fn` 第一个输出参与求导，其他输出值将直接返回。默认值：`False`。
- **return_ids** (`bool`) - 是否返回由梯度和指定求导输入位置的索引或网络变量组成的 `tuple`。若为 `True`，其输出中所有的梯度值将被替换为：由该梯度和其输入的位置索引，或者用于计算该梯度的网络变量组成的 `tuple`。默认值：`False`。

返回：

`Function`，用于计算给定函数的梯度的求导函数。例如 `out1, out2 = fn(*args)`，若 `has_aux` 为 `True`，梯度函数将返回 `(gradient, out2)` 形式的结果，其中 `out2` 不参与求导；若为 `False`，将直接返回 `gradient`。

若 `return_ids` 为 `True`，梯度函数返回的梯度将被替代为由返回的梯度和指定求导输入位置的索引或网络变量组成的 `tuple`。

异常：

- **ValueError** - 入参 `grad_position` 和 `weights` 同时为 `None`。
- **TypeError** - 入参类型不符合要求。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops, nn, grad
>>>
>>> # Cell object to be differentiated
>>> class Net(nn.Cell):
...     def construct(self, x, y, z):
...         return x * y * z
>>> x = Tensor([1, 2], mindspore.float32)
```

(下页继续)

(续上页)

```

>>> y = Tensor([-2, 3], mindspore.float32)
>>> z = Tensor([0, 3], mindspore.float32)
>>> net = Net()
>>> output = grad(net, grad_position=(1, 2))(x, y, z)
>>> print(output)
(Tensor(shape=[2], dtype=Float32, value=[ 0.00000000e+00,  6.00000000e+00]),
 Tensor(shape=[2], dtype=Float32, value=[-2.00000000e+00,  6.00000000e+00]))
>>>
>>> # Function object to be differentiated
>>> def fn(x, y, z):
...     res = x * ops.exp(y) * ops.pow(z, 2)
...     return res, z
>>> x = Tensor([3, 3], mindspore.float32)
>>> y = Tensor([0, 0], mindspore.float32)
>>> z = Tensor([5, 5], mindspore.float32)
>>> gradient, aux = grad(fn, (1, 2), None, True)(x, y, z)
>>> print(gradient)
(Tensor(shape=[2], dtype=Float32, value=[ 7.50000000e+01,  7.50000000e+01]),
 Tensor(shape=[2], dtype=Float32, value=[ 3.00000000e+01,  3.00000000e+01]))
>>> print(aux)
(Tensor(shape=[2], dtype=Float32, value=[ 5.00000000e+00,  5.00000000e+00]),)
>>>
>>> # For given network to be differentiated with both inputs and weights, there are 4 cases.
>>> net = nn.Dense(10, 1)
>>> loss_fn = nn.MSELoss()
>>> def forward(inputs, labels):
...     logits = net(inputs)
...     loss = loss_fn(logits, labels)
...     return loss, logits
>>> inputs = Tensor(np.random.randn(16, 10).astype(np.float32))
>>> labels = Tensor(np.random.randn(16, 1).astype(np.float32))
>>> weights = net.trainable_params()
>>> # Case 1: gradient with respect to inputs.
>>> # Aux value does not contribute to the gradient.
>>> grad_fn = grad(forward, grad_position=(0, 1), weights=None, has_aux=True)
>>> inputs_gradient, (aux_logits,) = grad_fn(inputs, labels)
>>> print(len(inputs_gradient))
2
>>> print(aux_logits.shape)
(16, 1)
>>>
>>> # Case 2: gradient with respect to weights.
>>> grad_fn = grad(forward, grad_position=None, weights=weights, has_aux=True)
>>> params_gradient, (aux_logits,) = grad_fn(inputs, labels)
>>> print(len(weights), len(params_gradient))
2 2
>>> print(aux_logits.shape)
(16, 1)
>>>
>>> # Case 3: gradient with respect to inputs and weights.
>>> grad_fn = grad(forward, grad_position=0, weights=weights, has_aux=False)
>>> inputs_gradient, params_gradient = grad_fn(inputs, labels)
>>> print(len(weights), len(params_gradient))
2 2
>>> # Case 4: return the gradient with ids.
>>> import numpy as np

```

(下页继续)

(续上页)

```

>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore import Tensor, ops
>>> from mindspore import grad
>>>
>>> # Cell object to be differentiated
>>> class Net(nn.Cell):
...     def construct(self, x, y, z):
...         return x * y * z
>>> x = Tensor([1, 2], mindspore.float32)
>>> y = Tensor([-2, 3], mindspore.float32)
>>> z = Tensor([0, 3], mindspore.float32)
>>> net = Net()
>>> output = grad(net, grad_position=(1, 2), return_ids = True)(x, y, z)
>>> print(output)
((1, Tensor(shape=[2], dtype=Float32, value=[ 0.00000000e+00,  6.00000000e+00])),
 (2, Tensor(shape=[2], dtype=Float32, value=[-2.00000000e+00,  6.00000000e+00]))

```

1.6.2 mindspore.value_and_grad

`mindspore.value_and_grad(fn, grad_position=0, weights=None, has_aux=False, return_ids=False)`

生成求导函数，用于计算给定函数的正向计算结果和梯度。

函数求导包含以下三种场景：

1. 对输入求导，此时 `grad_position` 非 `None`，而 `weights` 是 `None`；
2. 对网络变量求导，此时 `grad_position` 是 `None`，而 `weights` 非 `None`；
3. 同时对输入和网络变量求导，此时 `grad_position` 和 `weights` 都非 `None`。

参数：

- **fn** (`Union[Cell, Function]`) - 待求导的网络或函数。
- **grad_position** (`Union[NoneType, int, tuple[int]]`，可选) - 指定输入中需要求导的位置索引。默认值：0。
 - 若为 `int` 类型，表示对单个输入求导；
 - 若为 `tuple` 类型，表示对输入中 `tuple` 对应的索引位置求导，其中索引从 0 开始；
 - 若是 `None`，表示不对输入求导，这种场景下，`weights` 非 `None`。
- **weights** (`Union[ParameterTuple, Parameter, list[Parameter]]`，可选) - 训练网络中需要求导的网络变量。一般可通过 `weights = net.trainable_params()` 获取。默认值：`None`。
- **has_aux** (`bool`，可选) - 是否返回辅助参数的标志。若为 `True`，`fn` 输出数量必须超过一个，其中只有 `fn` 第一个输出参与求导，其他输出值将直接返回。默认值：`False`。
- **return_ids** (`bool`，可选) - 返回的求导函数中是否包含 `grad_position` 或 `weights` 信息。若为 `True`，返回的求导函数中所有的梯度值 `gradient` 将被替换为：`[gradient, grad_position]` 或 `[gradient, weights]`。默认值：`False`。

返回：

`Function`，用于计算给定函数梯度的求导函数。例如 `out1, out2 = fn(*args)`，求导函数将返回 `((out1, out2), gradient)` 形式的结果，若 `has_aux` 为 `True`，那么 `out2` 不参与求导。若 `return_ids` 为 `True`，求导函数返回的 `gradient` 将被替代为 `[gradient, grad_position]` 或 `[gradient, weights]`。

异常：

- **ValueError** - 入参 *grad_position* 和 *weights* 同时为 *None*。
- **TypeError** - 入参类型不符合要求。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops, nn
>>> from mindspore import value_and_grad
>>>
>>> # Cell object to be differentiated
>>> class Net(nn.Cell):
...     def construct(self, x, y, z):
...         return x * y * z
>>> x = Tensor([1, 2], mindspore.float32)
>>> y = Tensor([-2, 3], mindspore.float32)
>>> z = Tensor([0, 3], mindspore.float32)
>>> net = Net()
>>> grad_fn = value_and_grad(net, grad_position=1)
>>> output, inputs_gradient = grad_fn(x, y, z)
>>> print(output)
[-0. 18.]
>>> print(inputs_gradient)
[0. 6.]
>>>
>>> # Function object to be differentiated
>>> def fn(x, y, z):
...     res = x * ops.exp(y) * ops.pow(z, 2)
...     return res, z
>>> x = Tensor(np.array([3, 3]).astype(np.float32))
>>> y = Tensor(np.array([0, 0]).astype(np.float32))
>>> z = Tensor(np.array([5, 5]).astype(np.float32))
>>> output, inputs_gradient = value_and_grad(fn, grad_position=(1, 2), weights=None, has_
aux=True)(x, y, z)
>>> print(output)
(Tensor(shape=[2], dtype=Float32, value= [ 7.50000000e+01,  7.50000000e+01]),
 Tensor(shape=[2], dtype=Float32, value= [ 5.00000000e+00,  5.00000000e+00]))
>>> print(inputs_gradient)
(Tensor(shape=[2], dtype=Float32, value= [ 7.50000000e+01,  7.50000000e+01]),
 Tensor(shape=[2], dtype=Float32, value= [ 3.00000000e+01,  3.00000000e+01]))
>>>
>>> # For given network to be differentiated with both inputs and weights, there are 3 cases.
>>> net = nn.Dense(10, 1)
>>> loss_fn = nn.MSELoss()
>>> def forward(inputs, labels):
...     logits = net(inputs)
...     loss = loss_fn(logits, labels)
...     return loss, logits
>>> inputs = Tensor(np.random.randn(16, 10).astype(np.float32))
>>> labels = Tensor(np.random.randn(16, 1).astype(np.float32))
>>> weights = net.trainable_params()
>>>
>>> # Case 1: gradient with respect to inputs.
>>> # For has_aux is set True, only loss contributes to the gradient.
```

(下页继续)

(续上页)

```

>>> grad_fn = value_and_grad(forward, grad_position=0, weights=None, has_aux=True)
>>> (loss, logits), inputs_gradient = grad_fn(inputs, labels)
>>> print(logits.shape)
(16, 1)
>>> print(inputs.shape, inputs_gradient.shape)
(16, 10) (16, 10)
>>>
>>> # Case 2: gradient with respect to weights.
>>> # For has_aux is set True, only loss contributes to the gradient.
>>> grad_fn = value_and_grad(forward, grad_position=None, weights=weights, has_aux=True)
>>> (loss, logits), params_gradient = grad_fn(inputs, labels)
>>> print(logits.shape)
(16, 1)
>>> print(len(weights), len(params_gradient))
2 2
>>>
>>> # Case 3: gradient with respect to inputs and weights.
>>> # For has_aux is set False, both loss and logits contribute to the gradient.
>>> grad_fn = value_and_grad(forward, grad_position=0, weights=weights, has_aux=False)
>>> (loss, logits), (inputs_gradient, params_gradient) = grad_fn(inputs, labels)
>>> print(logits.shape)
(16, 1)
>>> print(inputs.shape, inputs_gradient.shape)
(16, 10) (16, 10)
>>> print(len(weights), len(params_gradient))
2 2

```

1.6.3 mindspore.get_grad

`mindspore.get_grad(gradients, identifier)`

当`mindspore.grad()`或`mindspore.value_and_grad()`的`return_ids`参数设置为`True`时, 将`mindspore.grad`的返回值或`mindspore.value_and_grad`的第二个返回值作为`gradients`, 再根据`identifier`在`gradients`中找到对应的梯度值。

根据`identifier`查找梯度值包含以下两种场景:

1. `identifier` 为指定求导输入位置的索引;
2. `identifier` 为网络变量。

参数:

- **gradients** (Union[tuple[int, Tensor], tuple[tuple, tuple]]) - `mindspore.grad()` 参数 `return_ids` 为 `True` 时的返回值。
- **identifier** (Union[int, Parameter]) - 指定求导输入位置的索引或者在`mindspore.grad()`中被使用的网络变量。

返回:

`identifier` 所对应的 Tensor 梯度值。

异常:

- **RuntimeError** - 无法找到 `identifier` 所对应的梯度值。
- **TypeError** - 入参类型不符合要求。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> from mindspore import grad, get_grad
>>>
>>> # Cell object to be differentiated
>>> class Net(nn.Cell):
...     def construct(self, x, y, z):
...         return x * y * z
>>> x = Tensor([1, 2], mindspore.float32)
>>> y = Tensor([-2, 3], mindspore.float32)
>>> z = Tensor([0, 3], mindspore.float32)
>>> net = Net()
>>> out_grad = grad(net, grad_position=(1, 2), return_ids=True)(x, y, z)
>>> output = get_grad(out_grad, 1)
>>> print(output)
[0. 6.]
```

1.6.4 mindspore.jacfwd

`mindspore.jacfwd(fn, grad_position=0, has_aux=False)`

通过前向模式计算给定网络的 Jacobian 矩阵，对应 前向模式自动微分。当网络输出数量远大于输入数量时，使用前向模式求 Jacobian 矩阵比反向模式性能更好。

参数:

- **fn** (Union[Cell, Function]) - 待求导的函数或网络。以 Tensor 为入参，返回 Tensor 或 Tensor 数组。
- **grad_position** (Union[int, tuple[int]], 可选) - 指定求导输入位置的索引。若为 int 类型，表示对单个输入求导；若为 tuple 类型，表示对 tuple 内索引的位置求导，其中索引从 0 开始。默认值：0。
- **has_aux** (bool, 可选) - 若 `has_aux` 为 True，只有 `fn` 的第一个输出参与 `fn` 的求导，其他输出将直接返回。此时，`fn` 的输出数量必须多于一个。默认值：False。

返回:

Function，用于计算给定函数的 Jacobian 矩阵。例如 `out1, out2 = fn(*args)`，若 `has_aux` 为 True，梯度函数将返回 `(Jacobian, out2)` 形式的结果，其中 `out2` 不参与求导，若为 False，将直接返回 `Jacobian`。

异常:

- **TypeError** - `grad_position` 或 `has_aux` 类型不符合要求。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import jacfwd
>>> from mindspore import Tensor
>>> class MultipleInputsMultipleOutputsNet(nn.Cell):
...     def construct(self, x, y, z):
...         return x ** 2 + y ** 2 + z ** 2, x * y * z
```

(下页继续)

(续上页)

```
>>> x = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
>>> y = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
>>> z = Tensor(np.array([[1, 1], [1, 1]]).astype(np.float32))
>>> net = MultipleInputsMultipleOutputsNet()
>>> jac, aux = jacfwd(net, grad_position=0, has_aux=True)(x, y, z)
>>> print(jac)
[[[ 2.  0.]
  [ 0.  0.]]
 [ 0.  4.]
 [ 0.  0.]]]
[[[ 0.  0.]
  [ 6.  0.]]
 [ 0.  0.]
 [ 0.  8.]]]
>>> print(aux)
[[ 1.  4.]
 [ 9. 16.]]
```

1.6.5 mindspore.jacrev

`mindspore.jacrev(fn, grad_position=0, has_aux=False)`

通过反向模式计算给定网络的 Jacobian 矩阵，对应 反向模式自动微分。当网络输出数量远小于输入数量时，使用反向模式求 Jacobian 矩阵比前向模式性能更好。

参数：

- **fn** (Union[Cell, Function]) - 待求导的函数或网络。以 Tensor 为入参，返回 Tensor 或 Tensor 数组。
- **grad_position** (Union[int, tuple[int]], 可选) - 指定求导输入位置的索引。若为 int 类型，表示对单个输入求导；若为 tuple 类型，表示对 tuple 内索引的位置求导，其中索引从 0 开始。默认值：0。
- **has_aux** (bool, 可选) - 若 has_aux 为 True，只有 fn 的第一个输出参与 fn 的求导，其他输出将直接返回。此时，fn 的输出数量必须超过一个。默认值：False。

返回：

Function，用于计算给定函数的 Jacobian 矩阵。例如 `out1, out2 = fn(*args)`，若 has_aux 为 True，梯度函数将返回 `(Jacobian, out2)` 形式的结果，其中 out2 不参与求导，若为 False，将直接返回 Jacobian。

异常：

- **TypeError** - grad_position 或 has_aux 类型不符合要求。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import jacrev
>>> from mindspore import Tensor
>>> class MultipleInputsMultipleOutputsNet(nn.Cell):
...     def construct(self, x, y, z):
...         return x ** 2 + y ** 2 + z ** 2, x * y * z
>>> x = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
>>> y = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
```

(下页继续)

(续上页)

```
>>> z = Tensor(np.array([[1, 1], [1, 1]]).astype(np.float32))
>>> net = MultipleInputsMultipleOutputsNet()
>>> jac, aux = jacrev(net, grad_position=0, has_aux=True)(x, y, z)
>>> print(jac)
[[[ [ 2.  0.]
      [ 0.  0.]
      [ 0.  4.]
      [ 0.  0.]]]
  [[ [ 0.  0.]
      [ 6.  0.]
      [ 0.  0.]
      [ 0.  8.]]]]
>>> print(aux)
[[ 1.  4.]
 [ 9. 16.]]
```

1.6.6 mindspore.jvp

`mindspore.jvp(fn, inputs, v, has_aux=False)`

计算给定网络的雅可比向量积 (Jacobian-vector product, JVP)。JVP 的计算过程可参考 [前向模式自动微分](#)。

参数：

- **fn** (Union[Function, Cell]) - 待求导的函数或网络。以 `Tensor` 为入参，返回 `Tensor` 或 `Tensor` 数组。
- **inputs** (Union[Tensor, tuple[Tensor], list[Tensor]]) - 输入网络 `fn` 的入参。
- **v** (Union[Tensor, tuple[Tensor], list[Tensor]]) - 与雅可比矩阵相乘的向量，`shape` 和 `type` 与网络的输入一致。
- **has_aux** (bool) - 若 `has_aux` 为 `True`，只有 `fn` 的第一个输出参与 `fn` 的求导，其他输出将直接返回。此时，`fn` 的输出数量必须超过一个。默认值：`False`。

返回：

- **net_output** (Union[Tensor, tuple[Tensor]]) - 输入网络的正向计算结果。
- **jvp** (Union[Tensor, tuple[Tensor]]) - 雅可比向量积的结果。
- **aux_value** (Union[Tensor, tuple[Tensor]], 可选) - 只有 `has_aux` 值为 `True`，才返回 `aux_value`。`aux_value` 是 `fn(inputs)` 的第一个除外的其他输出，且不参与 `fn` 的求导。

异常：

- **TypeError** - `inputs` 或 `v` 类型不符合要求。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import jvp
>>> from mindspore import Tensor
>>> import mindspore.nn as nn
>>> class Net(nn.Cell):
...     def construct(self, x, y):
...         return x**3 + y
>>> x = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
```

(下页继续)

(续上页)

```

>>> y = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
>>> v = Tensor(np.array([[1, 1], [1, 1]]).astype(np.float32))
>>> output = jvp(Net(), (x, y), (v, v))
>>> print(output[0])
[[ 2. 10.]
 [30. 68.]]
>>> print(output[1])
[[ 4. 13.]
 [28. 49.]]
>>>
>>> def fn(x, y):
...     return x ** 3 + y, y
>>> output, jvp_out, aux = jvp(fn, (x, y), (v, v), has_aux=True)
>>> print(output)
[[ 2. 10.]
 [30. 68.]]
>>> print(jvp_out)
[[ 4. 13.]
 [28. 49.]]
>>> print(aux)
[[ 1. 2.]
 [3. 4.]]

```

1.6.7 mindspore.vjp

`mindspore.vjp(fn, inputs, weights=None, has_aux=False)`

计算给定网络的向量雅可比积 (vector-jacobian-product, VJP)。VJP 对应 反向模式自动微分。

参数：

- **fn** (Union[Function, Cell]) - 待求导的函数或网络。以 Tensor 为入参，返回 Tensor 或 Tensor 数组。
- **inputs** (Union[Tensor, tuple[Tensor], list[Tensor]]) - 输入网络 *fn* 的入参。
- **weights** (Union[ParameterTuple, Parameter, list[Parameter]])，可选) - 训练网络中需要返回梯度的网络变量。一般可通过 `weights = net.trainable_params()` 获取。默认值：None。
- **has_aux** (bool，可选) - 若 *has_aux* 为 True，只有 *fn* 的第一个输出参与 *fn* 的求导，其他输出将直接返回。此时，*fn* 的输出数量必须超过一个。默认值：False。

返回：

正向输出和计算 vjp 的功能。

- **net_output** (Union[Tensor, tuple[Tensor]]) - *fn(inputs)* 的输出。特别是当 *has_aux* 设置为 True 时，*net_output* 是 *fn(inputs)* 的第一个输出。
- **vjp_fn** (Function) - 用于求解向量雅可比积的函数。接收 shape 和 type 与 *net_output* 一致的输入。
- **aux_value** (Union[Tensor, tuple[Tensor]])，可选) - 若 *has_aux* 为 True，则返回 *aux_value*。*aux_value* 是 *fn(inputs)* 的除第一个外的其他输出，且不参与 *fn* 的求导。

异常：

- **TypeError** - *inputs* 或 *v* 类型不符合要求。

支持平台：

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import vjp
>>> from mindspore import Tensor
>>> class Net(nn.Cell):
...     def construct(self, x, y):
...         return x**3 + y
>>> x = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
>>> y = Tensor(np.array([[1, 2], [3, 4]]).astype(np.float32))
>>> v = Tensor(np.array([[1, 1], [1, 1]]).astype(np.float32))
>>> outputs, vjp_fn = vjp(Net(), x, y)
>>> print(outputs)
[[ 2. 10.]
 [30. 68.]]
>>> gradient = vjp_fn(v)
>>> print(gradient)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 3.00000000e+00,  1.20000000e+01],
 [ 2.70000000e+01,  4.80000000e+01]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 1.00000000e+00,  1.00000000e+00],
 [ 1.00000000e+00,  1.00000000e+00]]))
>>> def fn(x, y):
...     return 2 * x + y, y ** 3
>>> outputs, vjp_fn, aux = vjp(fn, x, y, has_aux=True)
>>> gradient = vjp_fn(v)
>>> print(outputs)
[[ 3.  6.]
 [ 9. 12.]]
>>> print(aux)
[[ 1.  8.]
 [27. 64.]]
>>> print(gradient)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 2.00000000e+00,  2.00000000e+00],
 [ 2.00000000e+00,  2.00000000e+00]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 1.00000000e+00,  1.00000000e+00],
 [ 1.00000000e+00,  1.00000000e+00]]))
```

1.7 并行优化

1.7.1 自动向量化

mindspore.vmap

自动向量化 (Vectorizing Map, vmap), 是一种用于沿参数轴映射函数 *fn* 的高阶函数。

mindspore.vmap

`mindspore.vmap(fn, in_axes=0, out_axes=0)`

自动向量化 (Vectorizing Map, vmap), 是一种用于沿参数轴映射函数 *fn* 的高阶函数。

Vmap 由 Jax 率先提出, 它消除了算子对 batch 维度的限制, 并提供更加方便、统一的运算符表达。同时, 用户还可以与 `mindspore.grad()` 等其他功能模块组合使用, 提高开发效率。

此外, 由于自动向量化并不在函数外部执行循环, 而是将循环逻辑下沉至函数的各个原语操作中, 以获得更好的性能。当与图算融合特性相结合时, 执行效率将进一步提高。

警告: 这是一个实验性 API, 后续可能修改或删除。

说明:

- **vmap** 的能力来自于原语的 **VmapRules** 实现。虽然我们为用户自定义算子设计了一个通用规则, 但是我们无法保证它对所有算子都能很好地工作, 可能出现未知异常, 用户需知晓使用风险。
- 当在 **vmap** 作用域内调用随机数生成方法时, 每次在向量函数之间生成相同的随机数。如果希望每个向量分支使用不同的随机数, 需要提前从外部生成一批随机数, 然后将其传入 **vmap**。

参数:

- **fn** (`Union[Cell, Function, CellList]`) - 待沿参数轴映射的函数, 该函数至少拥有一个输入参数并且返回值为一个或多个 **Tensor** 或 **Tensor** 支持的数据类型。当 *fn* 的类型是 **CellList** 时, 为模型集成场景, 需要确保每个单元的结构相同, 并且单元数量与映射轴索引对应的 *size* (*axis_size*) 一致。
- **in_axes** (`Union[int, list, tuple]`) - 指定输入参数映射的轴索引。默认值: 0。
 - 如果 *in_axes* 是一个整数, 则 *fn* 的所有输入参数都将根据此轴索引进行映射。
 - 如果 *in_axes* 是一个 **tuple** 或 **list**, 仅支持由整数或 **None** 组成, 则其长度应与 *fn* 的输入参数的个数一致, 分别表示相应位置参数的映射轴索引。请注意, 每个参数对应的整数轴索引的取值范围必须在 $[-ndim, ndim)$ 中, 其中 *ndim* 是参数的维度。
 - 如果 *in_axes* 是 **None** 表示不沿任何轴映射, 并且 *in_axes* 中必须至少有一个位置参数的映射轴索引不为 **None**。所有参数的映射轴索引对应的 *size* (*axis_size*) 必须相等。
- **out_axes** (`Union[int, list, tuple]`) - 指定映射轴呈现在输出中的索引位置。默认值: 0。
 - 如果 *out_axes* 是一个整数, 则 *fn* 的所有输出都根据此 *axis* 指定。
 - 如果 *out_axes* 是一个 **tuple** 或 **list**, 仅支持由整数或 **None** 组成, 其长度应与 *fn* 的输出个数相等。请注意, 每个输出对应的整数轴索引的取值范围必须在 $[-ndim, ndim)$ 中, 其中 *ndim* 是 **vmap** 映射后的函数的输出的维度。
 - 所有具有非 **None** 映射轴的输出只能指定非 **None** 的 *out_axes*, 如果具有 **None** 映射轴的输出指定非 **None** 的 *out_axes*, 结果将沿映射轴进行广播。

返回:

Function, 返回 *fn* 的自动向量化后的函数。此函数的输入参数和输出与 *fn* 的相对应, 但它在 *in_axes* 和 *out_axes* 指定的位置新增了额外的批处理维度。

异常:

- **RuntimeError** -
 - 如果 *in_axes* 或 *out_axes* 中的基本元素不是 **None** 或整数。

- 如果 *in_axes* 或 *out_axes* 中的所有基本元素均为 `None`。
- 如果 *in_axes* 不是单个整数，并且 *in_axes* 的长度不等于输入参数的个数。
- 如果 *out_axes* 不是单个整数，并且 *out_axes* 的长度不等于输出个数。
- 如果 *vmap* 范围内每个参数的 *axis_size* 不相等。
- 如果 *in_axes* 或 *out_axes* 中的 *axis* 超出边界限制。

支持平台:

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import vmap
>>> def test_vmap(x, y, z):                                     # ([a],[a],[a]) -> [a]
    ... return x + y + z
>>> x = Tensor(np.array([[1, 2], [3, 4], [5, 6]]).astype(np.float32)) # [b, a]
>>> y = Tensor(np.array([[-3, -2, -1], [3, 2, 1]]).astype(np.float32)) # [a, b]
>>> z = Tensor(np.array([0, 3]).astype(np.float32))                # [a]
>>> output = vmap(test_vmap, in_axes=(0, 1, None), out_axes=1)(x, y, z) # ([b, a],[a, b],
    ... [a]) -> [a, b]
>>> print(output)
[[-2  1  4]
 [ 8  9 10]]
```

1.7.2 重计算

```
mindspore.recompute
```

该函数用于减少显存的使用，当运行选定的模块时，不再保存其中的前向计算产生的激活值，我们将在反向传播时，重新计算前向的激活值。

[mindsponge.recompute](https://mindsponge.com)

```
mindspore.recompute(block, *args, **kwargs)
```

该函数用于减少显存的使用，当运行选定的模块时，不再保存其中的前向计算产生的激活值，我们将在反向传播时，重新计算前向的激活值。

说明:

- 重计算函数只支持继承自 `Cell` 对象的模块。
- 该函数当前只支持 `PyNative` 模式，在图模式下，可以尝试使用 `Cell.recompute()` 接口。
- 当使用重计算函数时，传入的网络模块不能使用 `jit` 装饰器。

参数:

- **block (Cell)** - 需要重计算的模块。

- **args** (tuple) - 指需要重计算的模块的前向输入。
- **kwargs** (dict) - 可选输入。

返回：

同 **block** 的返回类型相同。

异常：

- **TypeError** - 如果 *block* 不是 **Cell** 对象。
- **AssertionError** - 如果执行模式不是 **PyNative** 模式。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore import Tensor, recompute
>>> class MyCell(nn.Cell):
...     def __init__(self):
...         super(MyCell, self).__init__(auto_prefix=False)
...         self.conv = nn.Conv2d(2, 2, 2, has_bias=False, weight_init='ones')
...         self.relu = ops.ReLU()
...
...     def construct(self, x):
...         y = recompute(self.conv, x)
...         return self.relu(y)
>>> inputs = Tensor(np.ones([2, 2, 2, 2]).astype(np.float32) * 2)
>>> my_net = MyCell()
>>> grad = ops.grad(my_net)(inputs)
>>> print(grad)
[[[2. 4.]
  [4. 8.]]
 [[2. 4.]
  [4. 8.]]]
[[[2. 4.]
  [4. 8.]]
 [[2. 4.]
  [4. 8.]]]
```

1.8 即时编译

<code>mindspore.JitConfig</code>	编译时所使用的 JitConfig 配置项。
<code>mindspore.jit</code>	将 Python 函数编译为一张可调用的 MindSpore 图。
<code>mindspore.jit_class</code>	用户自定义类的类装饰器。
<code>mindspore.ms_memory_recycle</code>	回收 MindSpore 使用的内存。
<code>mindspore.mutable</code>	设置一个常量值为可变的。
<code>mindspore.constexpr</code>	在图模式下，用来计算图编译过程中的常量值，以提升编译性能。

下页继续

表 12 – 续上页

<code>mindspore.lazy_inline</code>	指定一个 cell 是可复用的。
<code>mindspore.no_inline</code>	指定 python 函数是可复用的。
<code>mindspore.set_recursion_limit</code>	在图编译前指定函数调用的递归深度限制。

1.8.1 mindspore.JitConfig

class mindspore.**JitConfig** (*jit_level="", exc_mode='auto', jit_syntax_level="", debug_level='RELEASE', infer_boost='off',*
***kwargs*)

编译时所使用的 JitConfig 配置项。

参数：

- **jit_level** (str, 可选) - 用于控制编译优化等级，支持 ["O0", "O1", "O2"]。默认值：""，框架自动选择执行方式。不推荐使用，建议使用 JIT 装饰器。
 - "O0": 除影响功能的必要优化外，其他优化均关闭，使用逐算子执行的执行方式。
 - "O1": 使能常用优化和自动算子融合优化，使用逐算子执行的执行方式。这是一个实验性的优化等级，在持续完善中。
 - "O2": 开启极致性能优化，使用下沉的执行方式。
- **exc_mode** (str, 可选) - 用于控制模型的执行方式，目前仅支持 "auto"。默认值："auto"。
 - "auto": 框架自动选择执行方式。
 - "sink": 支持网络整体一次下发加载到 device 上，后续由输入驱动执行，无需逐个算子遍历下发，以便取得更好的执行性能。该模式仅在昇腾后端支持。
 - "no_sink": 网络模型按照单算子逐个异步下发的方式执行。
- **jit_syntax_level** (str, 可选) - 设置 JIT 语法支持级别，其值必须为 "STRICT", "LAX" 或 ""。默认是空字符串，表示忽略该项 JitConfig 配置，将使用 ms.context 的 jit_syntax_level，ms.context 请参考 [set_context](#)。默认值：""。
 - "STRICT": 仅支持基础语法，且执行性能最佳。可用于 MindIR 导入导出。
 - "LAX": 最大程度地兼容 Python 所有语法。执行性能可能会受影响，不是最佳。由于存在可能无法导出的语法，不能用于 MindIR 导入导出。
- **debug_level** (int) - 设置调试过程的配置。其值必须为 RELEASE 或 DEBUG。默认值：RELEASE。
 - RELEASE: 正常场景下使用，一些调试信息会被丢弃以获取一个较好的编译性能。
 - DEBUG: 当错误发生时，用来调试，在编译过程中，更多的调试信息会被记录下来。
- **infer_boost** (str, 可选): 使能推理加速模式。只能设置为 "on" 或 "off"。默认设置为 "off"，表示关闭推理加速。当使能了推理加速模式，MindSpore 会优先使用高性能算子库，并优化运行时，提高推理性能。注意：当前推理加速模式只能在 *jit_level* 设为 "O0" 时使用，且仅支持 Atlas A2 系列产品。
- **kwargs** (dict) - 关键字参数字典。

样例：

```
>>> from mindspore import JitConfig
>>>
>>> jitconfig = JitConfig(jit_level="O1")
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
```

(下页继续)

(续上页)

```
>>> net = LeNet5()
>>>
>>> net.set_jit_config(jitconfig)
```

1.8.2 mindspore.jit

`mindspore.jit` (*function: Optional[Callable] = None, *, capture_mode: str = 'ast', jit_level: str = 'O0', dynamic: int = 0, fullgraph: bool = False, backend: str = "", **options*)

将 Python 函数编译为一张可调用的 MindSpore 图。

MindSpore 可以在运行时对图进行优化。

说明:

- 不支持在静态图模式下，运行带装饰 `@jit(capture_mode="bytecode")` 的函数，此时该装饰 `@jit(capture_mode="bytecode")` 视为无效。
- 不支持在 `@jit(capture_mode="ast")` 装饰的函数内部调用带装饰 `@jit(capture_mode="bytecode")` 的函数，该装饰 `@jit(capture_mode="bytecode")` 视为无效。

参数:

- **function** (Function, 可选) - 要编译成图的 Python 函数。默认值: None。

关键字参数:

- **capture_mode** (str, 可选) - 创建一张可调用的 MindSpore 图的方式，可选值有 "ast"、"bytecode" 和 "trace"。默认值: "ast"。
 - **ast**: 解析 Python 的 ast 以构建静态图。
 - **bytecode**: 在运行时解析 Python 字节码以构建静态图。这是一个实验特性，可能会被更改或者删除。
 - **trace**: 追踪 Python 代码的执行以构建静态图。这是一个实验特性，可能会被更改或者删除。
- **jit_level** (str, 可选) - 控制编译优化的级别。目前仅在使用 `ms_backend` 后端时生效。可选值有 "O0" 和 "O1"。默认值: "O0"。
 - **O0**: 除必要影响功能的优化外，其他优化均关闭。
 - **O1**: 使能常用优化和自动算子融合优化。
- **dynamic** (int, 可选) - 是否需要进行动态 shape 编译。默认值: 0。其值范围如下:
 - **0**: 不进行动态 shape 编译。
 - **1**: 使能动态 shape 编译，自动检测 shape 的变化。
- **fullgraph** (bool, 可选) - 是否捕获整个函数来编译成图。如果为 `False`，`jit` 会尽可能地尝试兼容函数中的所有 Python 语法。如果为 `True`，则需要整个函数都可以被捕获成图，否则（即有不支持的 Python 语法），会抛出一个异常。当前只对 `capture_mode` 为 "ast" 时生效。默认值: `False`。
- **backend** (str, 可选) - 使用的编译后端。如果该参数未被设置，框架默认 Atlas 训练系列产品为 GE 后端，默认其他产品包括 Atlas A2 训练系列产品为 `ms_backend` 后端。
 - **ms_backend**: 使用逐算子执行的执行方式。
 - **GE**: 使用下沉的执行方式，整个模型都会下沉到 `device` 侧执行，仅可作用于模型的最外层 Cell。仅支撑昇腾硬件。

- ****options** (dict) - 传给编译后端的选项字典。

某些配置适用于特定的设备和后端，有关详细信息，请参见下表：

配置项	硬件平台支持	编译后端支持
<code>disable_format_transform</code>	GPU	ms_backend
<code>exec_order</code>	Ascend	ms_backend
<code>ge_options</code>	Ascend	GE
<code>infer_boost</code>	Ascend	ms_backend

- **disable_format_transform** (bool, 可选) - 表示是否取消 NCHW 到 NHWC 的自动格式转换功能。当 fp16 的网络性能不如 fp32 的时，可以设置 `disable_format_transform` 为 `True`，以尝试提高训练性能。默认值：`False`。
- **exec_order** (str, 可选) - 算子执行时的排序方法，GRAPH_MODE(0) 下 jit_level 为 O0 或者 O1 时生效。不同的执行顺序会使得网络的执行内存和性能有所差异，当前仅支持两种排序方法：`bfs` 和 `dfs`，默认方法为 `bfs`。
 - * `bfs`: 默认的排序方法，广度优先排序，具备较好的通信掩盖效果，执行性能相对较好。
 - * `dfs`: 可选择的排序方法，深度优先排序，性能相对 `bfs` 执行序较差，但内存占用较少，建议在其他执行序 OOM 的场景下尝试 `dfs`。
- **ge_options** (dict) - 设置 GE 的 `options` 配置项，配置项分为 `global` 和 `session` 二类。这是一个实验特性，可能会被更改或者删除。详细的配置请查询 [options 配置说明](#)。
 - * `global` (dict): 设置 `global` 类的选项。
 - * `session` (dict): 设置 `session` 类的选项。
- **infer_boost** (str, 可选) - 用来使能推理模式。默认值为 “off”，表示关闭。其值范围如下：
 - * `on`: 开启推理模式，推理性能得到较大提升。
 - * `off`: 关闭推理模式，使用前向运算进行推理，性能较差。

返回：

函数，如果 `fn` 不是 `None`，则返回一个已经将输入 `fn` 编译成图的可执行函数；如果 `fn` 为 `None`，则返回一个装饰器。当这个装饰器使用单个 `fn` 参数进行调用时，等价于 `fn` 不是 `None` 的场景。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> from mindspore import jit
...
>>> x = Tensor(np.ones([1, 1, 3, 3]).astype(np.float32))
>>> y = Tensor(np.ones([1, 1, 3, 3]).astype(np.float32))
...
>>> # create a callable MindSpore graph by calling jit
>>> def tensor_add(x, y):
...     z = x + y
...     return z
...
```

(下页继续)

(续上页)

```

>>> tensor_add_graph = jit(function=tensor_add)
>>> out = tensor_add_graph(x, y)
...
>>> # create a callable MindSpore graph through decorator @jit
>>> @jit
... def tensor_add_with_dec(x, y):
...     z = x + y
...     return z
...
>>> out = tensor_add_with_dec(x, y)
...
>>> # create a callable MindSpore graph and capture the entire function into the graph
>>> @jit(fullgraph=True)
... def tensor_add_fullgraph(x, y):
...     z = x + y
...     return z
...
>>> out = tensor_add_fullgraph(x, y)

```

1.8.3 mindspore.jit_class

`mindspore.jit_class(cls)`

用户自定义类的类装饰器。

MindSpore 可以通过 `jit_class` 识别用户定义的类，从而获取这些类的属性和方法。

参数：

- **cls** (Class) - 用户自定义的类。

返回：

类。

异常：

- **TypeError** - 如果 `jit_class` 用于非 `class` 类型或者 `nn.Cell`。
- **AttributeError** - 如果调用了 `jit_class` 装饰的类定义了私有属性或魔术方法。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore.nn as nn
>>> from mindspore import jit_class
...
>>> @jit_class
... class UserDefinedNet:
...     def __init__(self):
...         self.value = 10
...
...     def func(self, x):
...         return 2 * x
...
>>> class Net(nn.Cell):

```

(下页继续)

(续上页)

```

...     def __init__(self):
...         super(Net, self).__init__()
...         self.net = UserDefinedNet()
...
...     def construct(self, x):
...         out = self.net.value + self.net.func(x)
...         return out
...
>>> net = Net()
>>> out = net(5)
>>> print(out)
20

```

1.8.4 mindspore.ms_memory_recycle

`mindspore.ms_memory_recycle()`

回收 MindSpore 使用的内存。

在一个进程中训练多个神经网络模型时，MindSpore 使用的内存非常大，这是因为 MindSpore 为每个模型缓存了运行时的内存。

为了回收这些缓存的内存，用户可以在训练一个模型后调用这个函数。

样例：

```

>>> import mindspore as ms
>>> ms.ms_memory_recycle()

```

1.8.5 mindspore.mutable

`mindspore.mutable(input_data, dynamic_len=False)`

设置一个常量值为可变的。

当前除了 Tensor，所有顶层网络的输入，例如标量、tuple、list 和 dict，都被当做常量值。常量值是不能求导的，而且在编译优化阶段常量值会被忽略掉。另外，当网络的输入是 tuple[Tensor]、list[Tensor] 或 Dict[Tensor] 时，即使里面 Tensor 的 shape 和 dtype 没有发生变化，在多次调用同一个网络的时候，这个网络每次都会被重新编译，这是因为这些类型的输入被当做常量值处理了。

为解决以上的问题，我们提供了 mutable 接口去设置网络的常量输入为“可变的”。一个“可变的”输入意味着这个输入成为了像 Tensor 一样的变量，最重要的是，我们可以对其进行求导了。

当 input_data 是 tuple 或者 list 并且 dynamic_len 是 False 的情况下，mutable 的返回值是一个固定长度的 tuple 或者 list，且其中的每一个元素都是可变的。当 dynamic_len 被设置为 True 的时候，返回的 tuple 或者 list 长度是动态的。

当 mutable 返回的动态长度的 tuple 或者 list 被作为网络的输入并且这个网络被重复调用，即使每一次的输入的 tuple 或者 list 长度都不一致，这个网络也不需要被重新编译。

参数：

- **input_data** (Union[Tensor, scalar, tuple, list, dict]) - 要设置为可变的输入数据。如果 input_data 是 list、tuple 或者 dict，其内部元素的类型也需要是这些有效类型中的一个。
- **dynamic_len** (bool, 可选) - 是否要将整个序列设置为动态长度的。在图编译内，如果 dynamic_len 被设置为 True，那么 input_data 必须为 tuple 或者 list，并且其中的元素必须有相同的类型以及 shape。默认值：False。

警告： 这是一个实验性 API，后续可能修改或删除。*dynamic_len* 是实验性质的参数，暂不支持 *dynamic_len* 为 True。

说明：当前该接口只在图模式下生效。

返回：

状态设置为可变的原输入数据。

异常：

- **TypeError** - 如果 *input_data* 不是 Tensor、scalar、tuple、list 或 dict 中的任一种类型或者不是它们的嵌套结构。
- **TypeError** - 如果 *dynamic_len* 被设置为 True 并且 *input_data* 不是 tuple 或者 list。
- **ValueError** - 如果 *dynamic_len* 被设置为 True，*input_data* 是 tuple 或者 list 的情况下，其中的元素的 shape 或者类型不一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import mutable, nn, ops, Tensor, context
>>> from mindspore import dtype as mstype
>>> context.set_context(mode=context.GRAPH_MODE)
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.matmul = ops.MatMul()
...
...     def construct(self, z):
...         x = z[0]
...         y = z[1]
...         out = self.matmul(x, y)
...         return out
...
>>> class GradNetWrtX(nn.Cell):
...     def __init__(self, net):
...         super(GradNetWrtX, self).__init__()
...         self.net = net
...         self.grad_op = ops.GradOperation()
...
...     def construct(self, z):
...         gradient_function = self.grad_op(self.net)
...         return gradient_function(z)
...
>>> z = mutable((Tensor([[0.5, 0.6, 0.4], [1.2, 1.3, 1.1]], dtype=mstype.float32),
...                  Tensor([[0.01, 0.3, 1.1], [0.1, 0.2, 1.3], [2.1, 1.2, 3.3]], dtype=mstype.
↪float32)))
>>> output = GradNetWrtX(Net())(z)
>>> print(output)
(Tensor(shape=[2, 3], dtype=Float32, value=
[[ 1.41000009e+00,  1.60000002e+00,  6.59999943e+00],
 [ 1.41000009e+00,  1.60000002e+00,  6.59999943e+00]]), Tensor(shape=[3, 3], dtype=Float32, ↪
↪value=
```

(下页继续)

(续上页)

```
[ [ 1.70000005e+00, 1.70000005e+00, 1.70000005e+00],
  [ 1.89999998e+00, 1.89999998e+00, 1.89999998e+00],
  [ 1.50000000e+00, 1.50000000e+00, 1.50000000e+00] ] )
```

1.8.6 mindspore.constexpr

`mindspore.constexpr` (*fn=None, get_instance=True, name=None, reuse_result=True, check=True*)

在图模式下，用来计算图编译过程中的常量值，以提升编译性能。

参数：

- **fn** (function, 可选) - *fn* 用作输出算子的 `infer_value`。默认值：None。
- **get_instance** (bool, 可选) - 如果为 True，返回算子的实例，否则返回算子的类。默认值：True。
- **name** (str, 可选) - 定义算子的名称。如果 *name* 为 None，则使用函数名称作为算子名称。默认值：None。
- **reuse_result** (bool, 可选) - 如果为 True，仅实际执行算子一次，后续调用会直接返回结果。否则每次都实际执行算子来获取结果。默认值：True。
- **check** (bool, 可选) - 如果为 True，参数将被 check。如果参数不是常量，将发出警告信息。默认值：True。

样例：

```
>>> import mindspore as ms
>>> # define a constant calculate function with for loop inside and use use constexpr to
    ↪ accelerate the compile
>>> # process.
>>> @ms.constexpr
... def for_loop_calculate(range_num):
...     out = 0
...     for i in range(range_num):
...         if i % 2 == 0 and i % 7 != 0:
...             out = out + i
...     return out // range_num
...
>>> # construct a net and run with GRAPH_MODE.
>>> @ms.jit
... def my_func(x):
...     new_shape = for_loop_calculate(100000)
...     return ms.ops.broadcast_to(x, (new_shape, ))
...
>>> out = my_func(ms.Tensor([1]))
>>> print(out.shape)
>>> (21428, )
```

1.8.7 mindspore.lazy_inline

`mindspore.lazy_inline (fn=None, attrs=None, policy=None)`

指定一个 cell 是可复用的。该 cell 在前端编译为可复用的子图，后端根据策略内联。注册此装饰器到 cell 的内置函数 `__init__` 时，此装饰器会按照 `attrs` 的值去添加 `__init__` 函数对应的入参，以此作为 cell 的属性。

详细功能说明可参考 [使用 lazy_inline 装饰器](#)。

警告：该特性仅支持 Ascend，其他硬件不支持。cell 的 `construct` 函数参数必须是位置参数或者关键字参数，且不能有默认值。lazy inline 装饰的 cell 不包含控制流。

参数：

- **fn** (function) - cell 的 `__init__` 函数。
- **attrs** (Union[list[string], string]) - cell 需要添加的属性列表。
- **policy** (Union[None, "front"]) - inline 的策略。默认值：None。
 - None: Cell 编译为可复用的子图，该子图不 inline 到大图中。
 - "front": Cell 先编译为可复用的子图，然后 inline 到大图中。

返回：

function，原始函数。

支持平台：

Ascend

样例：

```
>>> import os
>>> import numpy as np
>>> from mindspore import Tensor
>>> import mindspore.nn as nn
>>> from mindspore import lazy_inline
>>> from mindspore import context
>>> from mindspore import ops
>>> def conv3x3(in_channels, out_channels, stride=1, padding=1, pad_mode='pad'):
...     return nn.Conv2d(in_channels, out_channels,
...                       kernel_size=3, stride=stride, padding=padding, pad_mode=pad_mode)
...
>>> def conv1x1(in_channels, out_channels, stride=1, padding=0, pad_mode='pad'):
...     return nn.Conv2d(in_channels, out_channels,
...                       kernel_size=1, stride=stride, padding=padding, pad_mode=pad_mode)
...
>>> class Block(nn.Cell):
...     expansion = 4
...
...     @lazy_inline
...     def __init__(self,
...                   in_channels,
...                   out_channels,
...                   stride=1,
...                   down_sample=False):
...         super(Block, self).__init__()
```

(下页继续)

(续上页)

```

...
... out_chls = out_channels
... self.conv1 = conv1x1(in_channels, out_chls, stride=1, padding=0)
... self.bn1 = nn.BatchNorm2d(out_chls)
...
... self.conv2 = conv3x3(out_chls, out_chls, stride=stride, padding=1)
... self.bn2 = nn.BatchNorm2d(out_chls)
...
... self.conv3 = conv1x1(out_chls, out_channels, stride=1, padding=0)
... self.bn3 = nn.BatchNorm2d(out_channels)
...
... self.relu = nn.ReLU()
... self.downsample = down_sample
...
... self.conv_down_sample = conv1x1(in_channels, out_channels,
...                                 stride=stride, padding=0)
... self.bn_down_sample = nn.BatchNorm2d(out_channels)
... self.add = ops.Add()
...
... def construct(self, x):
...     identity = x
...
...     out = self.conv1(x)
...     out = self.bn1(out)
...     out = self.relu(out)
...
...     out = self.conv2(out)
...     out = self.bn2(out)
...     out = self.relu(out)
...
...     out = self.conv3(out)
...     out = self.bn3(out)
...
...     if self.downsample:
...         identity = self.conv_down_sample(identity)
...         identity = self.bn_down_sample(identity)
...
...     out = self.add(out, identity)
...     out = self.relu(out)
...
...     return out
>>> class Net(nn.Cell):
...     def __init__(self, block, num_classes=100):
...         super(Net, self).__init__()
...
...         self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, pad_mode='pad')
...         self.bn1 = nn.BatchNorm2d(64)
...         self.relu = nn.ReLU()
...         self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, pad_mode='valid')
...
...         self.layer = self.MakeLayer(
...             block, 50, in_channels=64, out_channels=2048, stride=2)
...         self.avgpool = nn.AvgPool2d(7, 1)
...         self.flatten = ops.Flatten()
...

```

(下页继续)

(续上页)

```

...     def MakeLayer(self, block, layer_num, in_channels, out_channels, stride):
...         layers = []
...         resblk = block(in_channels, out_channels,
...                         stride=stride, down_sample=True)
...         layers.append(resblk)
...
...         for _ in range(1, layer_num):
...             resblk = block(out_channels, out_channels, stride=1)
...             layers.append(resblk)
...
...         return nn.SequentialCell(layers)
...
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.bn1(x)
...         x = self.relu(x)
...         x = self.maxpool(x)
...         x = self.layer(x)
...         x = self.avgpool(x)
...         x = self.flatten(x)
...         return x
...
>>> def test_compile():
...     net = Net(Block)
...     inp = Tensor(np.ones([1, 3, 224, 224]).astype(np.float32))
...     net(inp)
...
>>> context.set_context(mode=context.GRAPH_MODE)
>>> os.environ["MS_DEV_SAVE_GRAPHS"] = "2"
>>> os.environ["MS_DEV_SAVE_GRAPHS_PATH"] = os.path.realpath("./lazy")
...
>>> test_compile()

```

1.8.8 mindspore.no_inline

`mindspore.no_inline(fn=None)`

指定 python 函数是可复用的。该函数在前端编译为可复用的子图，该子图不 `inline`。

参数：

- **fn** (function) - python 函数。如果是 cell 的方法，请参考 `mindspore.lazy_inline()`。

返回：

function，原始函数。

支持平台：

Ascend

样例：

```

>>> from mindspore import no_inline, Tensor, jit
>>> @no_inline
... def no_inline_fun(val):
...     x = val * 3 + 2
...     return x

```

(下页继续)

(续上页)

```
>>> @jit
... def call_no_inline_fun(val):
...     for _ in range(100):
...         val = no_inline_fun(val)
...     return val
>>> call_no_inline_fun(Tensor(10))
```

1.8.9 mindspore.set_recursion_limit

`mindspore.set_recursion_limit(recursion_limit=1000)`

在图编译前指定函数调用的递归深度限制。当嵌套的函数调用过深或者子图数量过多时，需要调用该接口。如果 `recursion_limit` 被设置成大于以前的值，那么系统最大栈深度也要被设置成更大，否则会因为系统栈溢出而引起一个 *core dumped* 异常。

参数：

- **recursion_limit** (int, 可选) - 递归深度限制。必须为正整数。默认值：1000。

样例：

```
>>> import mindspore as ms
>>> ms.set_recursion_limit(10000)
```

1.9 工具

1.9.1 数据处理工具

<code>mindspore.DatasetHelper</code>	<code>DatasetHelper</code> 是一个处理 <code>MindData</code> 数据集的类，提供数据集信息。
<code>mindspore.Symbol</code>	符号，用来传递张量形状的符号信息（symbolic shape）的数据结构。
<code>mindspore.connect_network_with_dataset</code>	将 <i>network</i> 与 <i>dataset_helper</i> 中的数据集连接，只支持下沉模式，(<code>dataset_sink_mode=True</code>)。
<code>mindspore.data_sink</code>	对输入的函数封装生成一个新的函数。

mindspore.DatasetHelper

class `mindspore.DatasetHelper(dataset, dataset_sink_mode=True, sink_size=-1, epoch_num=1)`

`DatasetHelper` 是一个处理 `MindData` 数据集的类，提供数据集信息。

根据不同的上下文，改变数据集的迭代，在不同的上下文中使用相同的迭代。

说明：`DatasetHelper` 的迭代将提供一个 `epoch` 的数据。

参数：

- **dataset** (Dataset) - 训练数据集迭代器。数据集可以由数据集生成器 API 在 `mindspore.dataset` 模块中生成，例如 `mindspore.dataset.ImageFolderDataset`。
- **dataset_sink_mode** (bool) - 如果值为 `True`，使用 `mindspore.ops.GetNext` 在设备 (Device) 上通过数据通道获取数据；否则在主机 (Host) 直接遍历数据集获取数据。默认值: `True`。
- **sink_size** (int) - 控制每个下沉中的数据量。必须为 -1 或正数。如果 `sink_size` 为 -1，则下沉每个 epoch 的完整数据集。如果 `sink_size` 大于 0，则下沉每个 epoch 的 `sink_size` 数据。默认值: -1。
- **epoch_num** (int) - 控制待发送的 epoch 数据量。默认值: 1。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn
>>> from mindspore import dataset as ds
>>>
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> set_helper = ms.DatasetHelper(train_dataset, dataset_sink_mode=False)
>>>
>>> net = nn.Dense(10, 5)
>>> # Object of DatasetHelper is iterable
>>> for next_element in set_helper:
...     # `next_element` includes data and label, using data to run the net
...     data = next_element[0]
...     result = net(data)
```

continue_send()

在 epoch 开始时，继续向设备发送数据。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn
>>> from mindspore import dataset as ds
>>>
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> dataset_helper = ms.DatasetHelper(train_dataset, dataset_sink_mode=True)
>>> dataset_helper.continue_send()
```

release()

释放数据下沉资源。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn
>>> from mindspore import dataset as ds
>>>
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> dataset_helper = ms.DatasetHelper(train_dataset, dataset_sink_mode=True)
>>> dataset_helper.release()
```

sink_size()

获取每次迭代的 *sink_size* 。

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>>
>>> # Define a dataset pipeline
>>> def generator():
...     for i in range(5):
...         yield (np.ones((32, 10)),)
>>>
>>> train_dataset = ms.dataset.GeneratorDataset(generator, ["data"])
>>> dataset_helper = ms.DatasetHelper(train_dataset, dataset_sink_mode=True, sink_size=-
↪1)
>>>
>>> # if sink_size==-1, then will return the full size of source dataset.
>>> sink_size = dataset_helper.sink_size()
```

stop_send()

停止发送下沉数据。

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> # Define a dataset pipeline
>>> def generator():
...     for i in range(5):
...         yield (np.ones((32, 10)),)
>>> train_dataset = ms.dataset.GeneratorDataset(generator, ["data"])
>>> dataset_helper = ms.DatasetHelper(train_dataset, dataset_sink_mode=True, sink_size=-
↪1)
>>> dataset_helper.stop_send()
```

types_shapes()

从当前配置中的数据集获取类型和形状 (shape)。

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>>
>>> # Define a dataset pipeline
>>> def generator():
...     for i in range(5):
...         yield (np.ones((32, 10)),)
>>>
>>> train_dataset = ms.dataset.GeneratorDataset(generator, ["data"])
>>> dataset_helper = ms.DatasetHelper(train_dataset, dataset_sink_mode=True)
>>>
>>> types, shapes = dataset_helper.types_shapes()
```

mindspore.Symbol

class mindspore.Symbol (*max=0, min=1, divisor=1, remainder=0, unique=False, **kwargs*)

符号，用来传递张量形状的符号信息（symbolic shape）的数据结构。

对于动态 shape 网络，相比只设置 *shape* 的未知维度（None），提供未知维度的数学符号信息能帮助框架更好地优化计算图，提高网络执行性能。

参数：

- **max** (int) - 最大值，表示此维度的最大长度。只有当它大于 *min* 值时才有效。默认值：0。
- **min** (int) - 最小值，表示此维度的最小长度，要求是正数。默认值：1。
- **divisor** (int) - 约数 *d*。默认值：1。
- **remainder** (int) - 余数 *r*。与 *divisor* 一起表示符号值为 $d * N + r, N \geq 1$ 。默认值：0。
- **unique** (bool) - 符号具有唯一值。当这个 *Symbol* 对象被 *Tensor* 多次引用时，如果 *unique* 为 True，表示用到这个 *Symbol* 对象的维度的长度都相等；否则表示只共享符号信息，不一定相等。默认值：False。

输出：

Symbol。

异常：

- **TypeError** - 如果 *max*、*min*、*divisor*、*remainder* 不是整数。
- **TypeError** - 如果 *unique* 不是布尔值。
- **ValueError** - 如果 *min* 不是正数。
- **ValueError** - 如果 *divisor* 不是正数。
- **ValueError** - 如果 *remainder* 不在区间 $[0, d)$ 内。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, Tensor, Symbol
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.abs = ms.ops.Abs()
...     def construct(self, x):
...         return self.abs(x)
...
>>> net = Net()
>>> s1 = Symbol(divisor=8, remainder=1)
>>> s2 = Symbol(max=32, unique=True)
>>> dyn_t = Tensor(shape=(None, s1, s1, s2, s2), dtype=ms.float32)
>>> net.set_inputs(dyn_t)
>>> # the shape values of last two dimensions must be equal, because "s2" is set to "unique"
>>> net(Tensor(np.random.randn(1, 9, 17, 32, 32), dtype=ms.float32)).shape
(1, 9, 17, 32, 32)
>>> net(Tensor(np.random.randn(8, 25, 9, 30, 30), dtype=ms.float32)).shape
(8, 25, 9, 30, 30)
```

mindspore.connect_network_with_dataset

`mindspore.connect_network_with_dataset(network, dataset_helper)`

将 `network` 与 `dataset_helper` 中的数据连接，只支持下沉模式，(`dataset_sink_mode=True`)。

参数：

- **network** (Cell) - 数据集的训练网络。
- **dataset_helper** (DatasetHelper) - 一个处理 MindData 数据集的类，提供了数据集的类型、形状 (shape) 和队列名称。

返回：

Cell，一个新网络，包含数据集的类型、形状 (shape) 和队列名称信息。

异常：

- **RuntimeError** - 如果该接口在非数据下沉模式调用。

支持平台：

Ascend GPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn
>>> from mindspore import dataset as ds
>>>
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> dataset_helper = ms.DatasetHelper(train_dataset, dataset_sink_mode=True)
>>> net = nn.Dense(10, 5)
>>> net_with_dataset = ms.connect_network_with_dataset(net, dataset_helper)
```

mindspore.data_sink

`mindspore.data_sink(fn, dataset, sink_size=1, jit_config=None, input_signature=None)`

对输入的函数封装生成一个新的函数。

说明：使用数据下沉时，数据集将被自动循环发送至设备，设备侧最多缓存 100 个 batch 的数据且所占内存不大于 2G。此时仅需考虑每次下沉的步数 `sink_size`，`sink_size` 默认为 1，代表每个 epoch 仅从缓存中取一个 batch 的数据进行训练并输出 loss；若 `sink_size` 大于 1，则每个 epoch 从缓存中取出 `sink_size` 个 batch 的数据进行训练然后输出 loss。

参数：

- **fn** (Function) - 将与数据集一起运行的函数。
- **dataset** (Dataset) - 训练数据集迭代器。数据集可以由数据集生成器 API 在 `mindspore.dataset` 中生成，例如 `mindspore.dataset.ImageFolderDataset`。
- **sink_size** (int) - 控制每次数据下沉的 step 数量。`sink_size` 必须为正整数。默认值：1。
- **jit_config** (JitConfig) - 编译时所使用的 JitConfig 配置项，详细可参考 `mindspore.JitConfig`。默认值：None，表示以 PyNative 模式运行。

- **input_signature** (Union[Tensor, List or Tuple of Tensors]) - 用于表示输入参数的 Tensor。Tensor 的 shape 和 dtype 将作为函数的输入 shape 和 dtype。如果指定了 *input_signature*，则 *fn* 的每个输入都必须是 Tensor，并且 *fn* 的输入参数将不会接受 ***kwargs* 参数，实际输入的 shape 和 dtype 应与 *input_signature* 相同，否则会出现 *TypeError*。默认值：None。

返回：

函数，该生成的函数会以数据下沉模式执行。

异常：

- **ValueError** - 如果 *sink_size* 不是正整数。

支持平台：

Ascend GPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import dataset as ds
>>>
>>> data = {"x": np.ones((1,)), dtype=np.int32), "y": np.ones((1,)), dtype=np.int32)}
>>> dataset = ds.NumpySlicesDataset(data=data)
>>>
>>> def func_net(x, y):
...     out = x + y
...     return out
>>>
>>> sink_process = ms.data_sink(func_net, dataset, sink_size=1)
>>> for _ in range(2):
...     out = sink_process()
...     print(out)
2
2
```

1.9.2 调试调优

mindspore.profiler.profile	MindSpore 用户能够通过该类对神经网络的性能进行采集。
mindspore.profiler._ExperimentalConfig	在使用 <i>profile</i> 进行模型性能数据采集时，配置可扩展的参数。
mindspore.profiler.mstx	该类可在 NPU 上进行性能分析的标记和跟踪。
mindspore.profiler.DynamicProfilerMonitor	该类用于动态采集 MindSpore 神经网络性能数据。
mindspore.profiler.schedule	该类用于获取每一步的操作。
mindspore.profiler.tensorboard	对动态图模式的每一个 <i>step</i> ，调用该方法进行在线解析。
_trace_handler	
mindspore.profiler.profiler.analyse	离线分析训练的性能数据，性能数据采集结束后调用。
mindspore.SummaryCollector	<i>SummaryCollector</i> 可以帮助收集一些常用信息，比如 <i>loss</i> 、学习率、计算图等。
mindspore.SummaryLandscape	<i>SummaryLandscape</i> 可以帮助您收集 <i>loss</i> 地形图的信息。
mindspore.SummaryRecord	<i>SummaryRecord</i> 用于记录 <i>summary</i> 数据和 <i>lineage</i> 数据。
mindspore.set_dump	启用或者禁用 <i>target</i> 及其子节点的 <i>Dump</i> 数据功能。

下页继续

表 14 – 续上页

<code>mindspore.Profiler</code>	当前接口即将弃用, 请使用 <code>mindspore.profiler.profile</code> 代替。

mindspore.profiler.profile

class `mindspore.profiler.profile` (*activities: list = None, with_stack: bool = False, profile_memory: bool = False, data_process: bool = False, parallel_strategy: bool = False, start_profile: bool = True, hbm_ddr: bool = False, pcie: bool = False, sync_enable: bool = True, schedule: Schedule = None, on_trace_ready: Optional[Callable[... Any]] = None, experimental_config: Optional[_ExperimentalConfig] = None*)

MindSpore 用户能够通过该类对神经网络的性能进行采集。可以通过导入该类初始化 `profile` 对象, 使用 `profile.start()` 开始分析, 并使用 `profile.stop()` 停止收集并分析结果。可通过 [MindStudio Insight](#) 工具可视化分析结果。目前, `profile` 支持 AICORE 算子、AICPU 算子、HostCPU 算子、内存、设备通信、集群等数据的分析。

参数:

- **start_profile** (bool, 可选) - 该参数控制是否在 Profiler 初始化的时候开启数据采集。默认值: `True`。
- **activities** (list, 可选) - 表示需要收集的性能数据类型。默认值: `[ProfilerActivity.CPU, ProfilerActivity.NPU]`。
 - `ProfilerActivity.CPU`: 收集 MindSpore 框架数据。
 - `ProfilerActivity.NPU`: 收集 CANN 软件栈和 NPU 数据。
 - `ProfilerActivity.GPU`: 收集 GPU 数据。
- **schedule** (schedule, 可选) - 设置采集的动作策略, 由 `schedule` 类定义, 需要配合 `step` 接口使用, 默认值: `None`, 表示采集全部 `step` 的性能数据, 详细介绍请参考[mindspore.profiler.schedule](#)。
- **on_trace_ready** (Callable, 可选) - 设置当性能数据采集完成时, 执行的回调函数。默认值: `None`, 表示只采集, 不解析性能数据, 详细介绍请参考[mindspore.profiler.tensorboard_trace_handler\(\)](#)。
- **profile_memory** (bool, 可选) - (仅限 Ascend) 表示是否收集 Tensor 内存数据。当值为 `True` 时, 收集这些数据。使用该参数时, *activities* 必须设置为 `[ProfilerActivity.CPU, ProfilerActivity.NPU]`。在图编译等级为 `O2` 时收集算子内存数据, 需要从第一个 `step` 开始采集。默认值: `False`, 该参数目前采集的算子名称不完整。将在后续版本修复, 建议使用环境变量 `MS_ALLOC_CONF` 代替。
- **with_stack** (bool, 可选) - (仅限 Ascend) 表示是否收集 Python 侧的调用栈的数据, 此数据在 `timeline` 中采用火焰图的形式呈现, 使用该参数时, *activities* 必须包含 `ProfilerActivity.CPU`。默认值: `False`。
- **hbm_ddr** (bool, 可选) - (仅限 Ascend) 是否收集片上内存/DDR 内存读写速率数据, 当值为 `True` 时, 收集这些数据。默认值: `False`。
- **pcie** (bool, 可选) - (仅限 Ascend) 是否收集 PCIe 带宽数据, 当值为 `True` 时, 收集这些数据。默认值: `False`。
- **data_process** (bool, 可选) - (Ascend/GPU) 表示是否收集数据准备性能数据, 默认值: `False`。
- **parallel_strategy** (bool, 可选) - (仅限 Ascend) 表示是否收集并行策略性能数据, 默认值: `False`。
- **sync_enable** (bool, 可选) - (仅限 GPU) Profiler 是否用同步的方式收集算子耗时, 默认值: `True`。
 - `True`: 同步方式, 在把算子发送到 GPU 之前, 在 CPU 端记录开始时间戳。然后在算子执行完毕返回到 CPU 端后, 再记录结束时间戳。算子耗时为两个时间戳的差值。
 - `False`: 异步方式, 算子耗时为从 CPU 发送到 GPU 的耗时。这种方式能减少因增加 Profiler 对整体训练时间的影响。
- **experimental_config** (_ExperimentalConfig, 可选) - 可扩展的参数可以在此配置, 默认值为 `None`, 详细介绍请参考[mindspore.profiler._ExperimentalConfig](#)。

异常:

- **RuntimeError** - 当 CANN 的版本与 MindSpore 版本不匹配时, MindSpore 无法解析生成的 ascend_job_id 目录结构。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, context
>>> import mindspore.dataset as ds
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, AicoreMetrics, ExportType
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(net):
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = mindspore.train.Model(net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     context.set_context(mode=mindspore.GRAPH_MODE)
...     mindspore.set_device("Ascend")
...
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.Level0,
...         aic_metrics=AicoreMetrics.AiCoreNone,
...         l2_cache=False,
...         mstx=False,
...         data_simplification=False,
...         export_type=[ExportType.Text])
...
...     steps = 10
...     net = Net()
...     # Note that the Profiler should be initialized before model.train
...     with mindspore.profiler.profile(activities=[ProfilerActivity.CPU, ProfilerActivity.
↪NPU],
...                                     schedule=mindspore.profiler.schedule(wait=1, ↪
↪warmup=1, active=2,
...                                     repeat=1, skip_first=2),
...                                     on_trace_ready=mindspore.profiler.
...                                         tensorboard_trace_handler("./data"),
...                                     profile_memory=False,
```

(下页继续)

(续上页)

```

...                                     experimental_config=experimental_config) as prof:
...
...     # Train Model
...     for step in range(steps):
...         train(net)
...         prof.step()

```

add_metadata (*key: str, value: str*)

上报自定义 metadata 键值对数据。

参数:

- **key** (str) - metadata 键值对的 key。
- **value** (str) - metadata 键值对的 value。

样例:

```

>>> import mindspore
>>> # Profiler init.
>>> with mindspore.profiler.profile() as prof:
...     # Call Profiler add_metadata
...     prof.add_metadata("test_key", "test_value")

```

add_metadata_json (*key: str, value: str*)

上报自定义 metadata 键值对 value 为 json 字符串数据。

参数:

- **key** (str) - metadata 键值对的 key。
- **value** (str) - metadata 键值对的 value，格式为 json 字符串。

样例:

```

>>> import json
>>> import mindspore
>>> # Profiler init.
>>> with mindspore.profiler.profile() as prof:
...     # Call Profiler add_metadata_json
...     prof.add_metadata_json("test_key", json.dumps({"key1": 1, "key2": 2}))

```

start ()

开启 profile 数据采集。可以按条件开启 profile。

异常:

- **RuntimeError** - profile 已经开启。
- **RuntimeError** - 如果 *start_profile* 参数未设置或设置为 True。

样例:

```

>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, context
>>> import mindspore.dataset as ds
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, AicoreMetrics, \
↳ ExportType
>>>

```

(下页继续)

(续上页)

```

>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(net):
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = mindspore.train.Model(net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     context.set_context(mode=mindspore.GRAPH_MODE)
...     mindspore.set_device("Ascend")
...
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.Level0,
...         aic_metrics=AicoreMetrics.AiCoreNone,
...         l2_cache=False,
...         mstx=False,
...         data_simplification=False,
...         export_type=[ExportType.Text])
...
...     steps = 10
...     net = Net()
...     # Note that the Profiler should be initialized before model.train
...     prof = mindspore.profiler.profile(activities=[ProfilerActivity.CPU,
...     ↪ProfilerActivity.NPU],
...
...                                     schedule=mindspore.profiler.schedule(wait=1,
...     ↪warmup=1, active=2,
...
...                                     repeat=1, skip_first=2),
...     on_trace_ready=mindspore.profiler.
...         tensorboard_trace_handler("./data"),
...     profile_memory=False,
...     experimental_config=experimental_config)
...
...     prof.start()
...     # Train Model
...     for step in range(steps):
...         train(net)
...         prof.step()
...     prof.stop()

```

step()

用于在 Ascend 设备上，通过 schedule 和 on_trace_ready 区分步骤收集和解析性能数据。

异常：

- **RuntimeError** - 如果 start_profile 参数未设置或 profile 未开启。
- **RuntimeError** - 如果 schedule 参数未设置。

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, context
>>> import mindspore.dataset as ds
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, AicoreMetrics, \
↳ ExportType
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(net):
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = mindspore.train.Model(net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     context.set_context(mode=mindspore.GRAPH_MODE)
...     mindspore.set_device("Ascend")
...
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.Level0,
...         aic_metrics=AicoreMetrics.AiCoreNone,
...         l2_cache=False,
...         mstx=False,
...         data_simplification=False,
...         export_type=[ExportType.Text])
...
...     steps = 10
...     net = Net()
...     # Note that the Profiler should be initialized before model.train
...     with mindspore.profiler.profile(activities=[ProfilerActivity.CPU, \
↳ ProfilerActivity.NPU],
...                                     schedule=mindspore.profiler.schedule(wait=1, \
↳ warmup=1, active=2,
...                                     repeat=1, skip_first=2),
...                                     on_trace_ready=mindspore.profiler.tensorboard_
↳ trace_handler("./data"),
...                                     profile_memory=False,
...                                     experimental_config=experimental_config) as prof:
...
...         # Train Model
...         for step in range(steps):
...             train(net)
...             prof.step()
```

`stop()`

停止 profile。可以按条件停止 profile。

异常:

- **RuntimeError** - profile 没有开启。

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, context
>>> import mindspore.dataset as ds
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, AicoreMetrics, \
↳ ExportType
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(net):
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = mindspore.train.Model(net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     context.set_context(mode=mindspore.GRAPH_MODE)
...     mindspore.set_device("Ascend")
...
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.Level0,
...         aic_metrics=AicoreMetrics.AiCoreNone,
...         l2_cache=False,
...         mstx=False,
...         data_simplification=False,
...         export_type=[ExportType.Text])
...
...     steps = 10
...     net = Net()
...     # Note that the Profiler should be initialized before model.train
...     prof = mindspore.profiler.profile(activities=[ProfilerActivity.CPU, \
↳ ProfilerActivity.NPU],
...
...                                     schedule=mindspore.profiler.schedule(wait=1, \
↳ warmup=1, active=2,
...
...                                     repeat=1, skip_first=2),
...     on_trace_ready=mindspore.profiler.
...         tensorboard_trace_handler("./data"),
...     profile_memory=False,
...     experimental_config=experimental_config)
```

(下页继续)

(续上页)

```

...     prof.start()
...     # Train Model
...     for step in range(steps):
...         train(net)
...         prof.step()
...     prof.stop()

```

mindspore.profiler._ExperimentalConfig

```

class mindspore.profiler._ExperimentalConfig (profiler_level: ProfilerLevel = ProfilerLevel.Level0, aic_metrics:
    AicoreMetrics = AicoreMetrics.AiCoreNone, l2_cache: bool =
    False, mstx: bool = False, data_simplification: bool = True,
    export_type: list = None)

```

在使用 `profile` 进行模型性能数据采集时，配置可扩展的参数。

参数：

- **profiler_level** (ProfilerLevel, 可选) - (仅限 Ascend) 表示采集性能数据级别。默认值: ProfilerLevel.Level0。
 - ProfilerLevel.LevelNone: 该设置仅在开启 `mstx` 时生效，表示不采集 `device` 侧的任何算子数据。
 - ProfilerLevel.Level0: 最精简的采集性能数据级别，采集计算类算子的耗时数据和通信类大算子的基础数据。
 - ProfilerLevel.Level1: 在 Level0 的基础上额外采集 CANN 层中 AscendCL 数据、AICORE 性能数据以及通信类小算子数据。
 - ProfilerLevel.Level2: 在 Level1 的基础上额外采集 CANN 层中 GE 和 Runtime 数据。
- **aic_metrics** (AicoreMetrics, 可选) - (仅限 Ascend) 收集的 AICORE 性能数据类型，使用此参数时，`activities` 必须包含 ProfilerActivity.NPU，且值必须包含在 AicoreMetrics 枚举值中，当 `profiler_level` 为 Level0，默认值为: AicoreMetrics.AiCoreNone；`profiler_level` 为 Level1 或 Level2，默认值为: AicoreMetrics.PipeUtilization，当每种类型包含的数据项如下：
 - AicoreMetrics.AiCoreNone: 不收集任何 AICORE 数据。
 - AicoreMetrics.ArithmeticUtilization: 包含 `mac_fp16/int8_ratio`、`vec_fp32/fp16/int32_ratio`、`vec_misc_ratio` 等。
 - AicoreMetrics.PipeUtilization: 包含 `vec_ratio`、`mac_ratio`、`scalar_ratio`、`mte1/mte2/mte3_ratio`、`icache_miss_rate` 等。
 - AicoreMetrics.Memory: 包含 `ub_read/write_bw`、`l1_read/write_bw`、`l2_read/write_bw`、`main_mem_read/write_bw` 等。
 - AicoreMetrics.MemoryL0: 包含 `l0a_read/write_bw`、`l0b_read/write_bw`、`l0c_read/write_bw` 等。
 - AicoreMetrics.ResourceConflictRatio: 包含 `vec_bankgroup/bank/resc_cflt_ratio` 等。
 - AicoreMetrics.MemoryUB: 包含 `ub_read/write_bw_mte`、`ub_read/write_bw_vector`、`ub_read/write_bw_scalar` 等。
 - AicoreMetrics.L2Cache: 包含 `write_cache_hit`、`write_cache_miss_allocate`、`r0_read_cache_hit`、`r1_read_cache_hit` 等。本功能仅支持 Atlas A2 训练系列产品。
 - AicoreMetrics.MemoryAccess: 主存以及 L2 Cache 的存访带宽和存量统计。
- **l2_cache** (bool, 可选) - (仅限 Ascend) 是否收集 L2 Cache 数据，当值为 `True` 时，收集这些数据。默认值: `False`。该采集项在 `ASCEND_PROFILER_OUTPUT` 文件夹下生成 `l2_cache.csv` 文件。在 O2 模式下，仅支持 `schedule` 配置中 `wait` 和 `skip_first` 参数都为 0 的场景。

- **mstx** (bool, 可选) - (仅限 Ascend) 是否收集 MSTX 数据, 当值为 True 时, 收集这些数据。默认值: False。
- **data_simplification** (bool, 可选) - (仅限 Ascend) 是否开启数据精简, 开启后仅保留 profiler 的交付件以及 PROF_XXX 目录下的原始性能数据, 以节省空间。默认值: True。
- **export_type** (list, 可选) - (仅限 Ascend) 要导出的数据类型, 支持同时导出 db 和 text 格式, 默认值: None, 表示导出 text 类型数据。
 - ExportType.Text: 导出 text 类型的数据。
 - ExportType.Db: 导出 db 类型的数据。

异常:

- **RuntimeError** - 当 CANN 的版本与 MindSpore 版本不匹配时, MindSpore 无法解析生成的 ascend_job_id 目录结构。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, context
>>> import mindspore.dataset as ds
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, AicoreMetrics, ExportType
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(net):
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = mindspore.train.Model(net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     context.set_context(mode=mindspore.GRAPH_MODE)
...     mindspore.set_device("Ascend")
...
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.Level0,
...         aic_metrics=AicoreMetrics.AiCoreNone,
...         l2_cache=False,
...         mstx=False,
...         data_simplification=False,
...         export_type=[ExportType.Text])
```

(下页继续)

(续上页)

```

...     steps = 10
...     net = Net()
...     # Note that the Profiler should be initialized before model.train
...     with mindspore.profiler.profile(activities=[ProfilerActivity.CPU, ProfilerActivity.
↪NPU],
...                                     schedule=mindspore.profiler.schedule(wait=1, ↪
↪warmup=1, active=2,
...                                     repeat=1, skip_first=2),
...                                     on_trace_ready=mindspore.profiler.tensorboard_trace_
↪handler("./data"),
...                                     profile_memory=False,
...                                     experimental_config=experimental_config) as prof:
...
...         # Train Model
...         for step in range(steps):
...             train(net)
...             prof.step()

```

mindspore.profiler.mstx

class mindspore.profiler.mstx

该类可在 NPU 上进行性能分析的标记和跟踪。该类提供了 mark、range_start 和 range_end 三个静态方法，用于在性能分析中添加标记点和区间。

static mark (message: str, stream: mindspore.runtime.Stream = None)

在性能分析中添加标记点。

参数：

- **message** (str) - 标记点的描述信息。
- **stream** (Stream, 可选) - 用于异步执行的 NPU 流，期望类型：Stream。默认值：None，表示仅在 host 侧添加标记点，不在 device 侧的 stream 上添加标记点。

样例：

```

>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore
>>> from mindspore import nn
>>> import mindspore.dataset as ds
>>> from mindspore import Profiler
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, schedule, ↪
↪tensorboard_trace_handler
>>> from mindspore.profiler import mstx
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield (np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32))

```

(下页继续)

(续上页)

```

>>>
>>> def train(net):
...     stream = ms.runtime.current_stream()
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = ms.train.Model(net, loss, optimizer)
...     # Add marker before training
...     mstx.mark("train start", stream)
...     model.train(1, data)
...     # Add marker after training
...     mstx.mark("train end", stream)
>>>
>>> if __name__ == '__main__':
...     # Note: mstx only supports Ascend device and cannot be used in mindspore.nn.Cell.
...     ↪construct
...     # when in mindspore.GRAPH_MODE
...     ms.set_context(mode=ms.PYNATIVE_MODE)
...     ms.set_device(device_target="Ascend", device_id=0)
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.LevelNone,
...         mstx=True)
...     # Note that the Profiler should be initialized before model.train
...     with mindspore.profiler.profile(
...         activities=[ProfilerActivity.CPU, ProfilerActivity.NPU],
...         schedule=schedule(wait=0, warmup=0, active=3, repeat=1, skip_first=0),
...         on_trace_ready=mindspore.profiler.tensorboard_trace_handler("./data"),
...         experimental_config=experimental_config
...     ) as profiler:
...         net = Net()
...         for i in range(5):
...             train(net)
...             profiler.step()

```

static range_end (range_id: int)

结束一个性能分析区间。

参数:

- **range_id** (int) - 从 range_start 返回的区间 ID。

样例:

```

>>> # Please refer to the example in range_start
>>> # range_id = mstx.range_start("training process", stream)
>>> # model.train(1, data)
>>> # mstx.range_end(range_id)

```

static range_start (message: str, stream: mindspore.runtime.Stream = None)

开始一个性能分析区间。

参数:

- **message** (str) - 区间的描述信息。
- **stream** (*Stream*, 可选) - 用于异步执行的 NPU 流, 期望类型: *Stream*。默认值: None, 表示仅在 host 侧开始区间打点, 不在 device 侧的 stream 上开始区间打点。

返回:

int, 区间 ID, 用于 range_end 方法。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore
>>> from mindspore import nn
>>> import mindspore.dataset as ds
>>> from mindspore import Profiler
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, schedule,
↳tensorboard_trace_handler
>>> from mindspore.profiler import mstx
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield (np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32))
>>>
>>> def train(net):
...     stream = ms.runtime.current_stream()
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = ms.train.Model(net, loss, optimizer)
...     # Start profiling range
...     range_id = mstx.range_start("training process", stream)
...     model.train(1, data)
...     # End profiling range
...     mstx.range_end(range_id)
>>>
>>> if __name__ == '__main__':
...     # Note: mstx only supports Ascend device and cannot be used in mindspore.nn.Cell.
↳construct
...     # when in mindspore.GRAPH_MODE
...     ms.set_context(mode=ms.PYNATIVE_MODE)
...     ms.set_device(device_target="Ascend", device_id=0)
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.LevelNone,
...         mstx=True)
...     # Note that the Profiler should be initialized before model.train
...     with mindspore.profiler.profile(
...         activities=[ProfilerActivity.CPU, ProfilerActivity.NPU],
...         schedule=schedule(wait=0, warmup=0, active=3, repeat=1, skip_first=0),
...         on_trace_ready=mindspore.profiler.tensorboard_trace_handler("./data"),
...         experimental_config=experimental_config
...     ) as profiler:
...         net = Net()
...         for i in range(5):
...             train(net)
```

(下页继续)

(续上页)

```
... profiler.step()
```

mindspore.profiler.DynamicProfilerMonitor

```
class mindspore.profiler.DynamicProfilerMonitor (cfg_path, output_path='./dyn_profile_data', poll_interval=2,
**kwargs)
```

该类用于动态采集 MindSpore 神经网络性能数据。

参数：

- **cfg_path** (str) - (仅限 Ascend) 动态 Profiler 的 json 配置文件的文件夹路径。要求该路径是能够被所有节点访问到的共享目录。json 配置文件相关参数如下。
 - **start_step** (int, 必选) - 设置 Profiler 开始采集的步数，为相对值，训练的第一步为 1。默认值-1，表示在整个训练流程不会开始采集。
 - **stop_step** (int, 必选) - 设置 Profiler 开始停止的步数，为相对值，训练的第一步为 1，需要满足 stop_step 大于等于 start_step。默认值-1，表示在整个训练流程不会开始采集。
 - **aic_metrics** (int, 可选) - 设置采集 AI Core 指标数据，取值范围与 Profiler 一一对应。默认值-1，表示不采集 AI Core 指标，0 代表 PipeUtilization；1 代表 ArithmeticUtilization；2 代表 Memory；3 代表 MemoryL0；4 代表 MemoryUB；5 代表 ResourceConflictRatio；6 代表 L2Cache；7 代表 MemoryAccess。
 - **profiler_level** (int, 可选) - 设置采集性能数据级别，-1 代表 ProfilerLevel.LevelNone，0 代表 ProfilerLevel.Level0，1 代表 ProfilerLevel.Level1，2 代表 ProfilerLevel.Level2。默认值 0，表示 ProfilerLevel.Level0 的采集级别。
 - **activities** (int, 可选) - 设置采集性能数据的设备，0 代表 CPU+NPU，1 代表 CPU，2 代表 NPU。默认值 0，表示采集 CPU+NPU 的性能数据。
 - **export_type** (int, 可选) - 设置导出性能数据的类型，0 代表 text，1 代表 db，2 代表 text 和 db。默认值 0，表示只导出 text 类型的性能数据。
 - **profile_memory** (bool, 可选) - 设置是否采集内存性能数据，true 代表采集，false 代表不采集。默认值 false，表示不采集内存性能数据。
 - **mstx** (bool, 可选) - 设置是否开启 mstx，true 代表开启，false 代表不开启。默认值 false，表示不开启 mstx。
 - **analyse_mode** (int, 可选) - 设置在线解析的模式，对应 mindspore.Profiler.analyse 接口的 analyse_mode 参数，0 代表"sync"，1 代表"async"。默认值-1，表示不使用在线解析。
 - **parallel_strategy** (bool, 可选) - 设置是否采集并行策略性能数据，true 代表采集，false 代表不采集。默认值 false，表示不采集并行策略性能数据。
 - **with_stack** (bool, 可选) - 设置是否采集调用栈信息，true 代表采集，false 代表不采集。默认值 false，表示不采集调用栈。
 - **data_simplification** (bool, 可选) - 设置开启数据精简，true 代表开启，false 代表不开启。默认值 true，表示开启数据精简。
- **output_path** (str, 可选) - (仅限 Ascend) 动态 Profiler 的输出文件路径。默认值："./dyn_profile_data"。
- **poll_interval** (int, 可选) - (仅限 Ascend) 监控进程的轮询周期，单位为秒。默认值：2。

异常：

- **RuntimeError** - 创建监控进程失败次数超过最大限制。

支持平台：

Ascend GPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn
>>> import mindspore.dataset as ds
>>> from mindspore.profiler import DynamicProfilerMonitor
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield (np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32))
>>>
>>> def train(net):
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     dynprof_cb = DynamicProfilerMonitor(cfg_path="./dyn_cfg", output_path="./dyn_prof_
↳data")
...     model = ms.train.Model(net, loss, optimizer)
...     # register DynamicProfilerMonitor to model.train()
...     model.train(10, data, callbacks=[dynprof_cb])
```

step()

用于在 Ascend 设备上，区分 step 收集和解析性能数据。

异常:

- **RuntimeError** - 如果 *start_step* 参数设置大于 *stop_step* 参数设置。

样例:

```
>>> import json
>>> import os
>>> import numpy as np
>>>
>>> import mindspore
>>> import mindspore.dataset as ds
>>> from mindspore import context, nn
>>> from mindspore.profiler import DynamicProfilerMonitor
>>>
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator_net():
```

(下页继续)

(续上页)

```

...     for _ in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(test_net):
...     optimizer = nn.Momentum(test_net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator_net(), ["data", "label"])
...     model = mindspore.train.Model(test_net, loss, optimizer)
...     model.train(1, data)
>>>
>>> def change_cfg_json(json_path):
...     with open(json_path, 'r', encoding='utf-8') as file:
...         data = json.load(file)
...
...     data['start_step'] = 6
...     data['stop_step'] = 7
...
...     with open(json_path, 'w', encoding='utf-8') as file:
...         json.dump(data, file, ensure_ascii=False, indent=4)
>>>
>>> if __name__ == '__main__':
...     # set json configuration file
...     context.set_context(mode=mindspore.PYNATIVE_MODE)
...     mindspore.set_device("Ascend")
...     data_cfg = {
...         "start_step": 2,
...         "stop_step": 5,
...         "aic_metrics": -1,
...         "profiler_level": 0,
...         "activities": 0,
...         "export_type": 0,
...         "profile_memory": False,
...         "mstx": False,
...         "analyse_mode": 0,
...         "parallel_strategy": False,
...         "with_stack": False,
...         "data_simplification": True,
...     }
...     output_path = "./cfg_path"
...     cfg_path = os.path.join(output_path, "profiler_config.json")
...     os.makedirs(output_path, exist_ok=True)
...     # set cfg file
...     with open(cfg_path, 'w') as f:
...         json.dump(data_cfg, f, indent=4)
...     # cfg_path contains the json configuration file path, and output_path is the_
↪output path
...     dp = DynamicProfilerMonitor(cfg_path=output_path, output_path=output_path)
...     STEP_NUM = 15
...     # Define a network of training models
...     net = Net()
...     for i in range(STEP_NUM):
...         print(f"step {i}")
...         train(net)
...         # Modify the configuration file after step 7. For example, change start_
↪step to 8 and stop_step to 10
...         if i == 5:

```

(下页继续)

(续上页)

```

...         # Modify parameters in the JSON file
...         change_cfg_json(os.path.join(output_path, "profiler_config.json"))
...         # Call step collection
...         dp.step()

```

mindspore.profiler.schedule

class mindspore.profiler.schedule (*, wait: int, active: int, warmup: int = 0, repeat: int = 0, skip_first: int = 0)

该类用于获取每一步的操作。

调度如下：

```

(NONE)      (NONE)      (NONE)      (WARM_UP)      (RECORD)      (RECORD)      _
↪(RECORD_AND_SAVE)      None
START----->skip_first----->wait----->warmup----->active.....active.....
↪active----->stop
                                     |
                                     |
                                     repeat_1
                                     |
-----

```

Profiler 将跳过前 skip_first 步，然后等待 wait 步，接着在接下来的 warmup 步中进行预热，然后在接下来的 active 步中进行活动记录，然后从 wait 步开始重复循环。可选的循环次数由 repeat 参数指定，repeat 值为 0 表示循环将继续直到分析完成。

关键字参数：

- **wait** (int) - 预热阶段等待的步数，必须大于等于 0。如果外部不设置 wait 参数，会在初始化 schedule 类时，设置为 0。
- **active** (int) - 活动阶段执行的步数，必须大于等于 1。如果外部不设置 active 参数，会在初始化 schedule 类时，设置为 1。
- **warmup** (int, 可选) - 预热阶段执行的步数，必须大于等于 0。默认值：0。
- **repeat** (int, 可选) - 重复次数，必须大于等于 0。如果 repeat 设置为 0，Profiler 会根据模型训练次数来确定 repeat 值，此时会多生成一个采集不完整的性能数据，最后一个 step 的数据用户无需关注，为异常数据。默认值：0。
- **skip_first** (int, 可选) - 跳过第一个步数。默认值：0。

异常：

- **ValueError** - 参数 step 必须大于等于 0。

支持平台：

Ascend

样例：

```

>>> import numpy as np
>>> import mindspore
>>> import mindspore.dataset as ds
>>> from mindspore import context, nn
>>> from mindspore.profiler import ProfilerLevel, AicoreMetrics, ExportType, ProfilerActivity
>>>
>>> class Net(nn.Cell):
...     def __init__(self):

```

(下页继续)

(续上页)

```

...     super(Net, self).__init__()
...     self.fc = nn.Dense(2, 2)
...
...     def construct(self, x):
...         return self.fc(x)
...
>>>
>>> def generator_net():
...     for _ in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
...
>>> def train(test_net):
...     optimizer = nn.Momentum(test_net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator_net(), ["data", "label"])
...     model = mindspore.train.Model(test_net, loss, optimizer)
...     model.train(1, data)
...
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     context.set_context(mode=mindspore.GRAPH_MODE)
...     mindspore.set_device("Ascend")
...
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.Level0,
...         aic_metrics=AicoreMetrics.AiCoreNone,
...         l2_cache=False,
...         mstx=False,
...         data_simplification=False,
...         export_type=[ExportType.Text])
...
...     steps = 10
...     net = Net()
...     # Note that the Profiler should be initialized before model.train
...     with mindspore.profiler.profile(activities=[ProfilerActivity.CPU, ProfilerActivity.
... ↪NPU],
...                                     schedule=mindspore.profiler.schedule(wait=1, ↪
... ↪warmup=1, active=2,
...                                     repeat=1, skip_first=2),
...                                     on_trace_ready=mindspore.profiler.tensorboard_trace_
... ↪handler("./data"),
...                                     profile_memory=False,
...                                     experimental_config=experimental_config) as prof:
...
...         # Train Model
...         for step in range(steps):
...             train(net)
...             prof.step()

```

to_dict()

将 schedule 类转换为一个字典。

返回:

字典, schedule 类的参数与对应的值。

mindspore.profiler.tensorboard_trace_handler

`mindspore.profiler.tensorboard_trace_handler` (*dir_name: str = None, worker_name: str = None, analyse_flag: bool = True, async_mode: bool = False*)

对动态图模式的每一个 step，调用该方法进行在线解析。

参数：

- **dir_name** (str, 可选) - 指定保存分析结果的目录路径。默认为：None，表示使用默认的保存路径，默认路径为：“./data”。
- **worker_name** (str, 可选) - 指定工程线程名称。默认为：None，表示使用默认的工程线程名，默认工程线程名为：“当前操作系统名 + 进程号”。
- **analyse_flag** (bool, 可选) - 是否使用在线分析。默认为：True，表示使用在线分析。
- **async_mode** (bool, 可选) - 是否使用异步解析模式。默认值：False，表示使用同步解析模式。

样例：

```
>>> import numpy as np
>>> import mindspore
>>> import mindspore.dataset as ds
>>> from mindspore import context, nn
>>> from mindspore.profiler import ProfilerLevel, AicoreMetrics, ExportType, ProfilerActivity
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator_net():
...     for _ in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(test_net):
...     optimizer = nn.Momentum(test_net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator_net(), ["data", "label"])
...     model = mindspore.train.Model(test_net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     context.set_context(mode=mindspore.GRAPH_MODE)
...     mindspore.set_device("Ascend")
...
...     # Init Profiler
...     experimental_config = mindspore.profiler._ExperimentalConfig(
...         profiler_level=ProfilerLevel.Level0,
...         aic_metrics=AicoreMetrics.AiCoreNone,
...         l2_cache=False,
...         mstx=False,
...         data_simplification=False,
...         export_type=[ExportType.Text])
...
...     steps = 10
```

(下页继续)

(续上页)

```

...     net = Net()
...     # Note that the Profiler should be initialized before model.train
...     with mindspore.profiler.profile(activities=[ProfilerActivity.CPU, ProfilerActivity.
...         ↪NPU],
...                                     schedule=mindspore.profiler.schedule(wait=1, ↪
...         ↪warmup=1, active=2,
...                                     repeat=1, skip_first=2),
...                                     on_trace_ready=mindspore.profiler.tensorboard_trace_
...         ↪handler("./data"),
...                                     profile_memory=False,
...                                     experimental_config=experimental_config) as prof:
...
...         # Train Model
...         for step in range(steps):
...             train(net)
...             prof.step()

```

mindspore.profiler.profiler.analyse

`mindspore.profiler.profiler.analyse` (*profiler_path*: str, *max_process_number*: int = `os.cpu_count() // 2`, *pretty*=False, *step_list*=None, *data_simplification*=True)

离线分析训练的性能数据，性能数据采集结束后调用。

参数：

- **profiler_path** (str) - 需要进行离线分析的 profiling 数据路径，指定到 *_ascend_ms 上层目录。
- **max_process_number** (int, 可选) - 最大进程数，默认值：`os.cpu_count() // 2`。
- **pretty** (bool, 可选) - 对 json 文件进行格式化处理。默认值：False，即不进行格式化。
- **step_list** (list, 可选) - 只解析指定 step 的性能数据，指定的 step 必须是连续的整数。仅支持 GRAPH 模式下 CallBack 方式采集，只能切分 CANN 层及以下信息。默认值：None，即进行全解析。
- **data_simplification** (bool, 可选) - 数据精简开关功能。默认值：True，即开启数据精简。

样例：

```

>>> from mindspore.profiler.profiler import analyse
>>> analyse(profiler_path="./profiling_path")

```

mindspore.SummaryCollector

class `mindspore.SummaryCollector` (*summary_dir*, *collect_freq*=10, *num_process*=32, *collect_specified_data*=None, *keep_default_action*=True, *custom_lineage_data*=None, *collect_tensor_freq*=None, *max_file_size*=None, *export_options*=None)

SummaryCollector 可以帮助收集一些常用信息，比如 loss、学习率、计算图等。

说明：

1. 不允许在回调列表中存在多个 SummaryCollector 实例。
2. 并非所有信息都可以在训练阶段或评估阶段收集。
3. SummaryCollector 始终记录 summary 算子收集的数据。
4. SummaryCollector 仅支持 Linux 系统。

5. 编译 MindSpore 时, 设置 `-s on` 关闭维测功能后, `SummaryCollector` 不可用。

参数:

- **summary_dir** (str) - 收集的数据将存储到此目录。如果目录不存在, 将自动创建。
- **collect_freq** (int) - 设置数据收集的频率, 频率应大于零, 单位为 *step*。如果设置了频率, 将在 $(\text{current steps} \% \text{freq})=0$ 时收集数据, 并且将总是收集第一个 *step*。需要注意的是, 如果使用数据下沉模式, 单位将变成 *epoch*。不建议过于频繁地收集数据, 因为这可能会影响性能。默认值: 10。
- **num_process** (int) - 设置保存 Summary 数据的进程数, 进程越多性能越好, 设置进程多可能会出现 Host 内存不足的问题。默认值: 32。
- **collect_specified_data** (Union[None, dict]) - 对收集的数据进行自定义操作。您可以使用字典自定义需要收集的数据类型。例如, 您可以设置 `{'collect_metric': False}` 不去收集 *metrics*。支持控制的数据如下。默认值: None, 收集所有数据。
 - **collect_metric** (bool) - 表示是否收集训练 *metrics*, 目前只收集 *loss*。把第一个输出视为 *loss*, 并且算出其平均数。默认值: True。
 - **collect_graph** (bool) - 表示是否收集计算图。目前只收集训练计算图。默认值: True。
 - **collect_train_lineage** (bool) - 表示是否收集训练阶段的 *lineage* 数据。默认值: True。
 - **collect_eval_lineage** (bool) - 表示是否收集评估阶段的 *lineage* 数据。默认值: True。
 - **collect_input_data** (bool) - 表示是否为每次训练收集数据集。目前仅支持图像数据。如果数据集中有多列数据, 则第一列应为图像数据。默认值: True。
 - **collect_dataset_graph** (bool) - 表示是否收集训练阶段的数据集图。默认值: True。
 - **histogram_regular** (Union[str, None]) - 收集参数分布页面的权重和偏置。此字段允许正则表达式控制要收集的参数。不建议一次收集太多参数, 因为这会影响性能。注: 如果收集的参数太多并且内存不足, 训练将会失败。默认值: None, 表示只收集网络的前五个超参。
 - **collect_landscape** (Union[dict, None]) - 表示是否收集创建 *loss* 地形图所需要的参数。如果设置 None, 则不收集任何参数。默认收集所有参数并且将会保存在 `{summary_dir}/ckpt_dir/train_metadata.json` 文件中。
 - * **landscape_size** (int) - 指定生成 *loss* 地形图的图像分辨率。例如: 如果设置为 128, 则 *loss* 地形图的分辨率是 128*128。注意: 计算 *loss* 地形图的时间随着分辨率的增大而增加。默认值: 40。可选值: 3-256。
 - * **unit** (str) - 指定训练过程中保存 *checkpoint* 时, 下方参数 *intervals* 以何种形式收集模型权重。例如: 将 *intervals* 设置为 `[[1, 2, 3, 4]]`, 如果 *unit* 设置为 *step*, 则收集模型权重的频率单位为 *step*, 将保存 1-4 个 *step* 的模型权重, 而 *unit* 设置为 *epoch*, 则将保存 1-4 个 *epoch* 的模型权重。默认值: *step*。可选值: *epoch/step*。
 - * **create_landscape** (dict) - 选择创建哪种类型的 *loss* 地形图, 分为训练过程 *loss* 地形图 (*train*) 和训练结果 *loss* 地形图 (*result*)。默认值: `{"train": True, "result": True}`。可选值: True/False。
 - * **num_samples** (int) - 创建 *loss* 地形图所使用的数据集的大小。例如: 在图像数据集中, 您可以设置 *num_samples* 是 128, 这意味着将有 128 张图片被用来创建 *loss* 地形图。注意: *num_samples* 越大, 计算 *loss* 地形图时间越长。默认值: 128。
 - * **intervals** (List[List[int]]) - 指定 *loss* 地形图的区间。例如: 如果用户想要创建两张训练过程的 *loss* 地形图, 分别为 1-5epoch 和 6-10epoch, 则用户可以设置 `[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]]`。注意: 每个区间至少包含 3 个 *epoch*。
- **keep_default_action** (bool) - 此字段影响 *collect_specified_data* 字段的收集行为。True: 表示设置指定数据后, 其他数据按默认设置收集。False: 表示设置指定数据后, 只收集指定数据, 不收集其他数据。默认值: True。

- **custom_lineage_data** (Union[dict, None]) - 允许您自定义数据。在自定义数据中, key 支持 str 类型, value 支持 str、int 和 float 类型。默认值: None, 表示不存在自定义数据。
- **collect_tensor_freq** (Optional[int]) - 语义与 *collect_freq* 的相同, 但仅控制 TensorSummary。由于 TensorSummary 数据太大, 无法与其他 summary 数据进行比较, 因此此参数用于降低收集量。默认情况下, 收集 TensorSummary 数据的最大 step 数量为 20, 但不会超过收集其他 summary 数据的 step 数量。例如, 给定 *collect_freq=10*, 当总 step 数量为 600 时, TensorSummary 将收集 20 个 step, 而收集其他 summary 数据时会收集 61 个 step。但当总 step 数量为 20 时, TensorSummary 和其他 summary 将收集 3 个 step。另外请注意, 在并行模式下, 会平均分配总的 step 数量, 这会影响 TensorSummary 收集的 step 的数量。默认值: None, 表示要遵循上述规则。
- **max_file_size** (Optional[int]) - 可写入磁盘的每个文件的最大大小 (以字节为单位)。例如, 如果不大于 4GB, 则设置 *max_file_size=4*1024*3*。默认值: None, 表示无限制。
- **export_options** (Union[None, dict]) - 表示对导出的数据执行自定义操作。注: 导出文件的大小不受 *max_file_size* 的限制。您可以使用字典自定义导出的数据。例如, 您可以设置 {'tensor_format': 'numpy'} 将 tensor 导出为 numpy 文件。支持控制的数据如下所示。默认值: None, 表示不导出数据。
 - **tensor_format** (Union[str, None]) - 自定义导出的 tensor 的格式。支持 ['numpy', None]。默认值: None, 表示不导出 tensor。
 - * **numpy** - 将 tensor 导出为 NPY 文件。

异常:

- **ValueError** - 编译 MindSpore 时, 设置 *-s on* 关闭了维测功能。

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn, SummaryCollector
>>> from mindspore.train import Model, Accuracy
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     ms.set_context(mode=ms.GRAPH_MODE, device_target="Ascend")
...     mnist_dataset_dir = '/path/to/mnist_dataset_directory'
...     # Create the dataset taking MNIST as an example. Refer to
...     # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
...     ds_train = create_dataset()
...     # Define the network structure of LeNet5. Refer to
...     # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
...     network = LeNet5(10)
...     net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
...     net_opt = nn.Momentum(network.trainable_params(), 0.01, 0.9)
...     model = Model(network, net_loss, net_opt, metrics={"Accuracy": Accuracy()}, amp_
->level="O2")
...
...     # Simple usage:
...     summary_collector = SummaryCollector(summary_dir='./summary_dir')
...     model.train(1, ds_train, callbacks=[summary_collector], dataset_sink_mode=False)
...
...     # Do not collect metric and collect the first layer parameter, others are collected
->by default
...     specified={'collect_metric': False, 'histogram_regular': '^conv1.*'}
...     summary_collector = SummaryCollector(summary_dir='./summary_dir', collect_specified_
->data=specified)
...     model.train(1, ds_train, callbacks=[summary_collector], dataset_sink_mode=False)
```

mindspore.SummaryLandscape

class mindspore.SummaryLandscape(summary_dir)

SummaryLandscape 可以帮助您收集 loss 地形图的信息。通过计算 loss，可以在 PCA（Principal Component Analysis）方向或者随机方向创建地形图。

说明：SummaryLandscape 仅支持 Linux 系统。

参数：

- **summary_dir** (str) - 该路径将被用来保存创建地形图所使用的数据。

样例：

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore.train import Model, Accuracy, Loss
>>> from mindspore import SummaryCollector, SummaryLandscape
>>>
>>> if __name__ == '__main__':
...     # If the device_target is Ascend, set the device_target to "Ascend"
...     ms.set_context(mode=ms.GRAPH_MODE, device_target="GPU")
...     # Create the dataset taking MNIST as an example. Refer to
...     # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
...     ds_train = create_dataset()
...     # Define the network structure of LeNet5. Refer to
...     # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
...     network = LeNet5()
...     net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
...     net_opt = nn.Momentum(network.trainable_params(), 0.01, 0.9)
...     model = Model(network, net_loss, net_opt, metrics={"Accuracy": Accuracy()})
...     # Simple usage for collect landscape information:
...     interval_1 = [1, 2, 3, 4, 5]
...     summary_collector = SummaryCollector(summary_dir='./summary/lenet_interval_1',
...                                           collect_specified_data={'collect_landscape': {
...↪ "landscape_size": 4,
...↪ "unit": "step",
...↪ "create_
...↪ "landscape": {"train": True,
...↪ "result": False},
...↪ "num_samples": 2048,
...↪ "intervals": interval_1}}
...     model.train(1, ds_train, callbacks=[summary_collector], dataset_sink_mode=False)
...     # Simple usage for visualization landscape:
...     def callback_fn():
...         # Define the network structure of LeNet5. Refer to
...         # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
...         network = LeNet5()
...         net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
...         metrics = {"Loss": Loss() }
```

(下页继续)

(续上页)

```

...     model = Model(network, net_loss, metrics=metrics)
...     # Create the dataset taking MNIST as an example. Refer to
...     # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
...     ds_eval = create_dataset()
...     return model, network, ds_eval, metrics
...
...     summary_landscape = SummaryLandscape('./summary/lenet_interval_1')
...     # parameters of collect_landscape can be modified or unchanged
...     summary_landscape.gen_landscapes_with_multi_process(callback_fn,
...                                                         collect_landscape={"landscape_size
... ↪": 4,
...                                                         "create_landscape
... ↪":{"train":False,
...                                                         "result":False},
...                                                         "num_samples":↪
... ↪2048,
...                                                         "intervals":↪
... ↪[interval_1]},
...                                                         device_ids=[1])

```

clean_ckpt()

清理 checkpoint。

gen_landscapes_with_multi_process (callback_fn, collect_landscape=None, device_ids=None, output=None)

使用多进程来生成地形图。

参数：

- **callback_fn** (python function) - Python 函数对象，用户需要写一个没有输入的函数，返回值要求如下。
 - mindspore.train.Model：用户的模型。
 - mindspore.nn.Cell：用户的网络。
 - mindspore.dataset：创建 loss 所需要的用户数据集。
 - mindspore.train.Metrics：用户的评估指标。
- **collect_landscape** (Union[dict, None]) - 创建 loss 地形图所用的参数含义与 SummaryCollector 同名字段一致。此处设置的目的是，允许用户可以自由修改创建 loss 地形图参数。默认值：None。
 - **landscape_size** (int) - 指定生成 loss 地形图的图像分辨率。例如：如果设置为 128，则 loss 地形图的分辨率是 128*128。计算 loss 地形图的时间随着分辨率的增大而增加。默认值：40。可选值：3-256。
 - **create_landscape** (dict) - 选择创建哪种类型的 loss 地形图，分为训练过程 loss 地形图 (train) 和训练结果 loss 地形图 (result)。默认值：{"train": True, "result": True}。可选值：True/False。
 - **num_samples** (int) - 创建 loss 地形图所使用的数据集的大小。例如：在图像数据集中，您可以设置 num_samples 为 128，这意味着将有 128 张图片被用来创建 loss 地形图。注意：num_samples 越大，计算 loss 地形图时间越长。默认值：128。
 - **intervals** (List[List[int]]) - 指定创建 loss 地形图所需要的 checkpoint 区间。例如：如果用户想要创建两张训练过程的 loss 地形图，分别为 1-5epoch 和 6-10epoch，则用户可以设置 [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]]。注意：每个区间至少包含 3 个 epoch。
- **device_ids** (List(int)) - 指定创建 loss 地形图所使用的目标设备的 ID。例如：[0, 1] 表示使用设备 0 和设备 1 来创建 loss 地形图。默认值：None。
- **output** (str) - 指定保存 loss 地形图的路径。默认值：None。默认保存路径与 summary 文件相同。

mindspore.SummaryRecord

```
class mindspore.SummaryRecord (log_dir, file_prefix='events', file_suffix='_MS', num_process=32, network=None,
                                max_file_size=None, raise_exception=False, export_options=None)
```

SummaryRecord 用于记录 summary 数据和 lineage 数据。

该方法将在一个指定的目录中创建 summary 文件和 lineage 文件，并将数据写入文件。

它通过执行 *record* 方法将数据写入文件。

说明:

1. 确保在最后关闭 SummaryRecord，否则进程不会退出。请参阅下面的示例部分，了解如何用两种方式正确关闭 SummaryRecord。
2. 每次训练只允许创建一个 SummaryRecord 实例，否则会导致数据写入异常。
3. SummaryRecord 仅支持 Linux 系统。
4. 编译 MindSpore 时，设置 *-s on* 关闭维测功能后，SummaryRecord 不可用。

参数:

- **log_dir** (str) - log_dir 是用来保存 summary 文件的目录。
- **file_prefix** (str) - 文件的前缀。默认值: 'events'。
- **file_suffix** (str) - 文件的后缀。默认值: '_MS'。
- **num_process** (int) - 设置保存 summary 数据的进程数。进程越多性能越好，但是设置进程多可能会出现 Host 内存不足的问题。默认值: 32。
- **network** (Cell) - 表示用于保存计算图的网络。默认值: None。
- **max_file_size** (int, 可选) - 可写入磁盘的每个文件的最大大小（以字节为单位）。例如，预期写入文件最大不超过 4GB，则设置 *max_file_size=4*1024**3*。默认值: None，表示无限制。
- **raise_exception** (bool, 可选) - 设置在记录数据中发生 RuntimeError 或 OSError 异常时是否抛出异常。默认值: False，表示打印错误日志，不抛出异常。
- **export_options** (Union[None, dict]) - 可以将保存在 summary 中的数据导出，并使用字典自定义所需的数据和文件格式。注：导出的文件大小不受 *max_file_size* 的限制。例如，您可以设置 {'tensor_format': 'npz'} 将 Tensor 导出为 npz 文件。支持导出的数据类型如下所示。默认值: None，表示不导出数据。
 - **tensor_format** (Union[str, None]) - 自定义导出的 Tensor 的格式。支持 ['npz', None]，其中 'npz' 表示将 Tensor 导出为 NPY 文件。默认值: None，表示不导出 Tensor。

异常:

- **TypeError** - *max_file_size* 不是整型，或 *file_prefix* 和 *file_suffix* 不是字符串。
- **ValueError** - 编译 MindSpore 时，设置 *-s on* 关闭了维测功能。

样例:

```
>>> import mindspore as ms
>>> if __name__ == '__main__':
...     # use in with statement to auto close
...     with ms.SummaryRecord(log_dir="./summary_dir") as summary_record:
...         pass
... 
```

(下页继续)

(续上页)

```

...     # use in try .. finally .. to ensure closing
...     try:
...         summary_record = ms.SummaryRecord(log_dir="./summary_dir")
...     finally:
...         summary_record.close()

```

add_value (*plugin, name, value*)

添加需要记录的值。

参数：

- **plugin** (str) - 数据类型标签。
 - graph：代表添加的数据为计算图。
 - scalar：代表添加的数据为标量。
 - image：代表添加的数据为图片。
 - tensor：代表添加的数据为张量。
 - histogram：代表添加的数据为直方图。
 - train_lineage：代表添加的数据为训练阶段的 lineage 数据。
 - eval_lineage：代表添加的数据为评估阶段的 lineage 数据。
 - dataset_graph：代表添加的数据为数据图。
 - custom_lineage_data：代表添加的数据为自定义 lineage 数据。
 - LANDSCAPE：代表添加的数据为地形图。
- **name** (str) - 数据名称。
- **value** (Union[Tensor, GraphProto, TrainLineage, EvaluationLineage, DatasetGraph, UserDefinedInfo, LossLandscape]) - 待存储的值。
 - 当 plugin 为“graph”时，参数值的数据类型应为“GraphProto”对象。具体详情，请参见 [mindspore/ccsrc/anf_ir.proto](#)。
 - 当 plugin 为“scalar”、“image”、“tensor”或“histogram”时，参数值的数据类型应为“Tensor”对象。
 - 当 plugin 为“train_lineage”时，参数值的数据类型应为“TrainLineage”对象。具体详情，请参见 [mindspore/ccsrc/lineage.proto](#)。
 - 当 plugin 为“eval_lineage”时，参数值的数据类型应为“EvaluationLineage”对象。具体详情，请参见 [mindspore/ccsrc/lineage.proto](#)。
 - 当 plugin 为“dataset_graph”时，参数值的数据类型应为“DatasetGraph”对象。具体详情，请参见 [mindspore/ccsrc/lineage.proto](#)。
 - 当 plugin 为“custom_lineage_data”时，参数值的数据类型应为“UserDefinedInfo”对象。具体详情，请参见 [mindspore/ccsrc/lineage.proto](#)。
 - 当 plugin 为“LANDSCAPE”时，参数值的数据类型应为“LossLandscape”对象。具体详情，请参见 [mindspore/ccsrc/summary.proto](#)。

异常：

- **ValueError** - *plugin* 的值不在可选值内。
- **TypeError** - *name* 不是非空字符串，或当 *plugin* 为“scalar”、“image”、“tensor”或“histogram”时，*value* 的数据类型不是“Tensor”对象。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> if __name__ == '__main__':
...     with ms.SummaryRecord(log_dir="./summary_dir", file_prefix="xx_", file_suffix="_
    yy") \
...         as summary_record:
...             summary_record.add_value('scalar', 'loss', Tensor(0.1))
```

close()

将缓冲区中的数据立刻写入文件，并关闭 SummaryRecord。请使用 with 语句或 try...finally 语句进行自动关闭。

样例:

```
>>> import mindspore as ms
>>> if __name__ == '__main__':
...     try:
...         summary_record = ms.SummaryRecord(log_dir="./summary_dir")
...     finally:
...         summary_record.close()
```

flush()

刷新缓冲区，将缓冲区中的数据写入磁盘。

调用该函数以确保所有挂起事件都已写入到磁盘。

样例:

```
>>> import mindspore as ms
>>> if __name__ == '__main__':
...     with ms.SummaryRecord(log_dir="./summary_dir", file_prefix="xx_", file_suffix="_
    yy") \
...         as summary_record:
...             summary_record.flush()
```

property log_dir

获取日志文件的完整路径。

返回:

str, 日志文件的完整路径。

样例:

```
>>> import mindspore as ms
>>> if __name__ == '__main__':
...     with ms.SummaryRecord(log_dir="./summary_dir", file_prefix="xx_", file_suffix="_
    yy") \
...         as summary_record:
...             log_dir = summary_record.log_dir
```

record(step, train_network=None, plugin_filter=None)

记录 summary。

参数:

- **step** (int) - 表示当前的 step。
- **train_network** (Cell) - 表示用于保存计算图的训练网络。默认值: None，表示当原始网络的图为 None 时，不保存计算图。

- **plugin_filter** (Callable[[str], bool], 可选) - 过滤器函数，用于过滤需要写入的标签项。默认值：None。

返回：

bool，表示记录是否成功。

异常：

- **TypeError** - *step* 不为整型，或 *train_network* 的类型不为 *mindspore.nn.Cell*。

样例：

```
>>> import mindspore as ms
>>> if __name__ == '__main__':
...     with ms.SummaryRecord(log_dir="./summary_dir", file_prefix="xx_", file_suffix="_
->yy") \
...         as summary_record:
...             result = summary_record.record(step=2)
```

set_mode (*mode*)

设置模型运行阶段。不同的阶段会影响记录数据的内容。

参数：

- **mode** (str) - 待设置的网络阶段，可选值为“train”或“eval”。
 - train：代表训练阶段。
 - eval：代表评估阶段，此时 *summary_record* 不会记录 *summary* 算子的数据。

异常：

- **ValueError** - *mode* 的值不在可选值内。

样例：

```
>>> import mindspore as ms
>>> if __name__ == '__main__':
...     with ms.SummaryRecord(log_dir="./summary_dir", file_prefix="xx_", file_suffix="_
->yy") \
...         as summary_record:
...             summary_record.set_mode('eval')
```

mindspore.set_dump

mindspore.set_dump (*target*, *enabled=True*)

启用或者禁用 *target* 及其子节点的 Dump 数据功能。

target 为 *mindspore.nn.Cell* 或 *mindspore.ops.Primitive* 的实例。请注意，此 API 仅在开启同步 Dump 功能且 Dump 配置文件中的 *dump_mode* 字段为 "2" 时生效。有关详细信息，请参阅 [Dump 功能文档](#)。默认状态下，*mindspore.nn.Cell* 和 *mindspore.ops.Primitive* 实例不使能 Dump 数据功能。

说明：

1. 此 API 只在 Ascend 后端图编译等级为 O0/O1 的图模式下有效，且无法 Dump 被优化融合的 Primitive 算子。
2. 此 API 只支持训练开始前调用。如果在训练过程中调用这个 API，可能不会有效果。
3. 使用 *set_dump(Cell, True)* 后，Cell 正向计算和反向计算（梯度运算产生的计算）中的算子会被 Dump。

4. 对于 `mindspore.nn.SoftmaxCrossEntropyWithLogits` 层, 正向计算和反向计算使用同一组算子。因此, 只能看到反向计算中的 Dump 数据。请注意, 当使用 `sparse=True` 和 `reduce="mean"` 初始化时, `mindspore.nn.SoftmaxCrossEntropyWithLogits` 层也将在内部使用这些算子。

参数:

- **target** (Union[Cell, Primitive]) - 要设置 Dump 标志的 Cell 或 Primitive 的实例。
- **enabled** (bool, 可选) - True 表示启用 Dump, False 表示禁用 Dump, 默认值: True。

支持平台:

Ascend

样例:

说明: 运行此样例之前请设置环境变量 `MINDSPORE_DUMP_CONFIG` 到配置文件, 并将配置文件中的 `dump_mode` 字段设置为 2。详细信息请参阅 [Dump 功能文档](#)。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore import Tensor, set_dump
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> ms.set_device(device_target="Ascend")
>>>
>>> class MyNet(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.conv1 = nn.Conv2d(5, 6, 5, pad_mode='valid')
...         self.relu1 = nn.ReLU()
...
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu1(x)
...         return x
>>>
>>> if __name__ == "__main__":
...     net = MyNet()
...     set_dump(net.conv1)
...     input_tensor = Tensor(np.ones([1, 5, 10, 10], dtype=np.float32))
...     output = net(input_tensor)
```

mindspore.Profiler

class mindspore.Profiler (**kwargs)

当前接口即将弃用, 请使用 `mindspore.profiler.profile` 代替。MindSpore 用户能够通过该类对神经网络的性能进行采集。首先通过创建 **Profiler** 初始化对象开始分析, 然后使用 `Profiler.analyse()` 停止收集并分析结果。可通过 [MindStudio Insight](#) 工具可视化分析结果。目前, Profiler 支持 AICORE 算子、AICPU 算子、HostCPU 算子、内存、设备通信、集群等数据的分析。

参数:

- **start_profile** (bool, 可选) - 该参数控制是否在 Profiler 初始化时开启数据采集。默认值: True。

- **output_path** (str, 可选) - 表示输出数据的路径。默认值: `"./data"`。
- **profiler_level** (ProfilerLevel, 可选) - (仅限 Ascend) 表示采集性能数据级别。默认值: `ProfilerLevel.Level0`。
 - `ProfilerLevel.LevelNone`: 该设置仅在开启 `mstx` 时生效, 表示不采集 `device` 侧的任何算子数据。
 - `ProfilerLevel.Level0`: 最精简的采集性能数据级别, 采集计算类算子的耗时数据和通信类大算子的基础数据。
 - `ProfilerLevel.Level1`: 在 `Level0` 的基础上额外采集 CANN 层中 AscendCL 数据、AICORE 性能数据以及通信类小算子数据。
 - `ProfilerLevel.Level2`: 在 `Level1` 的基础上额外采集 CANN 层中 GE 和 Runtime 数据。
- **activities** (list, 可选) - 表示需要收集的性能数据类型。默认值: `[ProfilerActivity.CPU, ProfilerActivity.NPU]`。
 - `ProfilerActivity.CPU`: 收集 MindSpore 框架数据。
 - `ProfilerActivity.NPU`: 收集 CANN 软件栈和 NPU 数据。
 - `ProfilerActivity.GPU`: 收集 GPU 数据。
- **schedule** (schedule, 可选) - 设置采集的动作策略, 由 `schedule` 类定义, 需要配合 `step` 接口使用, 默认值: `None`, 表示采集全部 `step` 的性能数据, 详细介绍请参考 [mindspore.profiler.schedule](#)。
- **on_trace_ready** (Callable, 可选) - 设置当性能数据采集完成时, 执行的回调函数。默认值: `None`, 表示只采集, 不解析性能数据, 详细介绍请参考 [mindspore.profiler.tensorboard_trace_handler\(\)](#)。
- **profile_memory** (bool, 可选) - (仅限 Ascend) 表示是否收集 Tensor 内存数据。当值为 `True` 时, 收集这些数据。使用该参数时, `activities` 必须设置为 `[ProfilerActivity.CPU, ProfilerActivity.NPU]`。在图编译等级为 `O2` 时收集算子内存数据, 需要从第一个 `step` 开始采集。默认值: `False`, 该参数目前采集的算子名称不完整。将在后续版本修复, 建议使用环境变量 `MS_ALLOC_CONF` 代替。
- **aic_metrics** (AicoreMetrics, 可选) - (仅限 Ascend) 收集的 AICORE 性能数据类型, 使用该参数时, `activities` 必须包含 `ProfilerActivity.NPU`, 且值必须包含在 `AicoreMetrics` 枚举值中, 默认值: `AicoreMetrics.AiCoreNone`, 每种类型包含的数据项如下:
 - `AicoreMetrics.AiCoreNone`: 不收集任何 AICORE 数据。
 - `AicoreMetrics.ArithmeticUtilization`: 包含 `mac_fp16/int8_ratio`、`vec_fp32/fp16/int32_ratio`、`vec_misc_ratio` 等。
 - `AicoreMetrics.PipeUtilization`: 包含 `vec_ratio`、`mac_ratio`、`scalar_ratio`、`mte1/mte2/mte3_ratio`、`icache_miss_rate` 等。
 - `AicoreMetrics.Memory`: 包含 `ub_read/write_bw`、`l1_read/write_bw`、`l2_read/write_bw`、`main_mem_read/write_bw` 等。
 - `AicoreMetrics.MemoryL0`: 包含 `l0a_read/write_bw`、`l0b_read/write_bw`、`l0c_read/write_bw` 等。
 - `AicoreMetrics.ResourceConflictRatio`: 包含 `vec_bankgroup/bank/resc_cflt_ratio` 等。
 - `AicoreMetrics.MemoryUB`: 包含 `ub_read/write_bw_mte`、`ub_read/write_bw_vector`、`ub_/write_bw_scalar` 等。
 - `AicoreMetrics.L2Cache`: 包含 `write_cache_hit`、`write_cache_miss_allocate`、`r0_read_cache_hit`、`r1_read_cache_hit` 等。本功能仅支持 Atlas A2 训练系列产品。
 - `AicoreMetrics.MemoryAccess`: 主存以及 L2 Cache 的存访带宽和存量统计。
- **with_stack** (bool, 可选) - (仅限 Ascend) 表示是否收集 Python 侧的调用栈的数据, 此数据在 `timeline` 中采用火焰图的形式呈现, 使用该参数时, `activities` 必须包含 `ProfilerActivity.CPU`。默认值: `False`。
- **data_simplification** (bool, 可选) - (仅限 Ascend) 是否开启数据精简, 开启后仅保留 `profiler` 的交付件以及 `PROF_XXX` 目录下的原始性能数据, 以节省空间。默认值: `True`。

- **l2_cache** (bool, 可选) - (仅限 Ascend) 是否收集 L2 Cache 数据, 当值为 True 时, 收集这些数据。默认值: False。该采集项在 ASCEND_PROFILER_OUTPUT 文件夹下生成 l2_cache.csv 文件。在 O2 模式下, 仅支持 schedule 配置中 wait 和 skip_first 参数都为 0 的场景。
- **hbm_ddr** (bool, 可选) - (仅限 Ascend) 是否收集片上内存/DDR 内存读写速率数据, 当值为 True 时, 收集这些数据。默认值: False。
- **pcie** (bool, 可选) - (仅限 Ascend) 是否收集 PCIe 带宽数据, 当值为 True 时, 收集这些数据。默认值: False。
- **data_process** (bool, 可选) - (Ascend/GPU) 表示是否收集数据准备性能数据, 默认值: False。
- **parallel_strategy** (bool, 可选) - (仅限 Ascend) 表示是否收集并行策略性能数据, 默认值: False。
- **sync_enable** (bool, 可选) - (仅限 GPU) Profiler 是否用同步的方式收集算子耗时, 默认值: True。
 - True: 同步方式, 在把算子发送到 GPU 之前, 在 CPU 端记录开始时间戳。然后在算子执行完毕返回到 CPU 端后, 再记录结束时间戳。算子耗时为两个时间戳的差值。
 - False: 异步方式, 算子耗时为从 CPU 发送到 GPU 的耗时。这种方式能减少因增加 Profiler 对整体训练时间的影响。

异常:

- **RuntimeError** - 当 CANN 的版本与 MindSpore 版本不匹配时, MindSpore 无法解析生成的 ascend_job_id 目录结构。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn
>>> import mindspore.dataset as ds
>>> from mindspore import Profiler
>>> from mindspore.profiler import ProfilerLevel, ProfilerActivity, AicoreMetrics
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator():
...     for i in range(2):
...         yield (np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32))
>>>
>>> def train(net):
...     optimizer = nn.Momentum(net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator, ["data", "label"])
...     model = ms.train.Model(net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     # If the device_target is GPU, set the device_target to "GPU"
...     ms.set_context(mode=ms.GRAPH_MODE, device_target="Ascend")
... 
```

(下页继续)

(续上页)

```

...     # Init Profiler
...     # Note that the Profiler should be initialized before model.train
...     profiler = Profiler(profiler_level=ProfilerLevel.Level0,
...                         activities=[ProfilerActivity.CPU, ProfilerActivity.NPU],
...                         aic_metrics=AicoreMetrics.AiCoreNone)
...
...     # Train Model
...     net = Net()
...     train(net)
...
...     # Profiler end
...     profiler.analyse()

```

add_metadata (*key: str, value: str*)

上报自定义 metadata 键值对数据。

参数:

- **key** (str) - metadata 键值对的 key。
- **value** (str) - metadata 键值对的 value。

样例:

```

>>> from mindspore import Profiler
>>> # Profiler init.
>>> profiler = Profiler()
>>> # Call Profiler add_metadata
>>> profiler.add_metadata("test_key", "test_value")
>>> # Profiler end
>>> profiler.stop()

```

add_metadata_json (*key: str, value: str*)

上报自定义 metadata 键值对 value 为 json 字符串数据。

参数:

- **key** (str) - metadata 键值对的 key。
- **value** (str) - metadata 键值对的 value, 格式为 json 字符串。

样例:

```

>>> import json
>>> from mindspore import Profiler
>>> # Profiler init.
>>> profiler = Profiler()
>>> # Call Profiler add_metadata_json
>>> profiler.add_metadata_json("test_key", json.dumps({"key1": 1, "key2": 2}))
>>> # Profiler end, metadata will be saved in profiler_metadata.json
>>> profiler.stop()

```

analyse (*offline_path=None, pretty=False, step_list=None, mode='sync'*)

收集和分析训练的性能数据, 支持在训练中和训练后调用。样例如上所示。

参数:

- **offline_path** (Union[str, None], 可选) - 需要使用离线模式进行分析的数据路径。离线模式用于非正常退出场景。对于在线模式, 该参数应设置为 None。默认值: None。

- **pretty** (bool, 可选) - 对 json 文件进行格式化。该参数默认值为 `False`，即不进行格式化。
- **step_list** (list, 可选) - 只解析指定 `step` 的性能数据，指定的 `step` 必须是连续的整数。该参数默认值为 `None`，即进行全解析。
- **mode** (str, 可选) - 解析模式，同步解析或异步解析，可选参数为 `["sync", "async"]`，默认值为 `"sync"`。
 - `"sync"`：同步模式解析性能数据，会阻塞当前进程。
 - `"async"`：异步模式，另起一个子进程解析性能数据，不会阻塞当前进程。由于解析进程会额外占用 CPU 资源，请根据实际资源情况开启该模式。

样例：

```
>>> from mindspore.train import Callback
>>> from mindspore import Profiler
>>> class StopAtStep(Callback):
...     def __init__(self, start_step=1, stop_step=5):
...         super(StopAtStep, self).__init__()
...         self.start_step = start_step
...         self.stop_step = stop_step
...         self.profiler = Profiler(start_profile=False)
...
...     def step_begin(self, run_context):
...         cb_params = run_context.original_args()
...         step_num = cb_params.cur_step_num
...         if step_num == self.start_step:
...             self.profiler.start()
...
...     def step_end(self, run_context):
...         cb_params = run_context.original_args()
...         step_num = cb_params.cur_step_num
...         if step_num == self.stop_step:
...             self.profiler.stop()
...
...     def end(self, run_context):
...         self.profiler.analyse(step_list=[2, 3, 4], mode="sync")
```

classmethod offline_analyse (path: str, pretty=False, step_list=None, data_simplification=True)

离线分析训练的性能数据，性能数据采集结束后调用。

参数：

- **path** (str) - 需要进行离线分析的 profiling 数据路径，指定到 Profiler 上层目录。
- **pretty** (bool, 可选) - 对 json 文件进行格式化。该参数默认值为 `False`，即不进行格式化。
- **step_list** (list, 可选) - 只解析指定 `step` 的性能数据，指定的 `step` 必须是连续的整数。该参数默认值为 `None`，即进行全解析。
- **data_simplification** (bool, 可选) - 数据精简开关功能。默认值为 `True`，即开启数据精简。

样例：

```
>>> from mindspore import Profiler
>>> Profiler.offline_analyse("./profiling_path")
```

op_analyse (op_name, device_id=None)

获取 primitive 类型的算子性能数据。

参数：

- **op_name** (str 或 list) - 表示要查询的 primitive 算子类型。

- **device_id** (int, 可选) - 设备卡号, 表示指定解析哪张卡的算子性能数据。在网络训练或者推理时使用, 该参数可选。基于离线数据解析使用该接口时, 默认值: None。

异常:

- **TypeError** - *op_name* 参数类型不正确。
- **TypeError** - *device_id* 参数类型不正确。
- **RuntimeError** - 在 Ascend 上使用该接口获取性能数据。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import Profiler
>>> from mindspore import nn
>>> from mindspore import Model
>>> # Profiler init.
>>> profiler = Profiler()
>>> # Train Model or eval Model, taking LeNet5 as an example.
>>> # Refer to https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.
    ↪py
>>> net = LeNet5()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> # Create the dataset taking MNIST as an example.
>>> # Refer to https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.
    ↪py
>>> dataloader = create_dataset()
>>> model = Model(net, loss, optimizer)
>>> model.train(5, dataloader, dataset_sink_mode=False)
>>>
>>> # Profiler end
>>> profiler.analyse()
>>>
>>> profiler.op_analyse(op_name=["BiasAdd", "Conv2D"])
```

start()

开启 Profiler 数据采集。可以按条件开启 Profiler。

异常:

- **RuntimeError** - Profiler 已经开启。
- **RuntimeError** - 如果 *start_profile* 参数未设置或设置为 True。

样例:

```
>>> from mindspore.train import Callback
>>> from mindspore import Profiler
>>> class StopAtStep(Callback):
...     def __init__(self, start_step, stop_step):
...         super(StopAtStep, self).__init__()
...         self.start_step = start_step
...         self.stop_step = stop_step
...         self.profiler = Profiler(start_profile=False)
...
...     def step_begin(self, run_context):
...         cb_params = run_context.original_args()
```

(下页继续)

(续上页)

```

...     step_num = cb_params.cur_step_num
...     if step_num == self.start_step:
...         self.profiler.start()
...
...     def step_end(self, run_context):
...         cb_params = run_context.original_args()
...         step_num = cb_params.cur_step_num
...         if step_num == self.stop_step:
...             self.profiler.stop()
...
...     def end(self, run_context):
...         self.profiler.analyse()

```

step()

用于在 Ascend 设备上, 通过 `schedule` 和 `on_trace_ready` 区分步骤收集和解析性能数据。

异常:

- **RuntimeError** - 如果 `start_profile` 参数未设置或 Profiler 未开启。
- **RuntimeError** - 如果 `schedule` 参数未设置。

样例:

```

>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.dataset as ds
>>> from mindspore import context, nn, Profiler
>>> from mindspore.profiler import schedule, tensorboard_trace_handler, ProfilerLevel, \
↳AicoreMetrics,
>>> ExportType, ProfilerActivity
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(2, 2)
...
...     def construct(self, x):
...         return self.fc(x)
>>>
>>> def generator_net():
...     for _ in range(2):
...         yield np.ones([2, 2]).astype(np.float32), np.ones([2]).astype(np.int32)
>>>
>>> def train(test_net):
...     optimizer = nn.Momentum(test_net.trainable_params(), 1, 0.9)
...     loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
...     data = ds.GeneratorDataset(generator_net(), ["data", "label"])
...     model = ms.train.Model(test_net, loss, optimizer)
...     model.train(1, data)
>>>
>>> if __name__ == '__main__':
...     context.set_context(mode=ms.PYNATIVE_MODE, device_target="Ascend")
...
...     net = Net()
...     STEP_NUM = 15
...
...     with Profiler(schedule=schedule(wait=1, warmup=1, active=2, repeat=1, skip_
↳first=2),

```

(下页继续)

(续上页)

```

...         on_trace_ready=tensorboard_trace_handler) as prof:
...     for _ in range(STEP_NUM):
...         train(net)
...         prof.step()

```

stop()

停止 Profiler。可以按条件停止 Profiler。

异常：

- **RuntimeError** - Profiler 没有开启。

样例：

```

>>> from mindspore.train import Callback
>>> from mindspore import Profiler
>>> class StopAtEpoch(Callback):
...     def __init__(self, start_epoch, stop_epoch):
...         super(StopAtEpoch, self).__init__()
...         self.start_epoch = start_epoch
...         self.stop_epoch = stop_epoch
...         self.profiler = Profiler(start_profile=False)
...
...     def epoch_begin(self, run_context):
...         cb_params = run_context.original_args()
...         epoch_num = cb_params.cur_epoch_num
...         if epoch_num == self.start_epoch:
...             self.profiler.start()
...
...     def epoch_end(self, run_context):
...         cb_params = run_context.original_args()
...         epoch_num = cb_params.cur_epoch_num
...         if epoch_num == self.stop_epoch:
...             self.profiler.stop()
...
...     def end(self, run_context):
...         self.profiler.analyse()

```

1.9.3 日志

`mindspore.get_level`

获取日志记录器的级别。

`mindspore.get_log_config`

获取日志配置。

mindspore.get_level

`mindspore.get_level()`
获取日志记录器的级别。

返回:

string, 日志级别包括 4(EXCEPTION)、3(ERROR)、2(WARNING)、1(INFO) 和 0(DEBUG)。

样例:

```
>>> import os
>>> os.environ['GLOG_v'] = '3'
>>> import mindspore as ms
>>> level = ms.get_level()
>>> print(level)
'3'
```

mindspore.get_log_config

`mindspore.get_log_config()`
获取日志配置。

返回:

Dict, 日志配置字典。

样例:

```
>>> import os
>>> import mindspore as ms
>>> os.environ['GLOG_v'] = '1'
>>> os.environ['GLOG_logtostderr'] = '0'
>>> os.environ['GLOG_log_dir'] = '/var/log'
>>> os.environ['logger_maxBytes'] = '5242880'
>>> os.environ['logger_backupCount'] = '10'
>>> os.environ['GLOG_stderrthreshold'] = '2'
>>> config = ms.get_log_config()
>>> print(config)
{'GLOG_v': '1', 'GLOG_logtostderr': '0', 'GLOG_log_dir': '/var/log',
 'logger_maxBytes': '5242880', 'logger_backupCount': '10', 'GLOG_stderrthreshold': '2'}
```

1.9.4 安装验证

`mindspore.run_check`

提供了便捷的 API 用以查询 MindSpore 的安装是否成功。

mindspore.run_check

`mindspore.run_check()`

提供了便捷的 API 用以查询 MindSpore 的安装是否成功。如果检查返回结果中的版本与预期不符，请在 `run_check()` 之前使用 `mindspore.set_context()` 设置 `device_target`。

样例：

```
>>> import mindspore
>>> mindspore.run_check()
MindSpore version: xxx
The result of multiplication calculation is correct, MindSpore has been installed on_
↪platform [XX] successfully!
```

1.9.5 安全

`mindspore.obfuscate_ckpt`

根据用户配置的混淆策略，对模型的 Checkpoint 进行混淆保护，防止攻击者窃取模型资产。

`mindspore.load_obf_params_into_net`

根据用户配置的混淆策略，对模型结构进行修改，并将混淆态 Checkpoint 加载到模型中。

mindspore.obfuscate_ckpt

`mindspore.obfuscate_ckpt(network, ckpt_files, target_modules=None, obf_config=None, saved_path='.', obfuscate_scale=100)`

根据用户配置的混淆策略，对模型的 Checkpoint 进行混淆保护，防止攻击者窃取模型资产。

参数：

- **network** (`nn.Cell`) - 待混淆的原始网络。
- **ckpt_files** (`str`) - 待混淆的原始权重文件的存储路径。
- **target_modules** (`list[str]`，可选) - 需要混淆的目标算子。第一个字符串表示目标算子在原网络中的路径，需为 "A/B/C" 的形式，第二个字符串表示同一个路径下的多个目标算子名，需是 "D|E|F" 的形式。例如，GPT2 的 `target_modules` 可以是 `['backbone/blocks/attention', 'dense1|dense2|dense3']`。如果 `target_modules` 有第三个值，格式需是 `"obfuscate_layers:all"` 或 `"obfuscate_layers:int"`，表示需要混淆重复层（如 `transformer` 层或 `resnet` 块）的层数。默认值：None。
- **obf_config** (`dict`，可选) - 模型混淆策略的配置。默认值：None。
- **saved_path** (`str`，可选) - 混淆后权重文件的保存路径。默认值：'./'。
- **obfuscate_scale** (`Union[float, int]`，可选) - 权重混淆尺度，控制混淆系数的取值范围为大于 1 的 `int` 或 `float`。默认值：100。

返回：

解混淆元数据 (`dict`)。该数据在运行混淆网络时加载使用。

异常：

- **TypeError** - `network` 不是 `nn.Cell` 类型。
- **TypeError** - `ckpt_files` 不是 `str` 类型或者 `saved_path` 不是 `str` 类型。
- **TypeError** - `target_modules` 不是 `list` 类型。

- **TypeError** - *target_modules* 中的元素不是 `str` 类型。
- **TypeError** - *obf_config* 不是 `dict` 类型。
- **ValueError** - *ckpt_files* 目录不存在或者 *saved_path* 目录不存在。
- **ValueError** - *target_modules* 中的元素个数小于 2。
- **ValueError** - *target_modules* 的第一个字符串包含大小写字母、数字、'_' 或 '/' 以外的字符。
- **ValueError** - *target_modules* 的第二个字符串为空，或包含大小写字母、数字、'_' 或 '/' 以外的字符。
- **ValueError** - *target_modules* 的第三个字符串不是 "obfuscate_layers:all" 或 "obfuscate_layers:int" 的格式。

样例：

```
>>> from mindspore import obfuscate_ckpt, save_checkpoint
>>> # Refer to https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> save_checkpoint(net, './test_net.ckpt')
>>> target_modules = ['', 'fc1|fc2']
>>> obfuscate_ckpt(net, './', target_modules=target_modules, saved_path='./')
```

mindspore.load_obf_params_into_net

`mindspore.load_obf_params_into_net` (*network*, *target_modules*=None, *obf_ratios*=None, *obf_config*=None, *data_parallel_num*=1, ***kwargs*)

根据用户配置的混淆策略，对模型结构进行修改，并将混淆态 Checkpoint 加载到模型中。

参数：

- **network** (`nn.Cell`) - 待混淆的原始网络。
- **target_modules** (`list[str]`，可选) - 需要混淆的目标算子。第一个字符串表示目标算子在原网络中的路径，需为 "A/B/C" 的形式。第二个字符串表示同一个路径下的多个目标算子名，需为 "D|E|F" 的形式。例如，GPT2 的 *target_modules* 可以是 ['backbone/blocks/attention', 'dense1|dense2|dense3']。如果 *target_modules* 有第三个值，它的格式需为 "obfuscate_layers:all" 或 "obfuscate_layers:int"，这表示需要混淆重复层（如 transformer 层或 resnet 块）的层数。默认值：None。
- **obf_ratios** (`Tensor`，可选) - 混淆系数，由 *mindspore.obfuscate_ckpt* 接口生成。默认值：None。
- **obf_config** (`dict`，可选) - 模型混淆策略的配置。默认值：None。
- **data_parallel_num** (`int`，可选) - 模型并行训练的数据并行度。默认值：1。
- **kwargs** (`dict`) - 配置选项字典。
 - **ignored_func_decorators** (`list[str]`) - Python 代码中函数装饰器的名字列表。
 - **ignored_class_decorators** (`list[str]`) - Python 代码中类装饰器的名字列表。

返回：

混淆后模型 (`nn.Cell`)。

异常：

- **TypeError** - *network* 不是 `nn.Cell` 类型。
- **TypeError** - *obf_ratios* 不是 `Tensor` 类型。
- **TypeError** - *target_modules* 不是 `list` 类型。
- **TypeError** - *obf_config* 不是 `dict` 类型。

- **TypeError** - *target_modules* 中的元素不是 `str` 类型。
- **ValueError** - *obf_ratios* 为空。
- **ValueError** - *target_modules* 中的元素个数小于 2。
- **ValueError** - *target_modules* 的第一个字符串包含大小写字母、数字、'_' 和 '/' 以外的字符。
- **ValueError** - *target_modules* 的第二个字符串为空或包含大小写字母，数字，'_' 和 '/' 以外的字符。
- **ValueError** - *target_modules* 的第三个字符串不是 "obfuscate_layers:all" 或 "obfuscate_layers:int" 的格式。
- **TypeError** - *ignored_func_decorators* 不是字符串列表，或 *ignored_class_decorators* 不是字符串列表。

样例：

```
>>> from mindspore import obfuscate_ckpt, save_checkpoint, load_checkpoint, Tensor
>>> import mindspore.common.dtype as mstype
>>> import numpy as np
>>> # Refer to https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> save_checkpoint(net, './test_net.ckpt')
>>> target_modules = ['', 'fc1|fc2']
>>> # obfuscate ckpt files
>>> obfuscate_ckpt(net, './', target_modules=target_modules, saved_path='./')
>>> # load obf ckpt into network
>>> new_net = LeNet5()
>>> load_checkpoint('./test_net_obf.ckpt', new_net)
>>> obf_net = load_obf_params_into_net(new_net, target_modules)
```


mindspore.ops 提供了大量的 function 接口，包括神经网络函数、数学运算函数、Tensor 操作函数、Parameter 操作函数、微分函数等。

动态 shape 的支持情况详见 [ops 接口动态 shape 支持情况](#)。

模块导入方法如下：

```
from mindspore import ops
```

MindSpore 中 *mindspore.ops* 接口与上一版本相比，新增、删除和支持平台的变化信息请参考 [mindspore.ops API 接口变更](#)。

2.1 Tensor 操作函数

2.1.1 Tensor 创建

接口名	概述	支持平台
<i>mindspore.ops.eps</i>	创建一个与输入的数据类型和 shape 都相同的 tensor ，值为对应数据类型能表达的最小值。	Ascend GPU CPU
<i>mindspore.ops.eye</i>	返回一个主对角线上元素为 1，其余元素为 0 的 tensor 。	Ascend GPU CPU
<i>mindspore.ops.fill</i>	创建 tensor ，用指定值填充。	Ascend GPU CPU
<i>mindspore.ops.full</i>	创建 tensor ，用指定值填充。	Ascend GPU CPU
<i>mindspore.ops.full_like</i>	返回一个用指定值填充的 tensor ，shape 与 <i>input</i> 相同。	Ascend GPU CPU
<i>mindspore.ops.linspace</i>	创建一个 <i>steps</i> 个元素的，在 [start, end] 区间内均匀分布的一维 tensor 。	Ascend GPU CPU
<i>mindspore.ops.logspace</i>	返回一个 <i>steps</i> 个元素的，在 [$base^{start}$, $base^{end}$] 区间内均匀分布的一维 tensor 。	Ascend GPU CPU

下页继续

表 1 - 续上页

<code>mindspore.ops.move_to</code>	拷贝 tensor 到目标设备，包含同步和异步两种方式，默认是同步方式。	Ascend CPU
<code>mindspore.ops.one_hot</code>	生成一个新的 tensor ，索引 <i>indices</i> 表示的位置上取值 <i>on_value</i> ，其他所有位置上取值 <i>off_value</i> 。	Ascend GPU CPU
<code>mindspore.ops.ones</code>	创建一个值全为 1 的 tensor 。	Ascend GPU CPU
<code>mindspore.ops.ones_like</code>	返回一个 shape 与输入相同，值为 1 的 tensor 。	Ascend GPU CPU
<code>mindspore.ops.arange</code>	返回一个在 $[start, end)$ 区间内，步长为 <i>step</i> 的 tensor 。	Ascend GPU CPU
<code>mindspore.ops.range</code>	返回一个在 $[start, end)$ 区间内，步长为 <i>step</i> 的 tensor 。	GPU CPU
<code>mindspore.ops.zeros</code>	创建一个值全为 0 的 tensor 。	Ascend GPU CPU
<code>mindspore.ops.zeros_like</code>	返回一个 shape 与输入相同，值为 0 的 tensor 。	Ascend GPU CPU
<code>mindspore.ops.heaviside</code>	逐元素进行 Heaviside 阶跃函数运算。	Ascend GPU CPU

mindspore.ops.eps

`mindspore.ops.eps(x)`
创建一个与输入的数据类型和 **shape** 都相同的 **tensor**，值为对应数据类型能表达的最小值。

- 参数：
- **x** (Tensor) - 输入 **tensor**。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([4, 1, 2, 3], mindspore.float32)
>>> mindspore.ops.eps(x)
Tensor(shape=[4], dtype=Float32, value= [ 1.19209290e-07,  1.19209290e-07,  1.19209290e-07,  1.19209290e-07])
```

mindspore.ops.eye

`mindspore.ops.eye(n, m=None, dtype=None)`
返回一个主对角线上元素为 1，其余元素为 0 的 **tensor**。

- 参数：
- **n** (int) - 返回的行数。
 - **m** (int, 可选) - 返回的列数。如果为 **None**，与 **n** 相等。默认 **None**。
 - **dtype** (mindspore.dtype, 可选) - 指定数据类型。默认 **None**。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.eye(3)
>>> print(output)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
>>>
>>> output = mindspore.ops.eye(3, 4)
>>> print(output)
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

mindspore.ops.fill

`mindspore.ops.fill` (*type, shape, value*)

创建 `tensor`, 用指定值填充。

参数:

- **type** (`mindspore.dtype`) - 指定数据类型。
- **shape** (`Union(Tensor, tuple[int])`) - 指定的 `shape`。
- **value** (`Union(Tensor, number.Number, bool)`) - 指定值。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> mindspore.ops.fill(mindspore.float32, (2, 3), 1.2)
Tensor(shape=[2, 3], dtype=Float32, value=
[[ 1.20000005e+00,  1.20000005e+00,  1.20000005e+00],
 [ 1.20000005e+00,  1.20000005e+00,  1.20000005e+00]])
```

mindspore.ops.full

`mindspore.ops.full` (*size, fill_value, *, dtype=None*)

创建 `tensor`, 用指定值填充。

说明: `fill_value` 不支持复数类型。

参数:

- **size** (`Union(tuple[int], list[int])`) - 指定的 `shape`。
- **fill_value** (`number.Number`) - 指定值。

关键字参数：

- **dtype** (mindspore.dtype) - 指定数据类型。默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> mindspore.ops.full((2, 3), 1.2)
Tensor(shape=[2, 3], dtype=Int64, value=
[[1, 1, 1],
 [1, 1, 1]])
>>> mindspore.ops.full((2, 3), 1.2, dtype=mindspore.float32)
Tensor(shape=[2, 3], dtype=Float32, value=
[[ 1.20000005e+00,  1.20000005e+00,  1.20000005e+00],
 [ 1.20000005e+00,  1.20000005e+00,  1.20000005e+00]])
```

mindspore.ops.full_like

`mindspore.ops.full_like` (*input*, *fill_value*, *, *dtype=None*)

返回一个用指定值填充的 tensor，shape 与 *input* 相同。

说明： *fill_value* 不支持复数类型。

参数：

- **input** (Tensor) - 输入 tensor。
- **fill_value** (Number) - 指定值。

关键字参数：

- **dtype** (mindspore.dtype, 可选) - 指定数据类型。默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.ops.arange(4)
>>> mindspore.ops.full_like(input, 1.2)
Tensor(shape=[4], dtype=Int64, value= [1, 1, 1, 1])
>>> mindspore.ops.full_like(input, 1.2, dtype=mindspore.float32)
Tensor(shape=[4], dtype=Float32, value= [ 1.20000005e+00,  1.20000005e+00,  1.20000005e+00,  1.20000005e+00])
```

mindspore.ops.linspace

`mindspore.ops.linspace(start, end, steps)`

创建一个 `steps` 个元素的, 在 `[start, end]` 区间内均匀分布的一维 `tensor`。

$$step = (end - start) / (steps - 1)$$

$$output = [start, start + step, start + 2 * step, ..., end]$$

参数:

- **start** (`Union[Tensor, int, float]`) - 区间的起始值。
- **end** (`Union[Tensor, int, float]`) - 区间的末尾值。
- **steps** (`Union[Tensor, int]`) - 元素数量。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.linspace(3, 10, 5)
>>> print(output)
[ 3.    4.75  6.5   8.25 10. ]
>>> output = mindspore.ops.linspace(-10, 10, 5)
>>> print(output)
[-10.  -5.   0.   5.  10.]
>>> output = mindspore.ops.linspace(-10, 10, 1)
>>> print(output)
[-10.]
```

mindspore.ops.logspace

`mindspore.ops.logspace(start, end, steps, base=10, *, dtype=mstype.float32)`

返回一个 `steps` 个元素的, 在 `[basestart, baseend]` 区间内均匀分布的一维 `tensor`。

$$step = (end - start) / (steps - 1)$$

$$output = [base^{start}, base^{start+1*step}, ..., base^{start+(steps-2)*step}, base^{end}]$$

参数:

- **start** (`Union[float, Tensor]`) - 区间的起始值。
- **end** (`Union[float, Tensor]`) - 区间的末尾值。
- **steps** (`int`) - 元素数量。
- **base** (`int`, 可选) - 对数函数的底数。默认 10。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 `mstype.float32`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.logspace(1., 10., steps = 10, base = 10)
>>> print(output)
[1.e+01 1.e+02 1.e+03 1.e+04 1.e+05 1.e+06 1.e+07 1.e+08 1.e+09 1.e+10]
```

mindspore.ops.move_to

`mindspore.ops.move_to(input, to='CPU', blocking=True)`

拷贝 `tensor` 到目标设备, 包含同步和异步两种方式, 默认是同步方式。

说明: 该接口当前仅支持 `graph mode`, 并且 `jit_level` 为 `O0` 或 `O1`。

参数:

- **input** (`Union[Tensor, list[int], tuple[int]]`) - 输入 `tensor`, 当输入为 `list` 和 `tuple` 时会先转换为 `tensor` 再进行拷贝。
- **to** (`str`, 可选) - 指定目标设备, 可选值为 `"Ascend"`, `"CPU"`。默认 `"CPU"`。
- **blocking** (`bool`, 可选) - 是否使用同步拷贝。默认 `True` 表示同步拷贝。

返回:

目标设备上的新 `tensor`。

支持平台:

Ascend CPU

样例:

```
>>> import mindspore
>>> from mindspore import nn, ops, Tensor
>>> mindspore.set_context(mode=mindspore.GRAPH_MODE)
>>> class MoveToNet(nn.Cell):
...     def __init__(self):
...         super().__init__()
...
...     def construct(self, x):
...         cpu_x = ops.move_to(x, "CPU")
...         npu_x = ops.move_to(cpu_x, "Ascend")
...         return npu_x
...
>>> net = MoveToNet()
>>> x = Tensor([1, 2, 3], mindspore.int64)
>>> y = net(x)
>>> print(y)
[1 2 3]
```

mindspore.ops.one_hot

`mindspore.ops.one_hot(indices, depth, on_value=1, off_value=0, axis=-1)`

生成一个新的 tensor，索引 `indices` 表示的位置上取值 `on_value`，其他所有位置上取值 `off_value`。

说明：如果 `indices` 的秩为 n ，则输出 Tensor 的秩为 $n+1$ 。新轴在 `axis` 处创建。当执行设备是 Ascend 时，如果 `on_value` 为 int64 类型，则 `indices` 也必须为 int64 类型，且 `on_value` 和 `off_value` 的取值只能是 1 和 0。

参数：

- **indices** (Tensor) - 输入索引。
- **depth** (int) - one-hot 深度。
- **on_value** (Union[Tensor, int, float], 可选) - 填充索引位置上的值。默认 1。
- **off_value** (Union[Tensor, int, float], 可选) - 填充非索引位置上的值。默认 0。
- **axis** (int, 可选) - 指定计算轴。默认 -1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> indices = mindspore.tensor([0, 1, 2, 4])
>>> mindspore.ops.one_hot(indices, depth=5)
Tensor(shape=[4, 5], dtype=Int64, value=
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 0, 1]])
>>>
>>> mindspore.ops.one_hot(indices, depth=3)
Tensor(shape=[4, 3], dtype=Int64, value=
[[1, 0, 0],
 [0, 1, 0],
 [0, 0, 1],
 [0, 0, 0]])
>>> # If shape of indices is (N, C), and axis=-1, the returned shape will be (N, C, depth).
>>> indices = mindspore.tensor([[0, 2], [1, -1]])
>>> mindspore.ops.one_hot(indices, depth=3, on_value=10, off_value=4, axis=-1)
Tensor(shape=[2, 2, 3], dtype=Int64, value=
[[[10, 4, 4],
 [ 4, 4, 10]],
 [[ 4, 10, 4],
 [ 4, 4, 4]])]
>>> # If axis=0, the returned shape will be (depth, N, C).
>>> mindspore.ops.one_hot(indices, depth=3, on_value=10, off_value=4, axis=0)
Tensor(shape=[3, 2, 2], dtype=Int64, value=
[[[10, 4],
 [ 4, 4]],
```

(下页继续)

(续上页)

```
[[ 4, 4],
 [10, 4]],
 [[ 4, 10],
 [ 4, 4]]])
```

mindspore.ops.ones

`mindspore.ops.ones(shape, dtype=None)`

创建一个值全为 1 的 tensor。

警告：参数 *shape* 在后续版本中将不再支持 Tensor 类型的输入。

参数：

- **shape** (`Union[tuple[int], list[int], int, Tensor]`) - 指定的 shape。
- **dtype** (`mindspore.dtype`) - 指定数据类型，默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> mindspore.ops.ones(4)
Tensor(shape=[4], dtype=Float32, value= [ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00,  1.00000000e+00])
>>> mindspore.ops.ones((2, 3))
Tensor(shape=[2, 3], dtype=Float32, value=
[[ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00],
 [ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00]])
>>> mindspore.ops.ones(mindspore.tensor([1, 2, 3]))
Tensor(shape=[1, 2, 3], dtype=Float32, value=
[[[ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00],
 [ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00]])])
```

mindspore.ops.ones_like

`mindspore.ops.ones_like(input, *, dtype=None)`

返回一个 shape 与输入相同，值为 1 的 tensor。

参数：

- **input** (Tensor) - 输入 tensor。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 None 表示与 *input* 数据类型相同。

返回：

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.ops.arange(4)
>>> mindspore.ops.ones_like(input)
Tensor(shape=[4], dtype=Int64, value= [1, 1, 1, 1])
```

mindspore.ops.arange

mindspore.ops.arange (*start=0, end=None, step=1, *, dtype=None*)

返回一个在 $[start, end)$ 区间内, 步长为 *step* 的 tensor。

参数:

- **start** (Union[float, int, Tensor], 可选) - 区间的起始值。默认 0。
- **end** (Union[float, int, Tensor], 可选) - 区间的末尾值。默认 None 表示为 *start* 的值, 同时将 0 作为区间的起始值。
- **step** (Union[float, int, Tensor], 可选) - 值的间隔。默认 1。

关键字参数:

- **dtype** (mindspore.dtype, 可选) - 指定数据类型。默认 None。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> mindspore.ops.arange(4)
Tensor(shape=[4], dtype=Int64, value= [0, 1, 2, 3])
>>> mindspore.ops.arange(1, 6)
Tensor(shape=[5], dtype=Int64, value= [1, 2, 3, 4, 5])
>>> mindspore.ops.arange(0, 2, 0.5)
Tensor(shape=[4], dtype=Float32, value= [ 0.00000000e+00,  5.00000000e-01,  1.00000000e+00,  1.50000000e+00])
```

mindspore.ops.range

mindspore.ops.range (*start, end, step, maxlen=1000000*)

返回一个在 $[start, end)$ 区间内, 步长为 *step* 的 tensor。

说明: 三个输入必须全为整数或全为浮点数。

参数:

- **start** (number) - 区间的起始值。
- **end** (number) - 区间的末尾数。
- **step** (number) - 值的间隔。
- **maxlen** (int, 可选) - 该算子将会被分配能够存储 *maxlen* 个数据的内存。该参数是可选的，必须为正数，默认 1000000。如果输出的数量超过 *maxlen*，将会引起运行时错误。

返回：

Tensor

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> mindspore.ops.range(0, 6, 2)
Tensor(shape=[3], dtype=Int64, value= [0, 2, 4])
```

mindspore.ops.zeros

`mindspore.ops.zeros` (*size*, *dtype=None*)

创建一个值全为 0 的 tensor。

警告：参数 *size* 在后续版本中将不再支持 Tensor 类型的输入。

参数：

- **size** (Union[tuple[int], list[int], int, Tensor]) - 指定的 shape。
- **dtype** (*mindspore.dtype*, 可选) - 指定数据类型，默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> mindspore.ops.zeros(4)
Tensor(shape=[4], dtype=Float32, value= [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00])
>>> mindspore.ops.zeros((2, 3))
Tensor(shape=[2, 3], dtype=Float32, value=
[[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00]])
>>> mindspore.ops.zeros(mindspore.tensor([1, 2, 3]))
Tensor(shape=[1, 2, 3], dtype=Float32, value=
[[[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00]])])
```


mindspore.ops.zeros_like

`mindspore.ops.zeros_like(input, *, dtype=None)`

返回一个 shape 与输入相同, 值为 0 的 tensor。

参数:

- **input** (Tensor) - 输入 tensor。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 `None` 表示与 `input` 数据类型相同。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.ops.arange(4)
>>> mindspore.ops.zeros_like(input)
Tensor(shape=[4], dtype=Int64, value= [0, 0, 0, 0])
```

mindspore.ops.heaviside

`mindspore.ops.heaviside(input, values)`

逐元素进行 Heaviside 阶跃函数运算。

支持广播。

$$\text{heaviside}(\text{input}, \text{values}) = \begin{cases} 0, & \text{if input} < 0 \\ \text{values}, & \text{if input} = 0 \\ 1, & \text{if input} > 0 \end{cases}$$

参数:

- **input** (Tensor) - 输入 tensor。
- **values** (Tensor) - `input` 中元素为 0 时填充的值。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[ -2.,  0,  3],
...                           [ 5, -1,  0],
...                           [ 0,  7, -3]])
>>> values = mindspore.tensor([2, 0.5, 1])
>>> output = mindspore.ops.heaviside(input, values)
>>> print(output)
```

(下页继续)

(续上页)

```

[[0.  0.5  1. ]
 [1.  0.  1. ]
 [2.  1.  0. ]]
>>> output = mindspore.ops.heaviside(input, mindspore.tensor(0.5))
>>> print(output)
[[0.  0.5  1. ]
 [1.  0.  0.5]
 [0.5  1.  0. ]]
>>> output = mindspore.ops.heaviside(mindspore.tensor(-3.), values)
>>> print(output)
[0.  0.  0.]

```

2.1.2 随机生成函数

接口名	概述	支持平台
<code>mindspore.ops.bernoulli</code>	根据伯努利分布生成随机数 (0 或 1)。	GPU CPU
<code>mindspore.ops.gamma</code>	根据伽马分布生成随机数。	Ascend
<code>mindspore.ops.laplace</code>	根据拉普拉斯分布生成随机数。	Ascend GPU CPU
<code>mindspore.ops.multinomial</code>	生成一个多项式分布的 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.multinomial_with_replacement</code>	生成一个多项式分布的 <code>tensor</code> 。	CPU
<code>mindspore.ops.rand</code>	基于输入的 <code>size</code> 和 <code>dtype</code> , 返回一个 <code>tensor</code> , 其元素为服从均匀分布的 $[0, 1)$ 区间的随机数。	Ascend GPU CPU
<code>mindspore.ops.rand_like</code>	返回一个与输入 <code>shape</code> 相同的 <code>tensor</code> , 其元素为服从均匀分布的 $[0, 1)$ 区间的随机数。	Ascend GPU CPU
<code>mindspore.ops.randint</code>	返回一个 <code>tensor</code> , 其元素为 $[low, high)$ 区间的随机整数。	Ascend GPU CPU
<code>mindspore.ops.randint_like</code>	返回一个与输入 <code>shape</code> 相同的 <code>tensor</code> , 其元素为 $[low, high)$ 区间的随机整数。	Ascend GPU CPU
<code>mindspore.ops.randn</code>	基于输入的 <code>size</code> 和 <code>dtype</code> , 返回一个 <code>tensor</code> , 其元素为服从标准正态分布的随机数字。	Ascend GPU CPU
<code>mindspore.ops.randn_like</code>	返回一个与输入 <code>shape</code> 相同的 <code>tensor</code> , 其元素为服从标准正态分布的随机数。	Ascend GPU CPU
<code>mindspore.ops.random_gamma</code>	根据伽马分布生成随机数。	CPU
<code>mindspore.ops.random_poisson</code>	从指定均值为 <code>rate</code> 的泊松分布中, 生成指定 <code>shape</code> 的随机样本。	GPU CPU
<code>mindspore.ops.randperm</code>	生成从 0 到 <code>n-1</code> 的整数随机排列。	CPU
<code>mindspore.ops.standard_laplace</code>	根据标准 Laplace (<code>mean=0, lambda=1</code>) 分布生成随机数。	Ascend GPU CPU
<code>mindspore.ops.standard_normal</code>	根据标准正态 (高斯) 随机数分布生成随机数。	Ascend GPU CPU
<code>mindspore.ops.uniform</code>	生成服从均匀分布的随机数。	GPU CPU

mindspore.ops.bernoulli

`mindspore.ops.bernoulli(input, p=0.5, seed=None)`

根据伯努利分布生成随机数（0 或 1）。

$$out_i \sim \text{Bernoulli}(p_i)$$

参数：

- **input** (Tensor) - 输入 tensor。
- **p** (Union[Tensor, float], 可选) - 输出 tensor 中对应位置为 1 的概率。数值范围在 0 到 1 之间。默认 0.5。
- **seed** (Union[int, None], 可选) - 随机种子。默认 None 表示使用时间戳。

返回：

Tensor

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 3])
>>> mindspore.ops.bernoulli(input, p=1.0)
Tensor(shape=[3], dtype=Int64, value= [1, 1, 1])
>>> p = mindspore.tensor([0.0, 1.0, 1.0])
>>> mindspore.ops.bernoulli(input, p)
Tensor(shape=[3], dtype=Int64, value= [0, 1, 1])
```

mindspore.ops.gamma

`mindspore.ops.gamma(shape, alpha, beta, seed=None)`

根据伽马分布生成随机数。

支持广播。

警告：Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **shape** (tuple) - 指定 shape。
- **alpha** (Tensor) - shape 参数。
- **beta** (Tensor) - 逆尺度参数。
- **seed** (int, 可选) - 随机种子，默认 None。

返回：

Tensor

支持平台：

Ascend

样例：

```

>>> import mindspore
>>> # case 1: alpha_shape is (2, 2)
>>> shape = (3, 1, 2)
>>> alpha = mindspore.tensor([[3, 4], [5, 6]], mindspore.float32)
>>> beta = mindspore.tensor([1.0], mindspore.float32)
>>> output = mindspore.ops.gamma(shape, alpha, beta, seed=5)
>>> result = output.shape
>>> print(result)
(3, 2, 2)
>>> # case 2: alpha_shape is (2, 3), so shape is (3, 1, 3)
>>> shape = (3, 1, 3)
>>> alpha = mindspore.tensor([[1, 3, 4], [2, 5, 6]]), mindspore.float32)
>>> beta = mindspore.tensor([1.0], mindspore.float32)
>>> output = mindspore.ops.gamma(shape, alpha, beta, seed=5)
>>> result = output.shape
>>> print(result)
(3, 2, 3)
>>> # case 3: beta_shape is (1, 2), the output is different.
>>> shape = (3, 1, 2)
>>> alpha = mindspore.tensor([[3, 4], [5, 6]], mindspore.float32)
>>> beta = mindspore.tensor([1.0, 2], mindspore.float32)
>>> output = mindspore.ops.gamma(shape, alpha, beta, seed=5)
>>> print(output)
[[[ 2.2132034  5.8855834]
 [ 3.8825176  8.6066265]]
 [[ 3.3981476  7.5805717]
 [ 3.7190282 19.941492 ]]
 [[ 2.9512358  2.5969937]
 [ 3.786061   5.160872 ]]]
>>> # case 4: beta_shape is (2, 1), the output is different.
>>> shape = (3, 1, 2)
>>> alpha = mindspore.tensor([[3, 4], [5, 6]], mindspore.float32)
>>> beta = mindspore.tensor([[1.0], [2.0]], mindspore.float32)
>>> output = mindspore.ops.gamma(shape, alpha, beta, seed=5)
>>> print(output)
[[[ 5.6085486  7.8280783]
 [15.97684 16.116285]]
 [[ 1.8347423  1.713663]
 [ 3.2434065 15.667398]]
 [[ 4.2922077  7.3365674]
 [ 5.3876944 13.159832 ]]]

```

mindspore.ops.laplace

`mindspore.ops.laplace` (*shape, mean, lambda_param, seed=None*)

根据拉普拉斯分布生成随机数。

支持广播。

$$f(x; \cdot) = \frac{1}{2} \exp\left(-\frac{|x - \cdot|}{\cdot}\right),$$

其中 $\cdot$ 为均值, 代表了 *mean*, $\cdot$ 为方差, 代表了 *lambda_param*。

警告: Ascend 后端不支持随机数重现功能, *seed* 参数不起作用。

参数：

- **shape** (tuple) - 生成随机数的 shape。
- **mean** (Tensor) - 分布的均值。
- **lambda_param** (Tensor) - 控制分布方差。拉普拉斯分布的方差等于 *lambda_param* 平方的两倍。
- **seed** (int, 可选) - 随机种子。默认 None 表示使用 0。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> shape = (2, 3)
>>> mean = mindspore.tensor(1.0, mindspore.float32)
>>> lambda_param = mindspore.tensor(1.0, mindspore.float32)
>>> output = mindspore.ops.laplace(shape, mean, lambda_param, seed=5)
>>> print(output.shape)
(2, 3)
```

mindspore.ops.multinomial

mindspore.ops.multinomial (*input*, *num_samples*, *replacement=True*, *seed=None*)

生成一个多项式分布的 **tensor**。

多项式分布是一种概率分布，把二项分布公式推广至多种状态，就得到了多项式分布。在多项式分布中，每个事件都有一个固定的概率，这些概率的和为 1。

本接口的作用是对输入 *input* 进行 *num_samples* 次抽样，输出的 **tensor** 则为每一次抽样时输入 **tensor** 的索引，其中 *input* 中的值为每次抽样取到对应索引的概率。

这里我们给一个相对极端的用例方便理解，我们给定一个输入概率值 **tensor**，值为 $[90/100, 10/100, 0]$ ，代表我们一共可以对三个索引进行抽样，分别为索引 0、索引 1 和索引 2，它们被抽中的概率分别为 90%、10% 和 0%，我们对其进行 *n* 次抽样，抽样的结果则为多项式分布的计算结果，计算结果长度与抽样次数一致。

在样例代码 case 1 中，我们对其进行两次不放回抽样 (*replacement* 为 False)，由于每次抽样抽到索引 0 的概率为 90%，因此抽到的结果第一次大概率是抽到索引 0，由于抽到索引 2 的概率为 0，因此抽样两次结果不可能出现索引 2，那第二次结果一定是索引 1，因此结果序列为 $[0, 1]$ 。

在样例代码 case 2 中，我们对其进行 10 次放回抽样 (*replacement* 为 True)，可以看到计算结果中大概有 90% 的抽样结果为抽到索引 0，符合预期。

在样例代码 case 3 中，我们将输入扩展为二维，可以看到抽样结果在每一个维度中的结果也符合抽样预期。

说明：输入的行不要求和为 1（当使用值作为权重的情况下），但必须是非负的、有限的，并且和不能为 0。在使用值作为权重的情况下，可以理解为对输入沿最后一维进行了归一化操作，以此保证概率和为 1。

警告：Ascend 后端不支持随机数重现功能，*seed* 参数不起作用。

参数:

- **input** (Tensor) - 包含概率值的 tensor。
- **num_samples** (int) - 采样的次数。
- **replacement** (bool, 可选) - 是否放回, 默认 True。
- **seed** (int, 可选) - 随机种子, 默认 None。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: The output is random, and the length of the output is the same as num_sample.
>>> # replacement is False.
>>> input1 = mindspore.tensor([90 / 100, 10 / 100, 0])
>>> input2 = mindspore.tensor([90, 10, 0])
>>> # input1 and input2 have the same meaning.
>>> mindspore.ops.multinomial(input1, 2, replacement=False)
Tensor(shape=[2], dtype=Int32, value= [0, 1])
>>> mindspore.ops.multinomial(input2, 2, replacement=False)
Tensor(shape=[2], dtype=Int32, value= [1, 0])
>>>
>>> # case 2: The output is random, and the length of the output is the same as num_sample.
>>> # replacement is True.
>>> mindspore.ops.multinomial(input1, 10)
Tensor(shape=[10], dtype=Int32, value= [0, 0, 1, 0, 0, 0, 0, 0, 0, 0])
>>>
>>> # case 3: The output is random, and the length of the output is the same as num_sample.
>>> # replacement is True.
>>> # rank is 2
>>> input3 = mindspore.tensor([[90, 10, 0], [10, 90, 0]], mindspore.float32)
>>> output = mindspore.ops.multinomial(input3, 10)
>>> print(output)
[[0 0 0 0 0 0 0 0 0 0]
 [1 0 1 1 1 1 1 1 1 1]]
```

mindspore.ops.multinomial_with_replacement

mindspore.ops.multinomial_with_replacement (*x*, *seed*, *offset*, *numsamples*, *replacement=False*)
 生成一个多项式分布的 tensor。

说明:

- 输入的行不要求和为 1 (在这种情况下, 使用值作为权重), 但必须是非负的、有限的, 并且具有非零和。
- *seed* 如果为 -1 且 *offset* 为 0, 则随机数生成器将使用随机种子。

参数:

- **x** (Tensor) - 一维或二维的输入 tensor, 包含概率的累积和。

- **seed** (int) - 随机种子。
- **offset** (int) - 偏移量。
- **numsamples** (int) - 采样的次数。
- **replacement** (bool, 可选) - 是否放回。默认 `False`。

返回：

Tensor

支持平台：

CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[0., 9., 4., 0.]], mindspore.float32)
>>> mindspore.ops.multinomial_with_replacement(x, 2, 5, 2, True)
Tensor(shape=[1, 2], dtype=Int64, value=
[[1, 1]])
```

`mindspore.ops.rand`

`mindspore.ops.rand(*size, dtype=None, seed=None)`

基于输入的 *size* 和 *dtype*, 返回一个 `tensor`, 其元素为服从均匀分布的 $[0, 1)$ 区间的随机数。

警告：Ascend 后端不支持随机数重现功能，*seed* 参数不起作用。

参数：

- **size** (`Union[int, tuple(int), list(int)]`) - 输出 `tensor` 的 `shape`。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型，默认 `None`。
- **seed** (int, 可选) - 随机种子，必须大于或等于 0。默认 `None`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> print(mindspore.ops.rand((2, 3)))
[[4.1702199e-01 9.9718481e-01 7.2032452e-01]
 [9.3255734e-01 1.1438108e-04 1.2812445e-01]]
```

mindspore.ops.rand_like

`mindspore.ops.rand_like(input, seed=None, *, dtype=None)`

返回一个与输入 `shape` 相同的 `tensor`，其元素为服从均匀分布的 `[0, 1)` 区间的随机数。

警告：Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **input** (`Tensor`) - 输入 `tensor`。
- **seed** (`int`，可选) - 随机种子，必须大于或等于 0。默认 `None`。

关键字参数：

- **dtype** (`mindspore.dtype`，可选) - 指定数据类型。默认 `None`。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> a = mindspore.tensor([[2, 3, 4], [1, 2, 3]])
>>> print(mindspore.ops.rand_like(a, dtype=mindspore.float32))
[[4.1702199e-01 9.9718481e-01 7.2032452e-01]
 [9.3255734e-01 1.1438108e-04 1.2812445e-01]]
```

mindspore.ops.randint

`mindspore.ops.randint(low, high, size, seed=None, *, dtype=None)`

返回一个 `tensor`，其元素为 `[low, high)` 区间的随机整数。

警告：Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **low** (`int`) - 随机区间的起始值。
- **high** (`int`) - 随机区间的末尾值。
- **size** (`tuple`) - 输出 `tensor` 的 `shape`。
- **seed** (`int`，可选) - 随机种子，必须大于或等于 0。默认 `None`。

关键字参数：

- **dtype** (`mindspore.dtype`，可选) - 指定数据类型。默认 `None`。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> print(mindspore.ops.randint(1, 10, (2,3)))
[[4 9 7]
 [9 1 2]]
```

mindspore.ops.randint_like

`mindspore.ops.randint_like(input, low, high, *, dtype=None, seed=None)`
 返回一个与输入 shape 相同的 tensor，其元素为 $[low, high)$ 区间的随机整数。

警告：Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **input** (Tensor) - 输入 tensor。
- **low** (int) - 随机区间的起始值。
- **high** (int) - 随机区间的末尾值。
- **seed** (int, 可选) - 随机种子，必须大于或等于 0。默认 None。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> a = mindspore.tensor([[1, 2, 3], [3, 2, 1]])
>>> print(mindspore.ops.randint_like(a, 1, 10))
[[4 9 7]
 [9 1 2]]
```

mindspore.ops.randn

`mindspore.ops.randn(*size, dtype=None, seed=None)`
 基于输入的 `size` 和 `dtype`，返回一个 tensor，其元素为服从标准正态分布的随机数字。

警告：Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **size** (Union[int, tuple(int), list(int)]) - 输出 tensor 的 shape。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 指定数据类型。默认 None。
- **seed** (int, 可选) - 随机种子，必须大于或等于 0。默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> print(mindspore.ops.randn((2, 2)))
[[ 0.30639967 -0.42438635]
 [-0.4287376  1.3054721 ]]
```

mindspore.ops.randn_like

`mindspore.ops.randn_like(input, seed=None, *, dtype=None)`

返回一个与输入 shape 相同的 tensor，其元素为服从标准正态分布的随机数。

警告：Ascend 后端不支持随机数重现功能，*seed* 参数不起作用。

参数：

- **input** (Tensor) - 输入 tensor。
- **seed** (int, 可选) - 随机种子，必须大于或等于 0。默认 None。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 指定数据类型。默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> a = mindspore.tensor([[1, 2, 3], [4, 5, 6]])
>>> print(mindspore.ops.randn_like(a, dtype=mindspore.float32))
[[ 0.30639967 -0.42438635 -0.20454668]
 [-0.4287376  1.3054721  0.64747655]]
```

mindspore.ops.random_gamma

`mindspore.ops.random_gamma(shape, alpha, seed=None)`

根据伽马分布生成随机数。

参数：

- **shape** (Tensor) - 指定生成随机数的 shape。
- **alpha** (Tensor) - α 分布的参数。必须是非负数。
- **seed** (int, 可选) - 随机种子，必须为非负数。默认 None。

返回：

Tensor。形状为 `mindspore.ops.concat([shape, alpha.shape], axis=0)`。数据类型和 `alpha` 一致。

支持平台：

CPU

样例：

```
>>> import mindspore
>>> shape = mindspore.tensor([7, 5], mindspore.int32)
>>> alpha = mindspore.tensor([0.5, 1.5], mindspore.float32)
>>> output = mindspore.ops.random_gamma(shape, alpha, seed=5)
>>> print(output.shape, output.dtype)
(7, 5, 2) Float32
```

mindspore.ops.random_poisson

`mindspore.ops.random_poisson(shape, rate, seed=None, dtype=mstype.float32)`

从指定均值为 `rate` 的泊松分布中，生成指定 `shape` 的随机样本。

$$P(i) = \frac{\exp(-)^i}{i!}$$

警告： Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **shape** (Tensor) - 指定生成随机数的 shape。一维整型 tensor。
- **rate** (Tensor) - 泊松分布的参数，表示泊松分布的均值，同时也是分布的方差。
- **seed** (int, 可选) - 随机数种子。必须是一个非负整数，默认 None。
- **dtype** (mindspore.dtype) - 指定数据类型。默认 `mstype.float32`。

返回：

Tensor，形状为 `mindspore.ops.concat([shape, rate.shape], axis=0)`。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: 1-D shape, 2-D rate, float64 output
>>> shape = mindspore.tensor([2, 2], mindspore.int64)
>>> rate = mindspore.tensor([[5.0, 10.0], [5.0, 1.0]], mindspore.float32)
>>> output = mindspore.ops.random_poisson(shape, rate, seed=5, dtype=mindspore.float64)
>>> print(output.shape, output.dtype)
(2, 2, 2, 2) Float64
>>> # case 2: 1-D shape, scalar rate, int64 output
>>> shape = mindspore.tensor([2, 2], mindspore.int64)
>>> rate = mindspore.tensor(5.0, mindspore.float64)
>>> output = mindspore.ops.random_poisson(shape, rate, seed=5, dtype=mindspore.int64)
>>> print(output.shape, output.dtype)
(2, 2) Int64
```

mindspore.ops.randperm

`mindspore.ops.randperm(n, seed=0, offset=0, dtype=mstype.int64)`

生成从 0 到 $n-1$ 的整数随机排列。

警告:

- 这是一个实验性 API，后续可能修改或删除。
- Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数:

- **n** (`Union[Tensor, int]`) - 输入上界（不包含）。
- **seed** (`int`, 可选) - 随机种子。默认 0。当 `seed` 为 -1 时，`offset` 为 0，随机数由时间决定。
- **offset** (`int`, 可选) - 偏移量。生成随机数，优先级高于随机种子。必须是非负数，默认 0。
- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 `mstype.int64`。

返回:

`Tensor`, `shape` 由参数 n 决定。

支持平台:

CPU

样例:

```
>>> import mindspore
>>> n, seed, offset = 4, 0, 0
>>> output = mindspore.ops.randperm(n, seed, offset, dtype=mindspore.int64)
>>> print(output)
[0 2 1 3]
```

mindspore.ops.standard_laplace

`mindspore.ops.standard_laplace(shape, seed=None)`

根据标准 Laplace (mean=0, lambda=1) 分布生成随机数。

$$f(x) = \frac{1}{2} \exp(-|x|)$$

警告: Ascend 后端不支持随机数重现功能, *seed* 参数不起作用。

参数:

- **shape** (Union[tuple, Tensor]) - 指定生成随机数的 shape。
- **seed** (int, 可选) - 随机数种子。默认 None。

返回:

Tensor

异常:

- **ValueError** - *shape* 为 tuple 时, 包含非正的元素。
- **ValueError** - *shape* 为秩不等于 1 的 tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> shape = (4, 4)
>>> output = mindspore.ops.standard_laplace(shape, seed=5)
>>> print(f'output shape is {output.shape}')
output shape is (4, 4)
```

mindspore.ops.standard_normal

`mindspore.ops.standard_normal(shape, seed=None)`

根据标准正态 (高斯) 随机数分布生成随机数。

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{\left(-\frac{x^2}{2}\right)}$$

警告: Ascend 后端不支持随机数重现功能, *seed* 参数不起作用。

参数:

- **shape** (Union[tuple, Tensor]) - 指定生成随机数的 shape。
- **seed** (int, 可选) - 随机数种子。默认 None。

返回:

Tensor

异常:

- **ValueError** - *shape* 为 tuple 时, 包含非正的元素。
- **ValueError** - *shape* 为秩不等于 1 的 tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> shape = (4, 4)
>>> output = mindspore.ops.standard_normal(shape, seed=5)
>>> print(f'output shape is {output.shape}')
output shape is (4, 4)
```

mindspore.ops.uniform

`mindspore.ops.uniform(shape, minval, maxval, seed=None, dtype=mstype.float32)`
生成服从均匀分布的随机数。

说明: 广播后, 任意位置上 tensor 的最小值必须小于最大值。

参数:

- **shape** (Union[tuple, Tensor]) - 指定生成随机数的 shape。
- **minval** (Tensor) - 指定生成随机值的最小值。
- **maxval** (Tensor) - 指定生成随机值的最大值。
- **seed** (int) - 随机数种子。默认 None。
- **dtype** (mindspore.dtype) - 指定数据类型。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> # For discrete uniform distribution, only one number is allowed for both minval and
    ↪maxval:
>>> shape = (4, 2)
>>> minval = mindspore.tensor(1, mindspore.int32)
>>> maxval = mindspore.tensor(2, mindspore.int32)
>>> output = mindspore.ops.uniform(shape, minval, maxval, seed=5, dtype=mindspore.int32)
>>>
>>> # For continuous uniform distribution, minval and maxval can be multi-dimentional:
>>> shape = (3, 1, 2)
```

(下页继续)

(续上页)

```
>>> minval = mindspore.tensor([[3, 4], [5, 6]], mindspore.float32)
>>> maxval = mindspore.tensor([8.0, 10.0], mindspore.float32)
>>> output = mindspore.ops.uniform(shape, minval, maxval, seed=5)
>>> result = output.shape
>>> print(result)
(3, 2, 2)
```

2.1.3 Array 操作

接口名	概述	支持平台
<code>mindspore.ops.argwhere</code>	返回一个 <code>tensor</code> ，包含输入 <code>tensor</code> 所有非零元素的位置。	Ascend GPU CPU
<code>mindspore.ops.batch_to_space_nd</code>	用块划分批次维度，并将这些块交错回空间维度。	Ascend GPU CPU
<code>mindspore.ops.bincount</code>	统计一个输入 <code>tensor</code> 中每个非负整数出现的次数。	Ascend GPU CPU
<code>mindspore.ops.block_diag</code>	基于输入 <code>tensor</code> 创建块对角矩阵。	Ascend GPU CPU
<code>mindspore.ops.broadcast_to</code>	将输入的 <code>shape</code> 广播到目标 <code>shape</code> 。	Ascend GPU CPU
<code>mindspore.ops.cat</code>	在指定轴上拼接输入 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.channel_shuffle</code>	将 <code>shape</code> 为 $(*, C, H, W)$ 的 <code>tensor</code> 的通道划分成 g 组，并按如下方式重新排列 $(*, \frac{C}{g}, g, H * W)$ ，同时在最终输出中保持原始 <code>tensor</code> 的 <code>shape</code> 。	Ascend CPU
<code>mindspore.ops.chunk</code>	沿着指定轴将输入 <code>tensor</code> 切分成多个子 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.column_stack</code>	将多个 <code>tensor</code> 沿着水平方向维度堆叠成一个 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.concat</code>	<code>mindspore.ops.cat()</code> 的别名。	
<code>mindspore.ops.conj</code>	逐元素计算输入 <code>tensor</code> 的共轭。	Ascend GPU CPU
<code>mindspore.ops.count_nonzero</code>	计算输入 <code>tensor</code> 指定轴上的非零元素的数量。	Ascend GPU CPU
<code>mindspore.ops.deeppcopy</code>	返回输入 <code>tensor</code> 的深拷贝。	Ascend GPU CPU
<code>mindspore.ops.diag</code>	返回一个用输入 <code>tensor</code> 作为对角线，其余元素为 0 的 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.diagflat</code>	如果 <code>input</code> 是一维 <code>tensor</code> ，则返回 <code>input</code> 作为对角线的二维 <code>tensor</code> ，如果 <code>input</code> 是大于等于二维的 <code>tensor</code> ，则将 <code>input</code> 展开后作为对角线的二维 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.diagonal</code>	返回输入 <code>tensor</code> 在指定维度上的对角线视图。	Ascend GPU CPU
<code>mindspore.ops.diagonal_scatter</code>	将源 <code>tensor</code> 的值嵌入到输入 <code>tensor</code> 的对角线中（基于 <code>dim1</code> 和 <code>dim2</code> 指定的二维平面）。	Ascend GPU CPU
<code>mindspore.ops.diag_embed</code>	创建一个 <code>tensor</code> ，其特定二维平面（由 <code>dim1</code> 和 <code>dim2</code> 指定）的对角线由输入 <code>tensor</code> 填充，其余位置填充为 0。	Ascend GPU CPU
<code>mindspore.ops.dyn_shape</code>	返回输入 <code>tensor</code> 的 <code>shape</code> 。	Ascend GPU CPU
<code>mindspore.ops.dsplitt</code>	沿第三轴分割输入 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.dstack</code>	将多个 <code>tensor</code> 沿着第三维度进行堆叠。	Ascend GPU CPU
<code>mindspore.ops.einsum</code>	基于爱因斯坦求和约定（Einsum）符号，沿指定维度计算输入 <code>tensor</code> 元素的乘积之和。	GPU
<code>mindspore.ops.expand_dims</code>	为输入 <code>tensor</code> 新增额外的轴。	Ascend GPU CPU
<code>mindspore.ops.flip</code>	沿指定维度翻转 <code>tensor</code> 中元素。	Ascend GPU CPU

[下页继续](#)

表 3 - 续上页

<code>mindspore.ops.fliplr</code>	向左/向右翻转输入 tensor 。	Ascend GPU CPU
<code>mindspore.ops.flipud</code>	向上/向下翻转输入 tensor 。	Ascend GPU CPU
<code>mindspore.ops.gather</code>	返回输入 tensor 在指定轴及指定索引上对应元素的切片。	Ascend GPU CPU
<code>mindspore.ops.gather_d</code>	获取指定轴的元素。	Ascend GPU CPU
<code>mindspore.ops.gather_elements</code>	根据指定维度和索引获取元素。	Ascend GPU CPU
<code>mindspore.ops.gather_nd</code>	根据指定索引获取输入 tensor 的切片。	Ascend GPU CPU
<code>mindspore.ops.hsplit</code>	将输入 tensor 水平分割成多个子 tensor 。	Ascend GPU CPU
<code>mindspore.ops.hstack</code>	将多个 tensor 沿着水平方向堆叠。	Ascend GPU CPU
<code>mindspore.ops.index_add</code>	按照指定轴和索引将输入 <i>y</i> 的元素加到输入 <i>x</i> 中。	Ascend GPU CPU
<code>mindspore.ops.index_fill</code>	按照指定轴和索引用输入 <i>value</i> 填充输入 <i>x</i> 的元素。	Ascend GPU CPU
<code>mindspore.ops.index_select</code>	根据指定轴和索引对输入 tensor 进行选取, 返回一个新 tensor 。	Ascend GPU CPU
<code>mindspore.ops.inplace_add</code>	根据索引将 <i>x</i> 加 <i>v</i> 。	Ascend GPU CPU
<code>mindspore.ops.inplace_index_add</code>	根据指定轴和索引在输入 <i>var</i> 中的对应位置加 <i>updates</i> 。	Ascend CPU
<code>mindspore.ops.inplace_sub</code>	根据索引将 <i>x</i> 减 <i>v</i> 。	Ascend GPU CPU
<code>mindspore.ops.inplace_update</code>	根据索引将 <i>x</i> 更新为 <i>v</i> 。	GPU CPU
<code>mindspore.ops.is_nonzero</code>	判断输入 tensor 是否包含 0 或 <code>False</code> 。	Ascend GPU CPU
<code>mindspore.ops.masked_fill</code>	在掩码为 <code>True</code> 位置填充指定值。	Ascend GPU CPU
<code>mindspore.ops.masked_select</code>	返回一个一维 tensor , 它根据掩码对输入 tensor 进行索引。	Ascend GPU CPU
<code>mindspore.ops.meshgrid</code>	从一维输入生成多维坐标网格。	Ascend GPU CPU
<code>mindspore.ops.moveaxis</code>	将输入 tensor 的两个轴调换位置。	Ascend GPU CPU
<code>mindspore.ops.movedim</code>	将输入 tensor 的两个维度调换位置。	Ascend GPU CPU
<code>mindspore.ops.narrow</code>	沿指定轴 <i>axis</i> , 从 <i>start</i> 位置截取长度为 <i>length</i> 的 tensor 。	Ascend GPU CPU
<code>mindspore.ops.nan_to_num</code>	将 <i>input</i> 中的 <code>NaN</code> 、正无穷大和负无穷大值分别替换为 <i>nan</i> 、 <i>posinf</i> 和 <i>neginf</i> 指定的值。	Ascend CPU
<code>mindspore.ops.nanmean</code>	忽略 <code>NaN</code> 值, 按指定轴计算输入 tensor 的平均值。	Ascend GPU CPU
<code>mindspore.ops.nanmedian</code>	忽略 <code>NaN</code> 值, 按指定轴计算输入 tensor 的中值和索引。	CPU
<code>mindspore.ops.nansum</code>	忽略 <code>NaN</code> 值, 按指定轴计算输入 tensor 的和。	Ascend GPU CPU
<code>mindspore.ops.normal</code>	返回符合正态 (高斯) 分布的随机 tensor 。	Ascend GPU CPU
<code>mindspore.ops.nonzero</code>	返回所有非零元素下标位置。	Ascend GPU CPU
<code>mindspore.ops.numel</code>	返回 tensor 的元素的总数量。	Ascend GPU CPU
<code>mindspore.ops.permute</code>	按照指定轴重排输入 tensor 。	Ascend GPU CPU
<code>mindspore.ops.population_count</code>	计算输入 tensor 中每个元素二进制的 1 比特数量。	Ascend GPU CPU
<code>mindspore.ops.rank</code>	返回输入 tensor 的秩。	Ascend GPU CPU
<code>mindspore.ops.repeat_elements</code>	在指定轴上复制输入 tensor 的元素, 类似 <code>mindspore.numpy.repeat()</code> 的功能。	Ascend GPU CPU
<code>mindspore.ops.repeat_interleave</code>	在指定轴上复制输入 tensor 的元素, 类似 <code>mindspore.numpy.repeat()</code> 。	Ascend GPU CPU
<code>mindspore.ops.reshape</code>	按指定 <i>shape</i> , 对输入 tensor 进行重排。	Ascend GPU CPU
<code>mindspore.ops.reverse_sequence</code>	对输入序列进行部分反转。	Ascend GPU CPU
<code>mindspore.ops.roll</code>	按维度移动 tensor 的元素。	Ascend GPU

下页继续

表 3 - 续上页

<code>mindspore.ops.row_stack</code>	<code>mindspore.ops.vstack()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.scatter</code>	根据指定索引将 <code>src</code> 中的值更新到 <code>input</code> 中并返回输出。	Ascend GPU CPU
<code>mindspore.ops.scatter_nd</code>	根据指定的索引将更新值散布到新 <code>tensor</code> 上。	Ascend GPU CPU
<code>mindspore.ops.select</code>	根据条件判断 <code>tensor</code> 中的元素的值, 来决定输出中的相应元素是从 <code>input</code> (如果元素值为 <code>True</code>) 还是从 <code>other</code> (如果元素值为 <code>False</code>) 中选择。	Ascend GPU CPU
<code>mindspore.ops.select_scatter</code>	将 <code>src</code> 中的值散布到 <code>input</code> 指定轴 <code>axis</code> 的指定位置 <code>index</code> 上。	Ascend GPU CPU
<code>mindspore.ops.sequence_mask</code>	返回一个表示每个单元的前 <code>N</code> 个位置的掩码 <code>tensor</code> , 内部元素数据类型为 <code>bool</code> 。	GPU CPU
<code>mindspore.ops.shape</code>	返回输入 <code>tensor</code> 的 <code>shape</code> 。	Ascend GPU CPU
<code>mindspore.ops.shuffle</code>	沿着第一维随机打乱 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.size</code>	计算输入 <code>tensor</code> 的元素总数。	Ascend GPU CPU
<code>mindspore.ops.slice</code>	根据指定 <code>shape</code> 对输入 <code>tensor</code> 进行切片。	Ascend GPU CPU
<code>mindspore.ops.slice_scatter</code>	沿指定轴将源 <code>tensor</code> 嵌入到切片后的目标 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.sort</code>	对输入 <code>tensor</code> 指定轴的元素进行排序。	Ascend GPU CPU
<code>mindspore.ops.space_to_batch_nd</code>	将空间维度分块, 并在批次维度重排 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.sparse_segment_mean</code>	计算输入张量稀疏段的平均值。	GPU CPU
<code>mindspore.ops.split</code>	沿指定轴将输入 <code>tensor</code> 切分成多个子 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.squeeze</code>	删除输入 <code>tensor</code> 中长度为 1 的轴。	Ascend GPU CPU
<code>mindspore.ops.stack</code>	在指定轴上对输入的 <code>tensors</code> 进行堆叠。	Ascend GPU CPU
<code>mindspore.ops.strided_slice</code>	对输入 <code>tensor</code> 根据步长和索引进行切片提取。	Ascend GPU CPU
<code>mindspore.ops.sum</code>	计算 <code>tensor</code> 在指定维度上元素的和。	Ascend GPU CPU
<code>mindspore.ops.swapaxes</code>	交换 <code>tensor</code> 的两个轴。	Ascend GPU CPU
<code>mindspore.ops.swapdims</code>	交换 <code>tensor</code> 的两个维度。	Ascend GPU CPU
<code>mindspore.ops.tensor_scatter_add</code>	返回一个新 <code>tensor</code> , 根据指定索引和更新值对 <code>input_x</code> 进行加法更新。	Ascend GPU CPU
<code>mindspore.ops.tensor_scatter_div</code>	返回一个新 <code>tensor</code> , 根据指定索引和更新值对 <code>input_x</code> 进行除法更新。	GPU CPU
<code>mindspore.ops.tensor_scatter_max</code>	返回一个新 <code>tensor</code> , 根据指定索引和更新值对 <code>input_x</code> 进行最大值更新。	GPU CPU
<code>mindspore.ops.tensor_scatter_min</code>	返回一个新 <code>tensor</code> , 根据指定索引和更新值对 <code>input_x</code> 进行最小值更新。	Ascend GPU CPU
<code>mindspore.ops.tensor_scatter_mul</code>	根据指定索引和更新值对 <code>input_x</code> 进行乘法更新。	GPU CPU
<code>mindspore.ops.tensor_scatter_sub</code>	返回一个新 <code>tensor</code> , 根据指定索引和更新值对 <code>input_x</code> 进行减法更新。	Ascend GPU CPU
<code>mindspore.ops.tensor_scatter_elements</code>	返回一个新 <code>tensor</code> , 根据指定索引和更新值对 <code>input_x</code> 中的元素进行指定操作 (替换或相加)。	Ascend GPU CPU
<code>mindspore.ops.tensor_split</code>	根据指定索引或份数, 将输入 <code>tensor</code> 拆分成多个子 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.tile</code>	通过复制 <code>dims</code> 次输入 <code>tensor</code> 中的元素来创建新 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.tril</code>	将指定对角线上方的元素设置为 0。	Ascend GPU CPU
<code>mindspore.ops.triu</code>	将指定对角线下方的元素设置为 0。	Ascend GPU CPU
<code>mindspore.ops.transpose</code>	根据指定的维度排列顺序对输入 <code>tensor</code> 进行维度转置。	Ascend GPU CPU

下页继续

表 3 – 续上页

<code>mindspore.ops.unbind</code>	移除 tensor 的指定维度，返回一个沿该维度所有切片的元组。	Ascend GPU CPU
<code>mindspore.ops.unique</code>	对输入 tensor 中元素去重。	Ascend GPU CPU
<code>mindspore.ops.unique_consecutive</code>	对输入 tensor 中每组连续且重复的元素去重，仅保留首个元素。	Ascend GPU CPU
<code>mindspore.ops.unsorted_segment_max</code>	分段计算输入 tensor 的最大值。	Ascend GPU CPU
<code>mindspore.ops.unsorted_segment_min</code>	分段计算输入 tensor 的最小值。	Ascend GPU CPU
<code>mindspore.ops.unsorted_segment_prod</code>	分段计算输入 tensor 的乘积。	Ascend GPU CPU
<code>mindspore.ops.unsorted_segment_sum</code>	分段计算输入 tensor 的和。	Ascend GPU CPU
<code>mindspore.ops.unsqueeze</code>	在输入 tensor 的指定维度上增加一维。	Ascend GPU CPU
<code>mindspore.ops.unstack</code>	按指定轴对输入 tensor 进行拆分。	Ascend GPU CPU
<code>mindspore.ops.view_as_real</code>	返回一个最后一维大小为 2 的实数 tensor ，由输入 tensor 中复数元素的实部和虚部组成。	GPU CPU
<code>mindspore.ops.vsplit</code>	根据 <i>indices_or_sections</i> 将至少有两维的输入 tensor 垂直分割成多个子 tensor 。	Ascend GPU CPU
<code>mindspore.ops.vstack</code>	将多个 tensor 沿着垂直方向进行堆叠。	Ascend GPU CPU
<code>mindspore.ops.where</code>	返回一个 tensor ，其中元素根据 <i>condition</i> 从 <i>input</i> 或 <i>other</i> 中选择。	Ascend GPU CPU
<code>mindspore.ops.cross</code>	按指定维度计算两个输入 tensor 的叉积。	Ascend CPU
<code>mindspore.ops.renorm</code>	返回一个 tensor ，其中按指定轴计算每个子 tensor 的 <i>p</i> 范数小于等于 <i>maxnorm</i> 。	Ascend GPU CPU

mindspore.ops.argwhere

mindspore.ops.**argwhere** (*input*)
返回一个 **tensor**，包含输入 **tensor** 所有非零元素的位置。

- 参数：
- **input** (Tensor) - 输入 **tensor**。

返回：
二维 **tensor**

支持平台：
Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[[[1, 0], [-5, 0]]], mindspore.int32)
>>> output = mindspore.ops.argwhere(x)
>>> print(output)
[[0 0 0]
 [0 1 0]]
```

mindspore.ops.batch_to_space_nd

`mindspore.ops.batch_to_space_nd(input_x, block_shape, crops)`

用块划分批次维度，并将这些块交错回空间维度。

此函数会将批次维度 N 划分为具有 $block\_shape$ 的块，即输出张量的 N 维度是划分后对应的块数。输出张量的 $w_1, \dots, w_M$ 维度是原始的 $w_1, \dots, w_M$ 维度和 $block\_shape$ 的乘积从维度裁剪给定。若输入的 $shape$ 为 $(n, c_1, \dots, c_k, w_1, \dots, w_M)$ ，则输出的 $shape$ 为 $(n', c_1, \dots, c_k, w'_1, \dots, w'_M)$ 。其中，

$$\begin{aligned} n' &= n / (block\_shape[0] * \dots * block\_shape[M - 1]) \\ w'_i &= w_i * block\_shape[i - 1] - crops[i - 1][0] - crops[i - 1][1] \end{aligned}$$

参数：

- **input_x** (Tensor) - 输入 tensor，必须大于或者等于二维 (Ascend 平台必须为四维)。批次维度需能被 $block\_shape$ 整除。
- **block_shape** (Union[list(int), tuple(int), int]) - 分割批次维度的块的数量，取值需大于或者等于 1。如果 $block\_shape$ 为 list 或者 tuple，其长度 M 为空间维度的长度。如果 $block\_shape$ 为整数，那么所有空间维度分割的个数均为 $block\_shape$ 。在 Ascend 后端 M 必须为 2。
- **crops** (Union[list(int), tuple(int)]) - 空间维度的裁剪大小，包含 M 个长度为 2 的 list，取值需大于或等于 0。 $crops[i]$ 为对空间维度 i 的填充，对应输入 tensor 的维度 $i + offset$ ， $offset$ 为空间维度在输入 tensor 维度中的偏移量，其中 $offset = N - M$ ， N 是输入维度数。同时要求 $input\_shape[i + offset] * block\_shape[i] > crops[i][0] + crops[i][1]$ 。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> block_shape = [2, 2]
>>> crops = [[0, 0], [0, 0]]
>>> input_x = mindspore.tensor([[[[1]]], [[2]]], [[[3]]], [[[4]]], mindspore.float32)
>>> output = mindspore.ops.batch_to_space_nd(input_x, block_shape, crops)
>>> print(output)
[[[1.  2.]
  [3.  4.]]]
```

mindspore.ops.bincount

`mindspore.ops.bincount(input, weights=None, minlength=0)`

统计一个输入 tensor 中每个非负整数出现的次数。

如果不指定 $minlength$ ，输出 tensor 的长度为 $\max(input) + 1$ 。如果指定 $minlength$ ，则输出 tensor 的长度 $\max([\max(input) + 1, minlength])$ 。

如果指定了 $weights$ ，对输出的结果进行加权处理，如果 n 是输入 tensor 中索引 i 处的值，即 $out[n] += weight[i]$ 而不是 $out[n] += 1$ 。

说明：如果 $input$ 含有负数元素，则结果未定义。

参数:

- **input** (Tensor) - 一维输入 tensor。
- **weights** (Tensor, 可选) - 权重。默认 None。
- **minlength** (int, 可选) - 输出 tensor 的最小长度。默认 0。

返回:

Tensor, 如果输入为非空, 输出 shape 为 $[\max(\text{input})+1]$ 的 tensor, 否则 shape 为 [0]。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([2, 4, 1, 0, 0], dtype=mindspore.int64)
>>> print(mindspore.ops.bincount(x, minlength=7))
[2. 1. 1. 0. 1. 0. 0.]
>>> weights = mindspore.tensor([0, 0.25, 0.5, 0.75, 1], dtype=mindspore.float32)
>>> print(mindspore.ops.bincount(x, weights=weights))
[1.75 0.5 0. 0. 0.25]
```

mindspore.ops.block_diag

mindspore.ops.block_diag (*inputs)

基于输入 tensor 创建块对角矩阵。

参数:

- **inputs** (Tensor) - 一个或多个 tensor, tensor 的维度应该为 0、1 或 2。

返回:

二维 tensor。所有输入 tensor 按顺序排列, 使其左上角和右下角对角线相邻, 其他所有元素都置零。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x1 = mindspore.tensor([[4], [3], [2]], mindspore.int32)
>>> x2 = mindspore.tensor([7, 6, 5], mindspore.int32)
>>> x3 = mindspore.tensor(1, mindspore.int32)
>>> x4 = mindspore.tensor([[5, 4, 3], [2, 1, 0]], mindspore.int32)
>>> x5 = mindspore.tensor([[8, 7], [7, 8]], mindspore.int32)
>>> out = mindspore.ops.block_diag(x1, x2, x3, x4, x5)
>>> print(out.asnumpy())
[[4 0 0 0 0 0 0 0 0 0]
 [3 0 0 0 0 0 0 0 0 0]
 [2 0 0 0 0 0 0 0 0 0]
 [0 7 6 5 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0 0 0]
 [0 0 0 0 0 5 4 3 0 0]
 [0 0 0 0 0 2 1 0 0 0]
 [0 0 0 0 0 0 0 8 7]
 [0 0 0 0 0 0 0 7 8]]
```

mindspore.ops.broadcast_to

`mindspore.ops.broadcast_to(input, shape)`

将输入的 `shape` 广播到目标 `shape`。输入维度必须小于等于目标维度，设输入 `shape` 为 $(x_1, x_2, \dots, x_m)$ ，目标 `shape` 为 $(*, y_1, y_2, \dots, y_m)$ ，其中 $*$ 为任意额外的维度。广播规则如下：

依次比较 x_m 与 y_m 、 x_{m-1} 与 y_{m-1} 、 $\dots$ 、 x_1 与 y_1 的值确定是否可以广播以及广播后输出 `shape` 对应维的值。

- 如果相等，则这个值即为目标 `shape` 该维的值。比如说输入 `shape` 为 $(2, 3)$ ，目标 `shape` 为 $(2, 3)$ ，则输出 `shape` 为 $(2, 3)$ 。
- 如果不相等，分以下三种情况：
 - 情况一：如果目标 `shape` 该维的值为 -1，则输出 `shape` 该维的值为对应输入 `shape` 该维的值。比如说输入 `shape` 为 $(3, 3)$ ，目标 `shape` 为 $(-1, 3)$ ，则输出 `shape` 为 $(3, 3)$ ；
 - 情况二：如果目标 `shape` 该维的值不为 -1，但是输入 `shape` 该维的值为 1，则输出 `shape` 该维的值为目标 `shape` 该维的值。比如说输入 `shape` 为 $(1, 3)$ ，目标 `shape` 为 $(8, 3)$ ，则输出 `shape` 为 $(8, 3)$ ；
 - 情况三：如果两个 `shape` 对应值不满足以上情况则说明不支持由输入 `shape` 广播到目标 `shape`。

至此输出 `shape` 后面 m 维就确定好了，现在看一下前面 $*$ 维，有以下两种情况：

- 如果额外的 $*$ 维中不含有 -1，则输入 `shape` 从低维度补充维度使之与目标 `shape` 维度一致，比如说目标 `shape` 为 $(3, 1, 4, 1, 5, 9)$ ，输入 `shape` 为 $(1, 5, 9)$ ，则输入 `shape` 增维变成 $(1, 1, 1, 1, 5, 9)$ ，根据上面提到的情况二可以得出输出 `shape` 为 $(3, 1, 4, 1, 5, 9)$ ；
- 如果额外的 $*$ 维中含有 -1，说明此时该 -1 对应一个不存在的维度，不支持广播。比如说目标 `shape` 为 $(3, -1, 4, 1, 5, 9)$ ，输入 `shape` 为 $(1, 5, 9)$ ，此时不进行增维处理，而是直接报错。

参数：

- **input** (Tensor) - 输入 tensor。
- **shape** (tuple) - 目标 shape。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> shape = (2, 3)
>>> x = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.broadcast_to(x, shape)
>>> print(output)
[[1. 2. 3.]
 [1. 2. 3.]]
>>> shape = (-1, 2)
>>> x = mindspore.tensor([[1], [2]], mindspore.float32)
>>> output = mindspore.ops.broadcast_to(x, shape)
>>> print(output)
[[1. 1.]
 [2. 2.]]
```

mindspore.ops.cat

`mindspore.ops.cat` (*tensors*, *axis=0*)

在指定轴上拼接输入 *tensor*。

参数：

- **tensors** (Union[tuple[Tensor], list[Tensor]]) - 输入 *tensors*。除了指定的拼接轴 *axis* 之外，其他轴的 *shape* 都应相等。
- **axis** (int) - 指定轴。默认 0。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input_x1 = mindspore.tensor([[0, 1], [2, 1]], mindspore.float32)
>>> input_x2 = mindspore.tensor([[0, 1], [2, 1]], mindspore.float32)
>>> output = mindspore.ops.cat((input_x1, input_x2))
>>> print(output)
[[0. 1.]
 [2. 1.]
 [0. 1.]
 [2. 1.]]
>>> output = mindspore.ops.cat((input_x1, input_x2), 1)
>>> print(output)
[[0. 1. 0. 1.]
 [2. 1. 2. 1.]]
```

mindspore.ops.channel_shuffle

`mindspore.ops.channel_shuffle` (*x*, *groups*)

将 *shape* 为 $(*, C, H, W)$ 的 *tensor* 的通道划分成 *g* 组，并按如下方式重新排列 $(*, \frac{C}{g}, g, H * W)$ ，同时在最终输出中保持原始 *tensor* 的 *shape*。

参数：

- **x** (Tensor) - 输入 *tensor*。
- **groups** (int) - 通道划分数目。

返回：

Tensor

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor(mindspore.ops.arange(0, 16, dtype=mindspore.int16).reshape(1, 4, 2, 2),
↪ dtype=mindspore.int16)
```

(下页继续)

(续上页)

```
>>> y = mindspore.ops.channel_shuffle(x, groups=2)
>>> print(y)
[[[ 0  1]
  [ 2  3]
  [ 8  9]
  [10 11]
  [ 4  5]
  [ 6  7]
  [12 13]
  [14 15]]]]
```

mindspore.ops.chunk

`mindspore.ops.chunk(input, chunks, axis=0)`

沿着指定轴将输入 tensor 切分成多个子 tensor。

说明：此函数返回的数量可能小于通过 *chunks* 指定的数量！

参数：

- **input** (Tensor) - 被切分的 tensor。
- **chunks** (int) - 切分的数量。
- **axis** (int, 可选) - 指定轴。默认 0。

返回：

由 tensor 组成的 tuple。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.ops.arange(0, 9, dtype=mindspore.float32)
>>> output = mindspore.ops.chunk(input, 3)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  1.00000000e+00,  2.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 3.00000000e+00,  4.00000000e+00,  5.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 6.00000000e+00,  7.00000000e+00,  8.
↪00000000e+00]))
```

mindspore.ops.column_stack

`mindspore.ops.column_stack(tensors)`

将多个 `tensor` 沿着水平方向维度堆叠成一个 `tensor`。类似于 `mindspore.ops.hstack()`。

参数:

- **tensors** (Union[tuple[Tensor], list[Tensor]]) - 输入 tensors。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x1 = mindspore.tensor([1, 1, 1])
>>> x2 = mindspore.tensor([2, 2, 2])
>>> output = mindspore.ops.column_stack((x1, x2))
>>> print(output)
[[1 2]
 [1 2]
 [1 2]]
```

mindspore.ops.concat

`mindspore.ops.concat(tensors, axis=0)`

`mindspore.ops.cat()` 的别名。

教程样例:

- 张量 Tensor - 张量运算
- Vision Transformer 图像分类 - 整体构建 ViT
- RNN 实现情感分类 - Dense

mindspore.ops.conj

`mindspore.ops.conj(input)`

逐元素计算输入 `tensor` 的共轭。复数的形式必须是 $a + bj$ ，其中 a 是实部， b 是虚部。

返回的共轭形式为 $a - bj$ 。

如果 `input` 是实数，则直接返回 `input`。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor(1.3+0.4j, mindspore.complex64)
>>> output = mindspore.ops.conj(x)
>>> output
Tensor(shape=[], dtype=Complex64, value= 1.3-0.4j)
```

mindspore.ops.count_nonzero

`mindspore.ops.count_nonzero(x, axis=(), keep_dims=False, dtype=mstype.int32)`

计算输入 `tensor` 指定轴上的非零元素的数量。如果没有指定轴，则计算 `tensor` 中所有非零元素的数量。

参数:

- **x** (Tensor) - 输入 `tensor`。
- **axis** (Union[int, tuple(int), list(int)], 可选) - 指定轴。默认 `()`，计算所有非零元素的个数。
- **keep_dims** (bool, 可选) - 是否保留 `axis` 指定的维度。默认 `False`，不保留对应维度。
- **dtype** (Union[Number, mindspore.bool_], 可选) - 指定数据类型。默认 `mstype.int32`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: each value specified.
>>> x = mindspore.tensor([[0, 1, 0], [1, 1, 0]], mindspore.float32)
>>> nonzero_num = mindspore.ops.count_nonzero(x=x, axis=[0, 1], keep_dims=True,
dtype=mindspore.int32)
>>> print(nonzero_num)
[[3]]
>>> # case 2: all value is default.
>>> nonzero_num = mindspore.ops.count_nonzero(x=x)
>>> print(nonzero_num)
3
>>> # case 3: axis value was specified 0.
>>> nonzero_num = mindspore.ops.count_nonzero(x=x, axis=[0,])
>>> print(nonzero_num)
[1 2 0]
>>> # case 4: axis value was specified 1.
>>> nonzero_num = mindspore.ops.count_nonzero(x=x, axis=[1,])
>>> print(nonzero_num)
[1 2]
>>> # case 5: keep_dims value was specified.
>>> nonzero_num = mindspore.ops.count_nonzero(x=x, keep_dims=True)
>>> print(nonzero_num)
[[3]]
>>> # case 6: keep_dims and axis value was specified.
>>> nonzero_num = mindspore.ops.count_nonzero(x=x, axis=[0,], keep_dims=True)
>>> print(nonzero_num)
[[1 2 0]]
```

mindspore.ops.deepcopy

`mindspore.ops.deepcopy(input_x)`

返回输入 `tensor` 的深拷贝。

参数:

- **input_x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[0, 1], [2, 1]], dtype=mindspore.int32)
>>> output = mindspore.ops.deepcopy(input)
>>> print(output)
[[0 1]
 [2 1]]
```

mindspore.ops.diag

`mindspore.ops.diag(input)`

返回一个用输入 `tensor` 作为对角线, 其余元素为 0 的 `tensor`。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: When input is a 1-D tensor:
>>> input = mindspore.ops.randn(3)
>>> output = mindspore.ops.diag(input)
>>> print(output)
[[ 1.7477764  0.          0.          ]
 [ 0.         -1.2616369  0.          ]
 [ 0.          0.         2.3283238]]
>>>
>>> # case 2: When input is a multi-dimensional tensor:
>>> input = mindspore.ops.randn(2, 3)
```

(下页继续)

(续上页)

```

>>> print(input)
[[ 0.21546374 -0.0120403 -0.7330481 ]
 [-2.5405762  0.44775972 -1.4063131 ]]
>>> output = mindspore.ops.diag(input)
>>> print(output)
[[[ 0.21546374  0.          0.          ]
 [ 0.          0.          0.          ]]
 [[ 0.          -0.0120403  0.          ]
 [ 0.          0.          0.          ]]
 [[ 0.          0.          -0.7330481 ]
 [ 0.          0.          0.          ]]]
[[[ 0.          0.          0.          ]
 [-2.5405762  0.          0.          ]]
 [[ 0.          0.          0.          ]
 [ 0.          0.44775972  0.          ]]
 [[ 0.          0.          0.          ]
 [ 0.          0.          -1.4063131 ]]]]
>>> # Assume input has dimensions (D_1,... D_k), the output is a tensor of rank 2k with
↳dimensions
   (D_1,..., D_k, D_1,..., D_k).
>>> print(output.shape)
(2, 3, 2, 3)

```

mindspore.ops.diagflat

`mindspore.ops.diagflat(input, offset=0)`

如果 `input` 是一维 `tensor`，则返回 `input` 作为对角线的二维 `tensor`，如果 `input` 是大于等于二维的 `tensor`，则将 `input` 展开后作为对角线的二维 `tensor`。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **offset** (int, 可选) - 对角线偏移量。默认 0。
 - 当 `offset` 是正整数时，对角线向上方偏移。
 - 当 `offset` 是负整数时，对角线向下方偏移。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> mindspore.ops.diagflat(mindspore.tensor([1, 2, 3]))
Tensor(shape=[3, 3], dtype=Int64, value=
[[1, 0, 0],
 [0, 2, 0],
 [0, 0, 3]])
>>> mindspore.ops.diagflat(mindspore.tensor([1, 2, 3]), 1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[0, 1, 0, 0],

```

(下页继续)

(续上页)

```
[0, 0, 2, 0],
[0, 0, 0, 3],
[0, 0, 0, 0]])
>>> mindspore.ops.diagflat(mindspore.tensor([[1, 2], [3, 4]]))
Tensor(shape=[4, 4], dtype=Int64, value=
[[1, 0, 0, 0],
 [0, 2, 0, 0],
 [0, 0, 3, 0],
 [0, 0, 0, 4]])
```

mindspore.ops.diagonal

`mindspore.ops.diagonal(input, offset=0, dim1=0, dim2=1)`

返回输入 `tensor` 在指定维度上的对角线视图。

若输入 `tensor` 是二维的，则返回一个一维 `tensor`，包含给定偏移位置处的对角线。

若输入 `tensor` 超过二维，则返回由 `dim1` 和 `dim2` 指定二维平面的对角线。返回 `tensor` 的 `shape` 为移除 `input` 的 `dim1` 和 `dim2` 维度，并且由 `dim1` 和 `dim2` 确定的对角线元素插入 `input` 的最后一维。

参数：

- **input** (Tensor) - 维度至少为二维的输入 `tensor`。
- **offset** (int, 可选) - 对角线偏移量。默认 0。
 - 当 `offset` 是正整数时，对角线向上方偏移。
 - 当 `offset` 是负整数时，对角线向下方偏移。
- **dim1** (int, 可选) - 返回指定平面对角线的第一维度。默认 0。
- **dim2** (int, 可选) - 返回指定平面对角线的第二维度。默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[[1, 0, 0],
...                             [0, 2, 0],
...                             [0, 0, 3]],
...                             [[4, 0, 0],
...                             [0, 5, 0],
...                             [0, 0, 6]],
...                             [[7, 0, 0],
...                             [0, 8, 0],
...                             [0, 0, 9]]])
>>> mindspore.ops.diagonal(input)
Tensor(shape=[3, 3], dtype=Int64, value=
[[1, 0, 0],
 [0, 5, 0],
 [0, 0, 9]])
>>> mindspore.ops.diagonal(input, offset=1)
```

(下页继续)

(续上页)

```
Tensor(shape=[3, 2], dtype=Int64, value=
[[0, 0],
 [2, 0],
 [0, 6]])
>>> mindspore.ops.diagonal(input, offset=0, dim1=2, dim2=1)
Tensor(shape=[3, 3], dtype=Int64, value=
[[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]])
```

mindspore.ops.diagonal_scatter

mindspore.ops.diagonal_scatter (*input, src, offset=0, dim1=0, dim2=1*)

将源 **tensor** 的值嵌入到输入 **tensor** 的对角线中（基于 *dim1* 和 *dim2* 指定的二维平面）。

说明：目前，*input* 和 *src* 中的元素不支持 `inf` 值。

参数：

- **input** (Tensor) - 维度大于 1 的输入 **tensor**。
- **src** (Tensor) - 要嵌入的源 **tensor**。
- **offset** (int, 可选) - 对角线偏移量。默认 0。
 - 当 *offset* 是正整数时，对角线向上方偏移。
 - 当 *offset* 是负整数时，对角线向下方偏移。
- **dim1** (int, 可选) - 确定指定平面对角线的第一维度。默认 0。
- **dim2** (int, 可选) - 确定指定平面对角线的第一维度。默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.ops.zeros((3, 3))
>>> output = mindspore.ops.diagonal_scatter(input, mindspore.ops.ones(3), 0)
>>> print(output)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
>>> output = mindspore.ops.diagonal_scatter(input, mindspore.ops.ones(2), 1)
>>> print(output)
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

mindspore.ops.diag_embed

`mindspore.ops.diag_embed(input, offset=0, dim1=-2, dim2=-1)`

创建一个 `tensor`，其特定二维平面（由 `dim1` 和 `dim2` 指定）的对角线由输入 `tensor` 填充，其余位置填充为 0。不指定维度时，默认填充返回 `tensor` 的最后两个维度所形成的二维平面的对角线。

参数：

- **input** (Tensor) - 对角线填充值。
- **offset** (int, 可选) - 对角线偏移量。默认 0。
 - 当 `offset` 是正整数时，对角线向上方偏移。
 - 当 `offset` 是负整数时，对角线向下方偏移。
- **dim1** (int, 可选) - 填充对角线的第一个维度。默认 -2。
- **dim2** (int, 可选) - 填充对角线的第二个维度。默认 -1。

返回：

一个数据类型与 `input` 一致，但输出 `shape` 维度比 `input` 高一维的 `tensor`。

异常：

- **ValueError** - `input` 的维度不是 1D-6D。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3],
...                           [4, 5, 6],
...                           [7, 8, 9]])
>>> mindspore.ops.diag_embed(input)
Tensor(shape=[3, 3, 3], dtype=Int64, value=
[[[1, 0, 0], [0, 2, 0], [0, 0, 3]],
 [[4, 0, 0], [0, 5, 0], [0, 0, 6]],
 [[7, 0, 0], [0, 8, 0], [0, 0, 9]]])
>>> mindspore.ops.diag_embed(input, offset=1, dim1=0, dim2=1)
Tensor(shape=[4, 4, 3], dtype=Int64, value=
[[[0, 0, 0], [1, 4, 7], [0, 0, 0], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [2, 5, 8], [0, 0, 0]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0], [3, 6, 9]],
 [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]])
```

mindspore.ops.dyn_shape

`mindspore.ops.dyn_shape(input_x)`

返回输入 `tensor` 的 `shape`。

参数：

- **input_x** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor(mindspore.ops.ones(shape=[3, 2, 1]), mindspore.float32)
>>> output = mindspore.ops.dyn_shape(input_x)
>>> print(output)
[3 2 1]
```

mindspore.ops.dsplitt

mindspore.ops.dsplitt (*input, indices_or_sections*)

沿第三轴分割输入 **tensor**。等同于 *axis = 2* 时的 *ops.tensor_split*。

参数:

- **input** (**Tensor**) - 待分割的 **tensor**。
- **indices_or_sections** (**Union[int, tuple(int), list(int)]**) - 参考 *mindspore.ops.tensor_split()* 中的 *indices_or_sections* 参数。

返回:

多个 **tensor** 组成的 **tuple**。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.ops.arange(16.0).reshape(2, 2, 4)
>>> print(input)
[[[ 0.  1.  2.  3.]
  [ 4.  5.  6.  7.]
  [ 8.  9. 10. 11.]
  [12. 13. 14. 15.]]]
>>> output = mindspore.ops.dsplitt(input, 2)
>>> print(output)
(Tensor(shape=[2, 2, 2], dtype=Float32, value=
[[[ 0.00000000e+00,  1.00000000e+00],
  [ 4.00000000e+00,  5.00000000e+00]],
 [[ 8.00000000e+00,  9.00000000e+00],
  [1.20000000e+01,  1.30000000e+01]]]), Tensor(shape=[2, 2, 2], dtype=Float32, value=
[[[ 2.00000000e+00,  3.00000000e+00],
  [ 6.00000000e+00,  7.00000000e+00]],
 [[ 1.00000000e+01,  1.10000000e+01],
  [ 1.40000000e+01,  1.50000000e+01]]]))
>>> output = mindspore.ops.dsplitt(input, [3, 6])
>>> print(output)
(Tensor(shape=[2, 2, 3], dtype=Float32, value=
[[[ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00],
  [ 4.00000000e+00,  5.00000000e+00,  6.00000000e+00]],
 [[ 8.00000000e+00,  9.00000000e+00,  1.00000000e+01],
  [ 1.20000000e+01,  1.30000000e+01,  1.40000000e+01]]]), Tensor(shape=[2, 2, 1], dtype=Float32, value=
```

(下页继续)

(续上页)

```
[[[ 3.00000000e+00],
   [ 7.00000000e+00]],
 [[ 1.10000000e+01],
   [ 1.50000000e+01]]], Tensor(shape=[2, 2, 0], dtype=Float32, value=
))
```

mindspore.ops.dstack

`mindspore.ops.dstack(tensors)`

将多个 `tensor` 沿着第三维度进行堆叠。

说明:

- 一维 `tensor` (N ,) 重新排列为 $(1, N, 1)$ ，二维 `tensor` (M, N) 重新排列为 $(M, N, 1)$ 。
- 除了第三个轴外，所有的 `tensor` 必须有相同的 `shape`。如果是一维或二维 `tensor`，则它们的 `shape` 必须相同。

参数:

- tensors** (`Union(List[Tensor], Tuple[Tensor])`) - 输入 `tensors`。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x1 = mindspore.tensor(mindspore.ops.arange(1, 7).reshape(2, 3))
>>> x2 = mindspore.tensor(mindspore.ops.arange(7, 13).reshape(2, 3))
>>> out = mindspore.ops.dstack([x1, x2])
>>> print(out.asnumpy())
[[[ 1.  7.]
   [ 2.  8.]
   [ 3.  9.]]
 [[ 4. 10.]
   [ 5. 11.]
   [ 6. 12.]]]
```

mindspore.ops.einsum

`mindspore.ops.einsum(equation, *operands)`

基于爱因斯坦求和约定 (Einsum) 符号，沿指定维度计算输入 `tensor` 元素的乘积之和。

说明:

- 现在支持子列表模式。例如，`ops.einsum(op1, sublist1, op2, sublist2, ..., sublist_out)`。在子列表模式中，`equation` 由子列表推导得到，Python 的省略号和介于 $[0, 52)$ 的整数 `list` 组成子列表。每个操作数后面都有一个子列表，并且最后有一个表示输出的子列表。

- *equation* 只能包含字母、逗号、省略号和箭头。字母表示输入 **tensor** 维数，逗号表示单独的 **tensor**，省略号表示忽略的 **tensor** 维数，箭头的左边表示输入 **tensor**，右边表示期望输出的维度。

参数：

- **equation** (str) - 基于爱因斯坦求和约定的符号。
- **operands** (Tensor) - 输入 **tensor**。

返回：

Tensor

支持平台：

GPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> equation = "i->"
>>> output = mindspore.ops.einsum(equation, x)
>>> print(output)
[7.]
>>> x = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> y = mindspore.tensor([2.0, 4.0, 3.0], mindspore.float32)
>>> equation = "i,i->i"
>>> output = mindspore.ops.einsum(equation, x, y)
>>> print(output)
[ 2. 8. 12.]
>>> x = mindspore.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]], mindspore.float32)
>>> y = mindspore.tensor([[2.0, 3.0], [1.0, 2.0], [4.0, 5.0]], mindspore.float32)
>>> equation = "ij,jk->ik"
>>> output = mindspore.ops.einsum(equation, x, y)
>>> print(output)
[[16. 22.]
 [37. 52.]]
>>> x = mindspore.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]], mindspore.float32)
>>> equation = "ij->ji"
>>> output = mindspore.ops.einsum(equation, x)
>>> print(output)
[[1. 4.]
 [2. 5.]
 [3. 6.]]
>>> x = mindspore.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]], mindspore.float32)
>>> equation = "ij->j"
>>> output = mindspore.ops.einsum(equation, x)
>>> print(output)
[5. 7. 9.]
>>> x = mindspore.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]], mindspore.float32)
>>> equation = "...->"
>>> output = mindspore.ops.einsum(equation, x)
>>> print(output)
[21.]
>>> x = mindspore.tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> y = mindspore.tensor([2.0, 4.0, 1.0], mindspore.float32)
>>> equation = "j,i->ji"
>>> output = mindspore.ops.einsum(equation, x, y)
```

(下页继续)

(续上页)

```
>>> print(output)
[[ 2.  4.  1.]
 [ 4.  8.  2.]
 [ 6. 12.  3.]]
>>> x = mindspore.tensor([1, 2, 3, 4], mindspore.float32)
>>> y = mindspore.tensor([1, 2], mindspore.float32)
>>> output = mindspore.ops.einsum(x, [..., 1], y, [..., 2], [..., 1, 2])
>>> print(output)
[[1. 2.]
 [2. 4.]
 [3. 6.]
 [4. 8.]]
```

mindspore.ops.expand_dims

`mindspore.ops.expand_dims(input_x, axis)`

为输入 `tensor` 新增额外的轴。

说明:

- `input_x` 的维度应该大于等于 1。
- 如果指定的 `axis` 是负数，那么它会从后往前，从 1 开始计算 `index`。

参数:

- **input_x** (Tensor) - 输入 `tensor`。
- **axis** (int) - 新增的轴。仅接受常量。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_tensor = mindspore.tensor([[2, 2], [2, 2]], mindspore.float32)
>>> output = mindspore.ops.expand_dims(input_tensor, 0)
>>> print(output)
[[[2. 2.]
 [2. 2.]]]
```

mindspore.ops.flip

`mindspore.ops.flip(input, dims)`

沿指定维度翻转 tensor 中元素。

参数：

- **input** (Tensor) - 输入 tensor。
- **dims** (Union[list[int], tuple[int]]) - 指定维度。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor(mindspore.ops.arange(1, 9).reshape((2, 2, 2)))
>>> output = mindspore.ops.flip(input, (0, 2))
>>> print(output)
[[[6 5]
  [8 7]]
  [[2 1]
  [4 3]]]
```

mindspore.ops.fliplr

`mindspore.ops.fliplr(input)`

向左/向右翻转输入 tensor。

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor(mindspore.ops.arange(1, 9).reshape((2, 2, 2)))
>>> output = mindspore.ops.fliplr(input)
>>> print(output)
[[[3 4]
  [1 2]]
  [[7 8]
  [5 6]]]
```

mindspore.ops.flipud

`mindspore.ops.flipud(input)`

向上/向下翻转输入 tensor。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

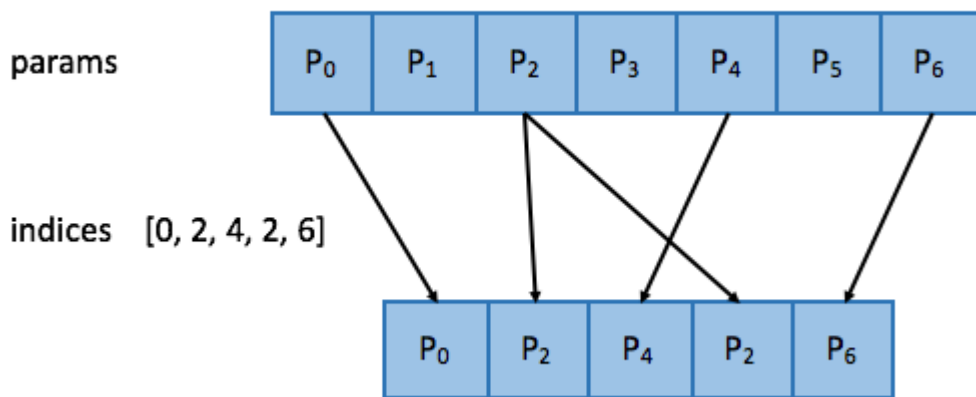
```
>>> import mindspore
>>> input = mindspore.tensor(mindspore.ops.arange(1, 9).reshape((2, 2, 2)))
>>> output = mindspore.ops.flipud(input)
>>> print(output)
[[[5 6]
  [7 8]]
 [[1 2]
  [3 4]]]
```

mindspore.ops.gather

`mindspore.ops.gather(input_params, input_indices, axis, batch_dims=0)`

返回输入 tensor 在指定轴及指定索引上对应元素的切片。

下图展示了 Gather 常用的计算过程:



其中, `params` 代表输入 `input_params`, `indices` 代表要切片的索引 `input_indices`。

说明:

- `input_indices` 的值必须在 `[0, input_params.shape[axis]]` 范围内。CPU 与 GPU 平台越界访问将会抛出异常, Ascend 平台越界访问的返回结果是未定义的。

- Ascend 平台上, `input_params` 的数据类型不能是 `mindspore.bool_`。
- 返回 tensor 的 shape 为 `input_params.shape[: axis] + input_indices.shape[batch_dims :] + input_params.shape[axis + 1 :]`。

参数:

- **input_params** (Tensor) - 输入 tensor。
- **input_indices** (Tensor) - 指定索引。
- **axis** (Union(int, Tensor[int])) - 指定轴。
- **batch_dims** (int) - batch 维的数量。默认 0。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case1: input_indices is a Tensor with shape (5, ).
>>> input_params = mindspore.tensor([1, 2, 3, 4, 5, 6, 7], mindspore.float32)
>>> input_indices = mindspore.tensor([0, 2, 4, 2, 6], mindspore.int32)
>>> axis = 0
>>> output = mindspore.ops.gather(input_params, input_indices, axis)
>>> print(output)
[1. 3. 5. 3. 7.]
>>> # case2: input_indices is a Tensor with shape (2, 2). When the input_params has one_
↳dimension,
>>> # the output shape is equal to the input_indices shape.
>>> input_indices = mindspore.tensor([[0, 2], [2, 6]], mindspore.int32)
>>> axis = 0
>>> output = mindspore.ops.gather(input_params, input_indices, axis)
>>> print(output)
[[1. 3.]
 [3. 7.]]
>>> # case3: input_indices is a Tensor with shape (2, ) and
>>> # input_params is a Tensor with shape (3, 4) and axis is 0.
>>> input_params = mindspore.tensor([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]], mindspore.
↳float32)
>>> input_indices = mindspore.tensor([0, 2], mindspore.int32)
>>> axis = 0
>>> output = mindspore.ops.gather(input_params, input_indices, axis)
>>> print(output)
[[ 1.  2.  3.  4.]
 [ 9. 10. 11. 12.]]
>>> # case4: input_indices is a Tensor with shape (2, ) and
>>> # input_params is a Tensor with shape (3, 4) and axis is 1, batch_dims is 1.
>>> input_params = mindspore.tensor([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]], mindspore.
↳float32)
>>> input_indices = mindspore.tensor([0, 2, 1], mindspore.int32)
>>> axis = 1
>>> batch_dims = 1
>>> output = mindspore.ops.gather(input_params, input_indices, axis, batch_dims)
```

(下页继续)

(续上页)

```
>>> print(output)
[ 1.  7. 10.]
```

mindspore.ops.gather_d

`mindspore.ops.gather_d(x, dim, index)`

获取指定轴的元素。有关更多详细信息，请参阅[mindspore.ops.gather_elements\(\)](#)。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2], [3, 4]], mindspore.int32)
>>> index = mindspore.tensor([[0, 0], [1, 0]], mindspore.int32)
>>> dim = 1
>>> output = mindspore.ops.gather_d(x, dim, index)
>>> print(output)
[[1 1]
 [4 3]]
```

mindspore.ops.gather_elements

`mindspore.ops.gather_elements(input, dim, index)`

根据指定维度和索引获取元素。

说明： *input* 与 *index* 维度大小一致，当 *input* 中的 *axis* $\neq$ *dim* 时， $index.shape[axis] \leq input.shape[axis]$ 。

警告：在 Ascend 后端，以下场景将导致不可预测的行为：

- 正向执行流程中，当 *index* 的取值不在范围 $[-input.shape[dim], input.shape[dim])$ 内；
- 反向执行流程中，当 *index* 的取值不在范围 $[0, input.shape[dim])$ 内。

参数：

- **input** (Tensor) - 输入 tensor。
- **dim** (int) - 指定维度。
- **index** (Tensor) - 指定索引。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2], [3, 4]], mindspore.int32)
>>> index = mindspore.tensor([[0, 0], [1, 0]], mindspore.int32)
>>> dim = 1
>>> output = mindspore.ops.gather_elements(x, dim, index)
>>> print(output)
[[1 1]
 [4 3]]
```

mindspore.ops.gather_nd

`mindspore.ops.gather_nd(input_x, indices)`

根据指定索引获取输入 `tensor` 的切片。

假设 `indices` 是一个 K 维的整型张量，遵循公式如下：

$$output[(i_0, ..., i_{K-2})] = input\_x[indices[(i_0, ..., i_{K-2})]]$$

需满足 `indices.shape[-1] <= len(input_x.shape)`。

参数：

- **input_x** (Tensor) - 输入 `tensor`。
- **indices** (Tensor) - 指定索引。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> input_x = mindspore.tensor([[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]], mindspore.float32)
>>> indices = mindspore.tensor([[0, 0], [1, 1]], mindspore.int32)
>>> output = mindspore.ops.gather_nd(input_x, indices)
>>> print(output)
[-0.1  0.5]
```

mindspore.ops.hsplit

`mindspore.ops.hsplit(input, indices_or_sections)`

将输入 `tensor` 水平分割成多个子 `tensor`。等同于 `axis = 1` 时的 `ops.tensor_split`。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 参考 `mindspore.ops.tensor_split()`。

返回：

由多个 `tensor` 组成的 `tuple`。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.ops.arange(0, 6).reshape((2, 3))
>>> output = mindspore.ops.hsplit(mindspore.tensor(input_x, mindspore.float32), 3)
>>> print(output)
(Tensor(shape=[2, 1], dtype=Float32, value=[[ 0.00000000e+00], [ 3.00000000e+00]]),
 Tensor(shape=[2, 1], dtype=Float32, value=[[ 1.00000000e+00], [ 4.00000000e+00]]),
 Tensor(shape=[2, 1], dtype=Float32, value=[[ 2.00000000e+00], [ 5.00000000e+00]]))
```

mindspore.ops.hstack

`mindspore.ops.hstack` (*tensors*)

将多个 `tensor` 沿着水平方向堆叠。

说明:

- 在 `graph mode` 下, 不支持 `mindspore.float64` 类型的 8-D `tensor` 的动态 `rank` 输入。
 - 对于一维 `tensor`, 沿第一个轴堆叠。其他维度的 `tensor` 沿第二个轴堆叠。
 - 对于维度大于一的 `tensor`, 除了第二个轴外, 所有 `tensor` 的 `shape` 必须相同。对于一维 `tensor`, 可拥有任意的长度。
-

参数:

- **tensors** (`Union[tuple[Tensor], list[Tensor]]`) - 输入 `tensors`。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x1 = mindspore.tensor([1, 1, 1])
>>> x2 = mindspore.tensor([2, 2, 2])
>>> output = mindspore.ops.hstack((x1, x2))
>>> print(output)
[1. 1. 1. 2. 2. 2.]
```


mindspore.ops.index_add

`mindspore.ops.index_add(x, indices, y, axis, use_lock=True, check_index_bound=True)`

按照指定轴和索引将输入 *y* 的元素加到输入 *x* 中。

说明:

- *indices* 为一维 tensor, 并且 *indices.shape[0] = y.shape[axis]*。
- *indices* 中元素的取值范围为 $[0, x.shape[axis] - 1]$ 。

参数:

- **x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **y** (Tensor) - 与 *x* 相加的 tensor。
- **axis** (int) - 指定轴。
- **use_lock** (bool, 可选) - 是否对参数更新过程加锁保护。默认 True。
- **check_index_bound** (bool, 可选) - 是否检查 *indices* 边界。默认 True。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.Parameter(mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.
↪float32),
...                               name="name_x")
>>> indices = mindspore.tensor([0, 2], mindspore.int32)
>>> y = mindspore.tensor([[0.5, 1.0], [1.0, 1.5], [2.0, 2.5]], mindspore.float32)
>>> output = mindspore.ops.index_add(x, indices, y, 1)
>>> print(output)
[[ 1.5  2.   4. ]
 [ 5.   5.   7.5]
 [ 9.   8.  11.5]]
```

mindspore.ops.index_fill

`mindspore.ops.index_fill(x, axis, index, value)`

按照指定轴和索引用输入 *value* 填充输入 *x* 的元素。

参数:

- **x** (Tensor) - 输入 tensor。
- **axis** (Union[int, Tensor]) - 指定轴。
- **index** (Tensor) - 指定索引。
- **value** (Union[bool, int, float, Tensor]) - 填充输入 tensor 的值。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.float32)
>>> index = mindspore.tensor([0, 2], mindspore.int32)
>>> value = mindspore.tensor(-2.0, mindspore.float32)
>>> y = mindspore.ops.index_fill(x, 1, index, value)
>>> print(y)
[[-2.  2. -2.]
 [-2.  5. -2.]
 [-2.  8. -2.]]
```

mindspore.ops.index_select

mindspore.ops.index_select (*input, axis, index*)

根据指定轴和索引对输入 **tensor** 进行选取, 返回一个新 **tensor**。

说明:

- *index* 的值必须在 $[0, \text{input.shape}[\text{axis}])$ 范围内, 超出该范围的结果未定义。
- 返回的 **tensor** 和输入 **tensor** 的维度数量相同, 其第 *axis* 维度的大小和 *index* 的长度相同, 其他维度和 *input* 相同。

参数:

- **input** (Tensor) - 输入 **tensor**。
- **axis** (int) - 指定轴。
- **index** (Tensor) - 指定索引, 一维 **tensor**。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor(mindspore.ops.arange(0, 16).reshape(2, 2, 4), mindspore.float32)
>>> print(input)
[[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]]]
>>> index = mindspore.tensor([0,], mindspore.int32)
>>> y = mindspore.ops.index_select(input, 1, index)
>>> print(y)
```

(下页继续)

(续上页)

```
[[[ 0.  1.  2.  3.]]
 [[ 8.  9. 10. 11.]]]
```

mindspore.ops.inplace_add

`mindspore.ops.inplace_add(x, v, indices)`

根据索引将 x 加 v 。

对于 $indices$ 的每个元素下标 $i, \dots, j$:

$$x[index[i, \dots, j]] += y[i, \dots, j]$$

参数:

- **x** (Tensor) - 输入 tensor。
- **v** (Tensor) - 与 x 相加的 tensor。
- **indices** (Union[int, tuple]) - 对输入 x 沿第零维度的索引。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> indices = (0, 1)
>>> x = mindspore.tensor([[1, 2], [3, 4], [5, 6]], mindspore.float32)
>>> input_v = mindspore.tensor([[0.5, 1.0], [1.0, 1.5]], mindspore.float32)
>>> output = mindspore.ops.inplace_add(x, input_v, indices)
>>> print(output)
[[1.5 3. ]
 [4.  5.5]
 [5.  6. ]]
```

mindspore.ops.inplace_index_add

`mindspore.ops.inplace_index_add(var, indices, updates, axis)`

根据指定轴和索引在输入 var 中的对应位置加 $updates$ 。

对于 $indices$ 的每个元素下标 $i, \dots, j$:

$$x[:, indices[i, \dots, j], :] += v[:, i, \dots, j, :]$$

其中 i 是 $indices$ 中元素的下标, $indices[i]$ 所在的轴由输入 $axis$ 决定。

参数:

- **var** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引, 一维 tensor。
- **updates** (Tensor) - 与 var 相加的 tensor。

- **axis** (int) - 指定轴。

返回：

Tensor

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> var = mindspore.Parameter(mindspore.tensor([[1, 2], [3, 4], [5, 6]], mindspore.float32))
>>> indices = mindspore.tensor([0, 1], mindspore.int32)
>>> updates = mindspore.tensor([[0.5, 1.0], [1.0, 1.5]], mindspore.float32)
>>> mindspore.ops.inplace_index_add(var, indices, updates, axis=0)
>>> print(var.asnumpy())
[[1.5 3. ]
 [4.  5.5]
 [5.  6. ]]
```

mindspore.ops.inplace_sub

`mindspore.ops.inplace_sub(x, v, indices)`

根据索引将 x 减 v 。

对于 *indices* 的每个元素下标 $i, \dots, j$ ：

$$x[\text{index}[i, \dots, j]] -= v[i, \dots, j]$$

参数：

- **x** (Tensor) - 输入 tensor。
- **v** (Tensor) - 被 x 减去的 tensor。
- **indices** (Union[int, tuple]) - 对输入 x 沿第 0 维度的索引。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> indices = (0, 1)
>>> x = mindspore.tensor([[1, 2], [3, 4], [5, 6]], mindspore.float32)
>>> input_v = mindspore.tensor([[0.5, 1.0], [1.0, 1.5]], mindspore.float32)
>>> output = mindspore.ops.inplace_sub(x, input_v, indices)
>>> print(output)
[[0.5 1. ]
 [2.  2.5]
 [5.  6. ]]
```

mindspore.ops.inplace_update

`mindspore.ops.inplace_update(x, v, indices)`

根据索引将 x 更新为 v 。

警告：这是一个实验性 API，后续可能修改或删除。

对于 $indices$ 的每个元素下标 $i, \dots, j$ ：

$$x[\text{index}[i, \dots, j]] = v[i, \dots, j]$$

参数：

- **x** (Tensor) - 输入 tensor。
- **v** (Tensor) - 更新的 tensor。
- **indices** (Union[int, tuple[int], Tensor]) - 对输入 x 沿第 0 维度的索引。

返回：

Tensor

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> indices = (0, 1)
>>> x = mindspore.tensor([[1, 2], [3, 4], [5, 6]], mindspore.float32)
>>> v = mindspore.tensor([[0.5, 1.0], [1.0, 1.5]], mindspore.float32)
>>> output = mindspore.ops.inplace_update(x, v, indices)
>>> print(output)
[[0.5 1. ]
 [1.  1.5]
 [5.  6. ]]
```

mindspore.ops.is_nonzero

`mindspore.ops.is_nonzero(input)`

判断输入 tensor 是否包含 0 或 False。输入只能是单元素。

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Bool

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x1 = mindspore.tensor([[[False]]])
>>> x2 = mindspore.tensor([[[3.5]]])
>>> out1 = mindspore.ops.is_nonzero(x1)
>>> print(out1)
False
>>> out2 = mindspore.ops.is_nonzero(x2)
>>> print(out2)
True
```

mindspore.ops.masked_fill

mindspore.ops.masked_fill (*input_x, mask, value*)

在掩码为 True 位置填充指定值。

支持广播。

参数：

- **input_x** (Tensor) - 输入 tensor。
- **mask** (Tensor[bool]) - 输入掩码。
- **value** (Union[Number, Tensor]) - 指定值。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input_x = mindspore.tensor([1., 2., 3., 4.], mindspore.float32)
>>> mask = mindspore.tensor([True, True, False, True], mindspore.bool_)
>>> output = mindspore.ops.masked_fill(input_x, mask, 0.5)
>>> print(output)
[0.5 0.5 3. 0.5]
```

mindspore.ops.masked_select

mindspore.ops.masked_select (*input, mask*)

返回一个一维 tensor，它根据掩码对输入 tensor 进行索引。

支持广播。

参数：

- **input** (Tensor) - 输入 tensor。
- **mask** (Tensor[bool]) - 输入掩码。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1, 2, 3, 4], mindspore.int64)
>>> mask = mindspore.tensor([1, 0, 1, 0], mindspore.bool_)
>>> output = mindspore.ops.masked_select(x, mask)
>>> print(output)
[1 3]
```

mindspore.ops.meshgrid

`mindspore.ops.meshgrid(*inputs, indexing='xy')`

从一维输入生成多维坐标网格。

说明:

- graph mode 下的 *inputs* 为 N 个一维 tensor, N 应大于 1。
- pynative mode 下的 *inputs* 为 N 个零维或一维 tensor, N 应大于 0。数据类型为 Number。
- 对于长度为 *M* 和 *N* 的二维输入, 取值为 'xy' 时, 输出的 shape 为 (*N*, *M*), 取值为 'ij' 时, 输出的 shape 为 (*M*, *N*)。
- 对于长度为 *M*, *N* 和 *P* 的三维输入, 取值为 'xy' 时, 输出的 shape 为 (*N*, *M*, *P*), 取值为 'ij' 时, 输出的 shape 为 (*M*, *N*, *P*)。

参数:

- **inputs** (Union[tuple[Tensor], list[Tensor]]) - 输入 tensors。

关键字参数:

- **indexing** (str, 可选) - 输出的网格矩阵的 size。可选 'xy' 或 'ij'。默认 'xy'。

返回:

由 N 个 N 维 tensor 组成的元组。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1, 2, 3, 4], mindspore.int32)
>>> y = mindspore.tensor([5, 6, 7], mindspore.int32)
>>> z = mindspore.tensor([8, 9, 0, 1, 2], mindspore.int32)
>>> output = mindspore.ops.meshgrid(x, y, z, indexing='xy')
>>> print(output)
(Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2],
  [3, 3, 3, 3, 3],
  [4, 4, 4, 4, 4]],
 [[1, 1, 1, 1, 1],
```

(下页继续)

(续上页)

```

    [2, 2, 2, 2, 2],
    [3, 3, 3, 3, 3],
    [4, 4, 4, 4, 4]],
  [[1, 1, 1, 1, 1],
   [2, 2, 2, 2, 2],
   [3, 3, 3, 3, 3],
   [4, 4, 4, 4, 4]]]),
Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5]],
 [[6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6]],
 [[7, 7, 7, 7, 7],
  [7, 7, 7, 7, 7],
  [7, 7, 7, 7, 7],
  [7, 7, 7, 7, 7]]]),
Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]],
 [[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]],
 [[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]]])

```

mindspore.ops.moveaxis

`mindspore.ops.moveaxis(x, source, destination)`

将输入 `tensor` 的两个轴调换位置。

更多参考详见 `mindspore.ops.movedim()`。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> x = mindspore.ops.zeros((3, 4, 5))
>>> output = mindspore.ops.moveaxis(x, 0, -1)
>>> print(output.shape)
(4, 5, 3)

```


mindspore.ops.movedim

`mindspore.ops.movedim(x, source, destination)`

将输入 `tensor` 的两个维度调换位置。

参数：

- **x** (Tensor) - 输入 `tensor`。
- **source** (Union[int, sequence[int]]) - 源维度。
- **destination** (Union[int, sequence[int]]) - 源维度的目标位置。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> # case1 : moving single axis
>>> import mindspore
>>> x = mindspore.tensor(mindspore.ops.zeros((3, 4, 5)))
>>> output = mindspore.ops.movedim(x, 0, -1)
>>> print(output.shape)
(4, 5, 3)
>>> # case 2 : moving multiple axes
>>> x = mindspore.tensor(mindspore.ops.zeros((3, 4, 5)))
>>> output = mindspore.ops.movedim(x, (0, 2), (1, 2))
>>> print(output.shape)
(4, 3, 5)
```

mindspore.ops.narrow

`mindspore.ops.narrow(input, axis, start, length)`

沿指定轴 `axis`，从 `start` 位置截取长度为 `length` 的 `tensor`。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **axis** (int) - 指定轴。
- **start** (int) - 指定起始位置。
- **length** (int) - 指定长度。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.int32)
>>> output = mindspore.ops.narrow(x, 0, 0, 2)
>>> print(output)
[[ 1 2 3]
 [ 4 5 6]]
>>> output = mindspore.ops.narrow(x, 1, 1, 2)
>>> print(output)
[[ 2 3]
 [ 5 6]
 [ 8 9]]
```

mindspore.ops.nan_to_num

`mindspore.ops.nan_to_num(input, nan=None, posinf=None, neginf=None)`

将 `input` 中的 `NaN`、正无穷大和负无穷大值分别替换为 `nan`、`posinf` 和 `neginf` 指定的值。

警告： 对于 Ascend，仅支持 Atlas A2 训练系列产品。这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 tensor。
- **nan** (number, 可选) - `NaN` 的替换值，默认 `None`。
- **posinf** (number, 可选) - `posinf` 的替换值。如果为 `None`，则为 `input` 类型支持的上限，默认 `None`。
- **neginf** (number, 可选) - `neginf` 的替换值。如果为 `None`，则为 `input` 类型支持的下限，默认 `None`。

返回：

Tensor

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([float('nan'), float('inf'), -float('inf'), 5.0], mindspore.
    ↪ float32)
>>> output = mindspore.ops.nan_to_num(input)
>>> print(output)
[ 0.00000000e+00  3.4028235e+38 -3.4028235e+38  5.0000000e+00]
>>> output = mindspore.ops.nan_to_num(input, 1.0)
>>> print(output)
[ 1.0000000e+00  3.4028235e+38 -3.4028235e+38  5.0000000e+00]
>>> output = mindspore.ops.nan_to_num(input, 1.0, 2.0)
>>> print(output)
[ 1.0000000e+00  2.0000000e+00 -3.4028235e+38  5.0000000e+00]
>>> output = mindspore.ops.nan_to_num(input, 1.0, 2.0, 3.0)
>>> print(output)
[1.  2.  3.  5.]
```

mindspore.ops.nanmean

`mindspore.ops.nanmean(input, axis=None, keepdims=False, *, dtype=None)`

忽略 NaN 值，按指定轴计算输入 tensor 的平均值。

参数：

- **input** (Tensor) - 输入 tensor。
- **axis** (int, 可选) - 指定轴，默认 None。
- **keepdims** (bool, 可选) - 输出 tensor 是否保留维度，默认 False。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型，默认 None。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[0.5, -1.1, float('nan')], [3.4, float('nan'), float('nan')]],
↳mindspore.float32)
>>> y = mindspore.ops.nanmean(x, axis=0, keepdims=False)
>>> print(y)
[ 1.95 -1.1      nan]
```

mindspore.ops.nanmedian

`mindspore.ops.nanmedian(input, axis=-1, keepdims=False)`

忽略 NaN 值，按指定轴计算输入 tensor 的中值和索引。

警告： 如果 *input* 的中值不唯一，则 *indices* 不一定包含第一个出现的中值。

参数：

- **input** (Tensor) - 输入 tensor。
- **axis** (int, 可选) - 指定轴，默认 -1。
- **keepdims** (bool, 可选) - 输出 tensor 是否保持维度，默认 False。

返回：

一个由 tensor 组成的 tuple(median, median_indices)

支持平台：

CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[0.57, 0.11, float("nan")],
...                       [0.38, float("nan"), float("nan")],
...                       [0.36, 0.16, float("nan")]], mindspore.float32)
>>> y, idx = mindspore.ops.nanmedian(x, axis=0, keepdims=False)
>>> print(y)
[0.38 0.11 nan]
>>> print(idx)
[1 0 0]
```

mindspore.ops.nansum

`mindspore.ops.nansum(input, axis=None, keepdims=False, *, dtype=None)`

忽略 NaN 值，按指定轴计算输入 tensor 的和。

参数：

- **input** (Tensor) - 输入 Tensor。
- **axis** (Union[int, tuple(int)], 可选) - 指定轴。假设 *input* 的秩为 *r*，取值范围 $[-r, r]$ ，默认 *None*，对 tensor 中的所有元素求和。
- **keepdims** (bool, 可选) - 输出 tensor 是否保持维度，默认 *False*，不保留维度。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 指定数据类型，默认 *None*。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[float("nan"), 2, 3], [1, float("nan"), 3], [1, 2, float("nan")]],
↪mindspore.float32)
>>> # case1: axis is None, keepdims is False,
>>> output1 = mindspore.ops.nansum(x, axis=None, dtype=mindspore.float32)
>>> print(output1)
12.0
>>> # case2: axis is int, set as 0, and keepdims is False
>>> output2 = mindspore.ops.nansum(x, axis=0, dtype=mindspore.float32)
>>> print(output2)
[2. 4. 6.]
>>> # case3: axis is int, set as 0, and keepdims is False
>>> output3 = mindspore.ops.nansum(x, axis=0, keepdims=True, dtype=mindspore.float32)
>>> print(output3)
[[2. 4. 6.]]
>>> # case4: axis is tuple(int) or list(int), set as (0, 1), and keepdims is False
>>> output4 = mindspore.ops.nansum(x, axis=(0, 1), dtype=mindspore.float32)
>>> print(output4)
12.0
```

mindspore.ops.normal

`mindspore.ops.normal(shape, mean, stddev, seed=None)`

返回符合正态（高斯）分布的随机 `tensor`。

警告：Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **shape** (tuple) - 指定 shape。
- **mean** (Union[`Tensor`, int, float]) - 返回 `tensor` 的正态分布平均值。
- **stddev** (Union[`Tensor`, int, float]) - 返回 `tensor` 的正态分布标准差。
- **seed** (int, 可选) - 随机种子，默认 `None`，等同于 0。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> shape = (3, 1, 2)
>>> mean = mindspore.tensor([[3, 4], [5, 6]], mindspore.float32)
>>> stddev = mindspore.tensor(1.0, mindspore.float32)
>>> output = mindspore.ops.normal(shape, mean, stddev, seed=5)
>>> result = output.shape
>>> print(result)
(3, 2, 2)
>>> shape = (3, 1, 3)
>>> mean = mindspore.tensor([[3, 4, 3], [3, 5, 6]], mindspore.float32)
>>> stddev = mindspore.tensor(1.0, mindspore.float32)
>>> output = mindspore.ops.normal(shape, mean, stddev, seed=5)
>>> result = output.shape
>>> print(result)
(3, 2, 3)
>>> shape = (3, 1, 3)
>>> mean = mindspore.tensor([[1, 2, 3], [3, 4, 3], [3, 5, 6]], mindspore.float32)
>>> stddev = mindspore.tensor(1.0, mindspore.float32)
>>> output = mindspore.ops.normal(shape, mean, stddev, seed=5)
>>> result = output.shape
>>> print(result)
(3, 3, 3)
```

mindspore.ops.nonzero

`mindspore.ops.nonzero(input, *, as_tuple=False)`

返回所有非零元素下标位置。

参数:

- **input** (Tensor) - 输入 tensor。

说明:

- Ascend: 输入 tensor 的秩可以等于 0, jit level O2 模式除外。
- CPU/GPU: 输入 tensor 秩应大于等于 1。

关键字参数:

- **as_tuple** (bool, 可选) - 是否以 tuple 形式输出, 默认 False。

返回:

Tensor 或者由 tensor 组成的 tuple。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[[1, 0], [-5, 0]]], mindspore.int32)
>>> output = mindspore.ops.nonzero(x)
>>> print(output)
[[0 0 0]
 [0 1 0]]
>>> x = mindspore.tensor([1, 0, 2, 0, 3], mindspore.int32)
>>> output = mindspore.ops.nonzero(x, as_tuple=False)
>>> print(output)
[[0]
 [2]
 [4]]
>>> x = mindspore.tensor([[[1, 0], [-5, 0]]], mindspore.int32)
>>> output = mindspore.ops.nonzero(x, as_tuple=True)
>>> print(output)
(Tensor(shape=[2], dtype=Int64, value=[0, 0]),
 Tensor(shape=[2], dtype=Int64, value=[0, 1]),
 Tensor(shape=[2], dtype=Int64, value=[0, 0]))
>>> x = mindspore.tensor([1, 0, 2, 0, 3], mindspore.int32)
>>> output = mindspore.ops.nonzero(x, as_tuple=True)
>>> print(output)
(Tensor(shape=[3], dtype=Int64, value=[0, 2, 4]), )
```

mindspore.ops.numel

`mindspore.ops.numel(input)`

返回 `tensor` 的元素的数量。

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor 元素总数量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([[2, 2], [2, 2]], mindspore.float32)
>>> print(mindspore.ops.numel(input_x))
4
```

mindspore.ops.permute

`mindspore.ops.permute(input, axis)`

按照指定轴重排输入 `tensor`。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **axis** (tuple(int)) - 指定顺序的轴。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]],
↪mindspore.float32)
>>> input_perm = (0, 2, 1)
>>> print(mindspore.ops.permute(input_x, input_perm))
[[ [ 1.  4.]
   [ 2.  5.]
   [ 3.  6.]]
 [ [ 7. 10.]
   [ 8. 11.]
   [ 9. 12.] ]]
```

mindspore.ops.population_count

`mindspore.ops.population_count(input_x)`

计算输入 `tensor` 中每个元素二进制的 1 比特数量。

参数:

- **input_x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([0, 1, 3], mindspore.int16)
>>> output = mindspore.ops.population_count(input_x)
>>> print(output)
[0 1 2]
```

mindspore.ops.rank

`mindspore.ops.rank(input_x)`

返回输入 `tensor` 的秩。

参数:

- **input_x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_tensor = mindspore.tensor([[2, 2], [2, 2]], mindspore.float32)
>>> output = mindspore.ops.rank(input_tensor)
>>> print(output)
2
>>> print(type(output))
<class 'int'>
```


mindspore.ops.repeat_elements

`mindspore.ops.repeat_elements(x, rep, axis=0)`

在指定轴上复制输入 `tensor` 的元素, 类似 `mindspore.numpy.repeat()` 的功能。

说明: 推荐使用 `mindspore.mint.repeat_interleave()`, 输入 `x` 的维度最大可支持 8, 并获得更好的性能。

参数:

- **x** (Tensor) - 输入 `tensor`。
- **rep** (int) - 复制次数, 为正数。
- **axis** (int) - 指定轴, 默认 0。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1 : repeat on axis 0
>>> x = mindspore.tensor([[0, 1, 2], [3, 4, 5]], mindspore.int32)
>>> output = mindspore.ops.repeat_elements(x, rep = 2, axis = 0)
>>> print(output)
[[0 1 2]
 [0 1 2]
 [3 4 5]
 [3 4 5]]
>>> # case 2 : repeat on axis 1
>>> x = mindspore.tensor([[0, 1, 2], [3, 4, 5]], mindspore.int32)
>>> output = mindspore.ops.repeat_elements(x, rep = 2, axis = 1)
>>> print(output)
[[0 0 1 1 2 2]
 [3 3 4 4 5 5]]
```

mindspore.ops.repeat_interleave

`mindspore.ops.repeat_interleave(input, repeats, axis=None)`

在指定轴上复制输入 `tensor` 的元素, 类似 `mindspore.numpy.repeat()`。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **repeats** (Union[int, tuple, list, Tensor]) - 复制次数, 为正数。
- **axis** (int, 可选) - 指定轴, 默认 `None`。如果为 `None`, 输入和输出 `tensor` 会被展平为一维。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1 : repeat on axis 0
>>> input = mindspore.tensor([[0, 1, 2], [3, 4, 5]], mindspore.int32)
>>> output = mindspore.ops.repeat_interleave(input, repeats=2, axis=0)
>>> print(output)
[[0 1 2]
 [0 1 2]
 [3 4 5]
 [3 4 5]]
>>> # case 2 : repeat on axis 1
>>> input = mindspore.tensor([[0, 1, 2], [3, 4, 5]], mindspore.int32)
>>> output = mindspore.ops.repeat_interleave(input, repeats=2, axis=1)
>>> print(output)
[[0 0 1 1 2 2]
 [3 3 4 4 5 5]]
```

mindspore.ops.reshape

`mindspore.ops.reshape` (*input*, *shape*)

按指定 *shape* , 对输入 *tensor* 进行重排。

说明: 参数 *shape* 中的-1 表示该维度值是从其他维度和输入的元素数量推断出来的。

参数:

- **input** (Tensor) - 输入 *tensor*。
- **shape** (Union[tuple[int], list[int], Tensor[int]]) - 新 *shape*。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]], mindspore.float32)
>>> # case1: Parameter `shape` does not contain -1.
>>> output = mindspore.ops.reshape(input, (3, 2))
>>> print(output)
[[-0.1  0.3]
 [ 3.6  0.4]
 [ 0.5 -3.2]]
>>> # case2: Parameter `shape` contains -1.
>>> output = mindspore.ops.reshape(input, (-1, 6))
>>> print(output)
[[-0.1  0.3  3.6  0.4  0.5 -3.2]]
```

mindspore.ops.reverse_sequence

`mindspore.ops.reverse_sequence(x, seq_lengths, seq_dim, batch_dim=0)`

对输入序列进行部分反转。

参数:

- **x** (Tensor) - 输入 tensor。
- **seq_lengths** (Tensor) - 指定反转长度。
- **seq_dim** (int) - 指定反转的维度。
- **batch_dim** (int) - 指定切片维度，默认 0。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.float32)
>>> seq_lengths = mindspore.tensor([1, 2, 3])
>>> output = mindspore.ops.reverse_sequence(x, seq_lengths, seq_dim=1)
>>> print(output)
[[1. 2. 3.]
 [5. 4. 6.]
 [9. 8. 7.]]
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.float32)
>>> seq_lengths = mindspore.tensor([1, 2, 3])
>>> output = mindspore.ops.reverse_sequence(x, seq_lengths, seq_dim=0, batch_dim=1)
>>> print(output)
[[1. 5. 9.]
 [4. 2. 6.]
 [7. 8. 3.]]
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.float32)
>>> seq_lengths = mindspore.tensor([2, 2, 3])
>>> output = mindspore.ops.reverse_sequence(x, seq_lengths, seq_dim=1)
>>> print(output)
[[2. 1. 3.]
 [5. 4. 6.]
 [9. 8. 7.]]
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.float32)
>>> seq_lengths = mindspore.tensor([3, 2, 3])
>>> output = mindspore.ops.reverse_sequence(x, seq_lengths, seq_dim=1)
>>> print(output)
[[3. 2. 1.]
 [5. 4. 6.]
 [9. 8. 7.]]
>>> x = mindspore.tensor([[1, 2, 3, 4], [5, 6, 7, 8]], mindspore.float32)
>>> seq_lengths = mindspore.tensor([4, 4])
>>> output = mindspore.ops.reverse_sequence(x, seq_lengths, seq_dim=1)
>>> print(output)
[[4. 3. 2. 1.]
 [8. 7. 6. 5.]]
```

mindspore.ops.roll

`mindspore.ops.roll(input, shifts, dims=None)`

按维度移动 `tensor` 的元素。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **shifts** (Union[list(int), tuple(int), int]) - 元素移动量。
- **dims** (Union[list(int), tuple(int), int], 可选) - 指定移动维度，默认 `None`，表示将输入 `tensor` 展平后再进行计算，然后将计算结果 `reshape` 为输入的 `shape`。

返回：

Tensor

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([0, 1, 2, 3, 4], mindspore.float32)
>>> # case1: Parameter `shifts` is positive
>>> output = mindspore.ops.roll(input, shifts=2, dims=0)
>>> print(output)
[3. 4. 0. 1. 2.]
>>> # case2: Parameter `shifts` is negative
>>> output = mindspore.ops.roll(input, shifts=-2, dims=0)
>>> print(output)
[2. 3. 4. 0. 1.]
```

mindspore.ops.row_stack

`mindspore.ops.row_stack(tensors)`

`mindspore.ops.vstack()` 的别名。

支持平台：

Ascend GPU CPU

mindspore.ops.scatter

`mindspore.ops.scatter(input, axis, index, src)`

根据指定索引将 `src` 中的值更新到 `input` 中并返回输出。有关更多详细信息，请参阅 `mindspore.ops.tensor_scatter_elements()`。

说明：如果 `src` 为 `tensor`，则仅当 `src` 的 `shape` 和 `index` 的 `shape` 相同时支持求反向梯度。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **axis** (int) - 要进行更新操作的轴。

- **index** (Tensor) - 要进行更新操作的索引。
- **src** (Tensor, float) - 指定对 *input* 进行更新操作的数据。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3, 4, 5]], dtype=mindspore.float32)
>>> src = mindspore.tensor([[8, 8]], dtype=mindspore.float32)
>>> index = mindspore.tensor([[2, 4]], dtype=mindspore.int64)
>>> out = mindspore.ops.scatter(input=input, axis=1, index=index, src=src)
>>> print(out)
[[1. 2. 8. 4. 8.]]
>>> input = mindspore.tensor(mindspore.ops.zeros((5, 5)), dtype=mindspore.float32)
>>> src = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=mindspore.float32)
>>> index = mindspore.tensor([[0, 0, 0], [2, 2, 2], [4, 4, 4]], dtype=mindspore.int64)
>>> out = mindspore.ops.scatter(input=input, axis=0, index=index, src=src)
>>> print(out)
[[1. 2. 3. 0. 0.]
 [0. 0. 0. 0. 0.]
 [4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0.]
 [7. 8. 9. 0. 0.]]
>>> input = mindspore.tensor(mindspore.ops.zeros((5, 5)), dtype=mindspore.float32)
>>> src = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], dtype=mindspore.float32)
>>> index = mindspore.tensor([[0, 2, 4], [0, 2, 4], [0, 2, 4]], dtype=mindspore.int64)
>>> out = mindspore.ops.scatter(input=input, axis=1, index=index, src=src)
>>> print(out)
[[1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6.]
 [7. 0. 8. 0. 9.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

mindspore.ops.scatter_nd

mindspore.ops.scatter_nd(*indices, updates, shape*)

根据指定的索引将更新值散布到新 tensor 上。

使用给定的 *shape* 创建一个空 tensor，并将 *updates* 的值通过索引 (*indices*) 来设置空 tensor 的值。空 tensor 的秩为 P ，*indices* 的秩为 Q 。

shape 为 $(s_0, s_1, \dots, s_{P-1})$ ，其中 $P \geq 1$ 。

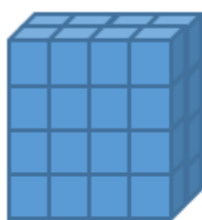
indices 的 *shape* 为 $(i_0, i_1, \dots, i_{Q-2}, N)$ ，其中 $Q \geq 2$ 且 $N \leq P$ 。

indices 的最后一个维度（长度为 N ）表示沿着空 tensor 的第 N 个维度进行切片。

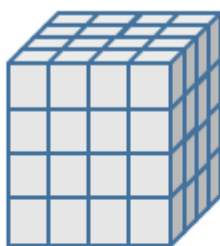
updates 是一个秩为 $Q - 1 + P - N$ 的 tensor，*shape* 为 $(i_0, i_1, \dots, i_{Q-2}, s_N, s_{N+1}, \dots, s_{P-1})$ 。

如果 *indices* 包含重复的下标，则 *updates* 的值会被累加到同一个位置上。

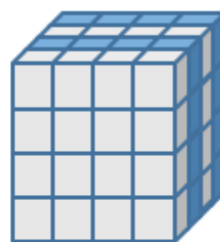
在秩为 3 的第一个维度中插入两个新值矩阵的计算过程如下图所示：



updates



shape



output

参数:

- **indices** (Tensor) - 指定新 tensor 中散布的索引。索引的秩须至少为 2, 并且 $indices.shape[-1] \leq len(shape)$ 。
- **updates** (Tensor) - 指定更新 tensor。shape 为 $indices.shape[:-1] + shape[indices.shape[-1]:]$ 。
- **shape** (tuple[int]) - 指定输出 tensor 的 shape。shape 不能为空, 且 shape 中的任何元素的值都必须大于等于 1。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> indices = mindspore.tensor([[0], [2]], mindspore.int32)
>>> updates = mindspore.tensor([[[1, 1, 1, 1], [2, 2, 2, 2],
...                               [3, 3, 3, 3], [4, 4, 4, 4]],
...                               [[1, 1, 1, 1], [2, 2, 2, 2],
...                               [3, 3, 3, 3], [4, 4, 4, 4]]], mindspore.float32)
>>> shape = (4, 4, 4)
>>> output = mindspore.ops.scatter_nd(indices, updates, shape)
>>> print(output)
[[[1. 1. 1. 1.]
  [2. 2. 2. 2.]
  [3. 3. 3. 3.]
  [4. 4. 4. 4.]]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
 [[1. 1. 1. 1.]
  [2. 2. 2. 2.]
  [3. 3. 3. 3.]
  [4. 4. 4. 4.]]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]]
>>> indices = mindspore.tensor([[0, 1], [1, 1]], mindspore.int32)
```

(下页继续)

(续上页)

```

>>> updates = mindspore.tensor([3.2, 1.1], mindspore.float32)
>>> shape = (3, 3)
>>> output = mindspore.ops.scatter_nd(indices, updates, shape)
>>> # In order to facilitate understanding, explain the operator pseudo-operation process.
>>> # step by step:
>>> # Step 1: Generate an empty Tensor of the specified shape according to the shape
>>> # [
>>> #     [0. 0. 0.]
>>> #     [0. 0. 0.]
>>> #     [0. 0. 0.]
>>> # ]
>>> # Step 2: Modify the data at the specified location according to the indicators
>>> # 0th row of indices is [0, 1], 0th row of updates is 3.2.
>>> # means that the empty tensor in the 0th row and 1st col set to 3.2
>>> # [
>>> #     [0. 3.2. 0.]
>>> #     [0. 0. 0.]
>>> #     [0. 0. 0.]
>>> # ]
>>> # 1th row of indices is [1, 1], 1th row of updates is 1.1.
>>> # means that the empty tensor in the 1th row and 1st col set to 1.1
>>> # [
>>> #     [0. 3.2. 0.]
>>> #     [0. 1.1 0.]
>>> #     [0. 0. 0.]
>>> # ]
>>> # The final result is as follows:
>>> print(output)
[[0. 3.2 0.]
 [0. 1.1 0.]
 [0. 0. 0.]]

```

mindspore.ops.select

`mindspore.ops.select` (*condition, input, other*)

根据条件判断 `tensor` 中的元素的值，来决定输出中的相应元素是从 *input*（如果元素值为 `True`）还是从 *other*（如果元素值为 `False`）中选择。

该算法可以被定义为：

$$out_i = \begin{cases} input_i, & \text{if } condition_i \\ other_i, & \text{otherwise} \end{cases}$$

参数：

- **condition** (`Tensor[bool]`) - 条件 `tensor`。
- **input** (`Union[Tensor, int, float]`) - 第一个被选择的 `tensor` 或数字。
- **other** (`Union[Tensor, int, float]`) - 第二个被选择的 `tensor` 或数字。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case1: Both `input` and `other` are tensor
>>> cond = mindspore.tensor([True, False])
>>> x = mindspore.tensor([2,3], mindspore.float32)
>>> y = mindspore.tensor([1,2], mindspore.float32)
>>> output1 = mindspore.ops.select(cond, x, y)
>>> print(output1)
[2. 2.]
>>> # case2: Both `input` and `other` are number
>>> output2 = mindspore.ops.select(cond, input=1, other=2)
>>> print(output2)
[1 2]
>>> # case3: `input` is tensor and `other` is number
>>> output3 = mindspore.ops.select(cond, x, other=3)
>>> print(output3)
[2. 3.]
```

mindspore.ops.select_scatter

mindspore.ops.select_scatter (*input, src, axis, index*)

将 *src* 中的值散布到 *input* 指定轴 *axis* 的指定位置 *index* 上。

参数:

- **input** (Tensor) - 输入 tensor。
- **src** (Tensor) - 源 tensor。
- **axis** (int) - 指定轴。
- **index** (int) - 在指定轴上散布的位置。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.ops.zeros((2, 3, 3))
>>> src = mindspore.ops.ones((2, 3))
>>> output = mindspore.ops.select_scatter(input, src, axis=1, index=1)
>>> print(output)
[[[0. 0. 0.]
  [1. 1. 1.]
  [0. 0. 0.]]
 [[0. 0. 0.]
  [1. 1. 1.]
  [0. 0. 0.]]]
```


`mindspore.ops.sequence_mask`

`mindspore.ops.sequence_mask` (*lengths*, *maxlen=None*)

返回一个表示每个单元的前 *N* 个位置的掩码 `tensor`，内部元素数据类型为 `bool`。

如果 *lengths* 的 `shape` 为 $(d_1, d_2, \dots, d_n)$ ，则生成的 `tensor` 掩码拥有数据类型，其 `shape` 为 $(d_1, d_2, \dots, d_n, maxlen)$ 。且 $mask[i_1, i_2, \dots, i_n, j] = (j < lengths[i_1, i_2, \dots, i_n])$ 。

参数：

- **lengths** (`Tensor`) - 输入 `tensor`。此 `tensor` 中的所有值都应小于或等于 *maxlen*，大于 *maxlen* 的值将被视为 *maxlen*。
- **maxlen** (`int`) - 指定返回 `tensor` 的长度。

返回：

`Tensor`，`shape` 为 *lengths.shape* + (*maxlen*,)。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: When maxlen is assigned
>>> x = mindspore.tensor([1, 2, 3, 4])
>>> output = mindspore.ops.sequence_mask(x, 5)
>>> print(output)
[[ True False False False False]
 [ True  True False False False]
 [ True  True  True False False]
 [ True  True  True  True False]]
>>> # case 2: When there is 0 in x
>>> x = mindspore.tensor([[1, 3], [2, 0]])
>>> output = mindspore.ops.sequence_mask(x, 5)
>>> print(output)
[[ True False False False False]
 [ True  True  True False False]
 [[ True  True False False False]
 [False False False False False]]]
>>> # case 3: when the maxlen is not assigned
>>> x = mindspore.tensor([[1, 3], [2, 4]])
>>> output = mindspore.ops.sequence_mask(x)
>>> print(output)
[[[ True False False False ]
 [ True  True  True False ]]
 [[ True  True False False ]
 [ True  True  True  True ]]]
```

mindspore.ops.shape

`mindspore.ops.shape(input_x)`

返回输入 tensor 的 shape。

参数:

- **input_x** (Tensor) - 输入 tensor。

返回:

`tuple[int]`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.ops.ones(shape=[3, 2, 1])
>>> output = mindspore.ops.shape(input_x)
>>> print(output)
(3, 2, 1)
```

mindspore.ops.shuffle

`mindspore.ops.shuffle(x, seed=None)`

沿着第一维随机打乱 tensor。

参数:

- **x** (Tensor) - 输入 tensor。
- **seed** (int, 可选) - 随机种子，默认 None，等同于 0。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1, 2, 3, 4], mindspore.float32)
>>> output = mindspore.ops.shuffle(x, seed=1)
>>> print(output)
[3. 4. 2. 1.]
```

mindspore.ops.size

`mindspore.ops.size(input_x)`

计算输入 `tensor` 的元素总数。

参数:

- **input_x** (Tensor) - 输入 `tensor`。

返回:

`int`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([[2, 2], [2, 2]], mindspore.float32)
>>> output = mindspore.ops.size(input_x)
>>> print(output)
4
```

mindspore.ops.slice

`mindspore.ops.slice(input_x, begin, size)`

根据指定 `shape` 对输入 `tensor` 进行切片。

说明: `begin` 的起始值为 0, `size` 的起始值为 1。

如果 `size[i]` 为 -1, 则维度 `i` 中的所有剩余元素都包含在切片中。这相当于 `size[i] = input_x.shape(i) - begin[i]`。

参数:

- **input_x** (Tensor) - 输入 `tensor`。
- **begin** (Union[tuple, list]) - 切片的起始位置, 表示每个维度的偏移。
- **size** (Union[tuple, list]) - 切片的大小。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> data = mindspore.tensor([[[[1, 1, 1], [2, 2, 2]],
...                           [[3, 3, 3], [4, 4, 4]],
...                           [[5, 5, 5], [6, 6, 6]]], mindspore.int32)
>>> output = mindspore.ops.slice(data, (1, 0, 0), (1, 1, 3))
>>> print(output)
[[[3 3 3]]]
>>> output = mindspore.ops.slice(data, (1, 0, 0), (1, 1, 2))
```

(下页继续)

(续上页)

```
>>> print(output)
[[[3 3]]]
>>> output = mindspore.ops.slice(data, (1, 0, 0), (1, 1, 1))
>>> print(output)
[[[3]]]
>>> output = mindspore.ops.slice(data, (1, 1, 0), (1, 1, 3))
>>> print(output)
[[[4 4 4]]]
>>> output = mindspore.ops.slice(data, (1, 0, 1), (1, 1, 2))
>>> print(output)
[[[3 3]]]
```

mindspore.ops.slice_scatter

`mindspore.ops.slice_scatter(input, src, axis=0, start=None, end=None, step=1)`

沿指定轴将源 `tensor` 嵌入到切片后的目标 `tensor`。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **src** (Tensor) - 用于嵌入到 `input` 的源 `tensor`。
- **axis** (int, 可选) - 指定轴，默认 0。
- **start** (int, 可选) - 指定轴中，嵌入的开始索引，默认 `None`，表示 `start` 为 0。
- **end** (int, 可选) - 指定轴中，嵌入的结束索引，默认 `None`，表示 `end` 是 `input` 在指定维度的长度。
- **step** (int, 可选) - 嵌入时跳过的步长，默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> a = mindspore.ops.zeros((4, 6))
>>> b = mindspore.ops.ones((4, 3))
>>> output = mindspore.ops.slice_scatter(input=a, src=b, axis=1, start=0, end=5, step=2)
>>> print(output)
[[1. 0. 1. 0. 1. 0.]
 [1. 0. 1. 0. 1. 0.]
 [1. 0. 1. 0. 1. 0.]
 [1. 0. 1. 0. 1. 0.]]
```

mindspore.ops.sort

`mindspore.ops.sort(input_x, axis=-1, descending=False)`

对输入 `tensor` 指定轴的元素进行排序。

说明：当前 Ascend 后端只支持对最后一维进行排序。

参数：

- **input_x** (Tensor) - 输入 `tensor`。
- **axis** (int, 可选) - 指定轴，默认 `-1`，表示指定最后一维。
- **descending** (bool, 可选) - 排序方式。True 表示按降序排列，否则按升序排列，默认 `False`。

警告：目前能良好支持的数据类型有：`float16`、`uint8`、`int8`、`int16`、`int32`、`int64`。如果使用 `float32`，可能产生精度误差。

返回：

一个由 `tensor` 组成的 `tuple(sorted_tensor, indices)`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[8, 2, 1], [5, 9, 3], [4, 6, 7]], mindspore.float16)
>>> output = mindspore.ops.sort(x)
>>> # The output below is based on the Ascend platform.
>>> print(output)
(Tensor(shape=[3, 3], dtype=Float16, value=
[[ 1.0000e+00,  2.0000e+00,  8.0000e+00],
[ 3.0000e+00,  5.0000e+00,  9.0000e+00],
[ 4.0000e+00,  6.0000e+00,  7.0000e+00]]), Tensor(shape=[3, 3], dtype=Int32, value=
[[2, 1, 0],
[2, 0, 1],
[0, 1, 2]]))
```

mindspore.ops.space_to_batch_nd

`mindspore.ops.space_to_batch_nd(input_x, block_size, paddings)`

将空间维度分块，并在批次维度重排 `tensor`。

$$n' = n * (block\_size[0] * \dots * block\_size[M])$$

$$w'_i = (w_i + paddings[i][0] + paddings[i][1]) // block\_size[i]$$

说明：

- 此函数将输入的空间维度 $[1, \dots, M]$ 按 `block_size` 拆分成小块，并重排至批次维度（默认第 0 维）。在分块前，输入的空间维度会根据 `paddings` 填充零。

- 若输入的形狀为 $(n, c_1, \dots, c_k, w_1, \dots, w_M)$ ，则输出的形狀为 $(n', c_1, \dots, c_k, w'_1, \dots, w'_M)$ 。
- 如果 *block_size* 为 list 或者 tuple，其长度 *M* 为空间维度的长度。如果 *block_size* 为整数，那么所有空间维度分割的个数均为 *block_size*。在 Ascend 平台 *M* 必须为 2。

参数：

- **input_x** (Tensor) - 输入张量。
- **block_size** (Union[list(int), tuple(int), int]) - 空间维度分块大小。
- **paddings** (Union[tuple, list]) - 空间维度的填充大小。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> block_size = [2, 2]
>>> paddings = [[0, 0], [0, 0]]
>>> input_x = mindspore.tensor([[[[1, 2], [3, 4]]]], mindspore.float32)
>>> output = mindspore.ops.space_to_batch_nd(input_x, block_size, paddings)
>>> print(output)
[[[1.]]]
[[[2.]]]
[[[3.]]]
[[[4.]]]
```

mindspore.ops.sparse_segment_mean

mindspore.ops.sparse_segment_mean (*x*, *indices*, *segment_ids*)

计算输入张量稀疏段的平均值。

$$output_i = \frac{\sum_j x_{indices[j]}}{N}$$

其中 *N* 为满足 *segment_ids[j] == i* 的元素个数，如果 *segment_ids* 中没有某个值 *i*，则 *output[i] = 0*。

说明：

- 在 CPU 平台，*segment_ids* 必须是升序，*indices* 必须在 $[0, x.shape[0])$ 范围内。
- 在 GPU 平台，对于 *segment_ids* 无需排序，但可能导致未定义的行为。超出范围的 *indices* 将被忽略。

参数：

- **x** (Tensor) - 输入 tensor，至少为一维。
- **indices** (Tensor) - 指定索引，一维 tensor。
- **segment_ids** (Tensor) - 一维 tensor，必须按升序排列，可包含重复值。

返回：

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[0, 1, 2], [1, 2, 3], [3, 6, 7]], dtype=mindspore.float32)
>>> indices = mindspore.tensor([0, 1, 2], dtype=mindspore.int32)
>>> segment_ids = mindspore.tensor([1, 2, 2], dtype=mindspore.int32)
>>> out = mindspore.ops.sparse_segment_mean(x, indices, segment_ids)
>>> print(out)
[[0. 0. 0.]
 [0. 1. 2.]
 [2. 4. 5.]]
```

mindspore.ops.split

`mindspore.ops.split` (*tensor*, *split_size_or_sections*, *axis=0*)

沿指定轴将输入 `tensor` 切分成多个子 `tensor`。

参数:

- **tensor** (Tensor) - 输入 `tensor`。
- **split_size_or_sections** (Union[int, tuple(int), list(int)]) - 切分后子 `tensor` 的大小。
- **axis** (int, 可选) - 指定轴, 默认 0。

说明:

- 如果 `split_size_or_sections` 是 `int` 类型, `tensor` 将被均匀切分成块, 每块大小为 `split_size_or_sections`, 若 `tensor.shape[axis]` 不能被 `split_size_or_sections` 整除, 则最后一块大小为余数;
- 如果 `split_size_or_sections` 是 `tuple` 或 `list` 类型, `tensor` 将沿 `axis` 轴被切分成 `len(split_size_or_sections)` 块, 大小为 `split_size_or_sections`。

返回:

一个由 `tensor` 组成的 `tuple`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case1: `split_size_or_sections` is an int type
>>> input_x = mindspore.ops.arange(10).astype("float32")
>>> output = mindspore.ops.split(tensor=input_x, split_size_or_sections=3)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value=[0.00000000e+00, 1.00000000e+00, 2.00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value=[3.00000000e+00, 4.00000000e+00, 5.00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value=[6.00000000e+00, 7.00000000e+00, 8.00000000e+00]),
 Tensor(shape=[1], dtype=Float32, value=[9.00000000e+00]))
```

(下页继续)

(续上页)

```
>>> # case2: `split_size_or_sections` is a list type
>>> output = mindspore.ops.split(tensor=input_x, split_size_or_sections=[3, 3, 4])
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value=[0.00000000e+00, 1.00000000e+00, 2.00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value=[3.00000000e+00, 4.00000000e+00, 5.00000000e+00]),
 Tensor(shape=[4], dtype=Float32, value=[6.00000000e+00, 7.00000000e+00, 8.00000000e+00, 9.
↪00000000e+00]))
```

mindspore.ops.squeeze

`mindspore.ops.squeeze` (*input*, *axis=None*)

删除输入 `tensor` 中长度为 1 的轴。

说明:

- 请注意，在动态图模式下，输出 `tensor` 将与输入 `tensor` 共享数据，并且没有 `tensor` 数据复制过程。
- 维度索引从 0 开始，并且必须在 `[-input.ndim, input.ndim)` 范围内。
- 在 GE 模式下，只支持对 `input shape` 中大小为 1 的维度进行压缩。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **axis** (Union[int, tuple(int), list(int)]) - 待删除的轴，默认 `None`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.ops.ones(shape=[3, 2, 1])
>>> output = mindspore.ops.squeeze(input)
>>> print(output)
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```


mindspore.ops.stack

`mindspore.ops.stack` (*tensors*, *axis=0*)
 在指定轴上对输入的 *tensors* 进行堆叠。

参数:

- **tensors** (Union[tuple, list]) - 输入 *tensors*。
- **axis** (int) - 指定轴, 默认 0。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x1 = mindspore.tensor([0, 1], mindspore.float32)
>>> input_x2 = mindspore.tensor([2, 3], mindspore.float32)
>>> output = mindspore.ops.stack((input_x1, input_x2), 0)
>>> print(output)
[[0. 1.]
 [2. 3.]]
```

mindspore.ops.strided_slice

`mindspore.ops.strided_slice` (*input_x*, *begin*, *end*, *strides*, *begin_mask=0*, *end_mask=0*, *ellipsis_mask=0*,
new_axis_mask=0, *shrink_axis_mask=0*)

对输入 *tensor* 根据步长和索引进行切片提取。

说明:

- *begin*、*end* 和 *strides* 的 *shape* 必须相同。
- *begin*、*end* 和 *strides* 是一维 *tensor*, 且 *shape* 小于或等于 *input_x* 的维度。

例: *input_x* 是 *shape* 为 (5,6,7) 的三维 *tensor*, *begin* 为 (1, 3, 2), *end* 为 (3, 5, 6), *strides* 为 (1, 1, 2)。切片时, 第零维从索引 1 开始取到 3, 步长为 1; 第一维从索引 3 开始取到 5, 步长为 1; 第二维从索引 2 开始取到 6, 步长为 2。相当于 Python 式切片 *input_x[1:3, 3:5, 2:6:2]*。

如果 *begin*、*end* 和 *strides* 的长度小于 *input_x* 的维度, 则缺失的维度默认切取所有的元素, 相当于 *begin* 用 0 补足, *end* 用相应维度的长度补足, *strides* 用 1 补足。

例: *input_x* 是 *shape* 为 (5,6,7) 的三维 *tensor*, *begin* 为 (1, 3), *end* 为 (3, 5), *strides* 为 (1, 1)。切片时, 第零维从索引 1 开始取到 3, 步长为 1; 第一维从索引 3 开始取到 5, 步长为 1; 第二维从索引 0 开始取到 6, 步长为 1。相当于 Python 式切片 *input_x[1:3, 3:5, 0:7]*。

mask (掩码) 的工作原理:

对于每个特定的 *mask*, 内部先将其转化为二进制表示, 然后倒序排布后进行计算。比如, 对于一个 *shape* 为 (5,6,7) 的 *tensor*, *mask* 设置为 3, 3 转化为二进制表示为 0b011, 倒序后为 0b110, 则该 *mask* 只在第零维和第一维产生作用。下面各自举例说明, 为简化表达, 后面提到的 *mask* 都表示转换为二进制并且倒序后的值。

- *begin_mask* 和 *end_mask*

如果 *begin_mask* 的第 *i* 位为 1, 则忽略 *begin[i]*, 第 *i* 维从索引 0 开始取; 若 *end_mask* 的第 *i* 位为 1, 则忽略 *end[i]*, 结束的位置为可以取到的最大范围。对于 shape 为 (5, 6, 7, 8) 的 tensor, 若 *begin_mask* 为 0b110, *end_mask* 为 0b011, 将得到切片 *input_x[0:3, 0:6, 2:7:2]*。

- *ellipsis_mask*

如果 *ellipsis_mask* 的第 *i* 位为 1, 则将在其他维度之间插入所需的任意数量的未指定维度。*ellipsis_mask* 中只允许一个非零位。对于 shape 为 (5, 6, 7, 8) 的 tensor, *input_x[2:, ..., :6]* 等同于 *input_x[2:5, :, :, 0:6]*, *input_x[2:, ...]* 等同于 *input_x[2:5, :, :, :]*。

- *new_axis_mask*

如果 *new_axis_mask* 的第 *i* 位为 1, 则在输出的第 *i* 维添加新的长度为 1 的维度, 并忽略第 *i* 维的 *begin*、*end* 和 *strides*。对于 shape 为 (5, 6, 7) 的 tensor, 若 *new_axis_mask* 为 0b010, 则第二维将新增一维, 输出 shape 为 (5, 1, 6, 7) 的 tensor。

- *shrink_axis_mask*

如果 *shrink_axis_mask* 的第 *i* 位为 1, 则第 *i* 维被收缩掉, 忽略 *begin[i]*、*end[i]* 和 *strides[i]* 索引处的值。对于 shape 为 (5, 6, 7) 的 tensor, 若 *shrink_axis_mask* 为 0b010, 则第一维收缩掉, 相当于切片 *x[:, 5, :]* 使得输出 shape 为 (5, 7)。

说明: *new_axis_mask* 和 *shrink_axis_mask* 不建议同时使用, 可能会产生预料之外的结果。

参数:

- **input_x** (Tensor) - 输入 tensor。
- **begin** (tuple[int]) - 指定开始切片的索引。
- **end** (tuple[int]) - 指定结束切片的索引。
- **strides** (tuple[int]) - 指定各维度切片的步长。
- **begin_mask** (int, 可选) - 切片的起始索引掩码。默认 0。
- **end_mask** (int, 可选) - 切片的结束索引掩码。默认 0。
- **ellipsis_mask** (int, 可选) - 维度掩码, 值为 1 说明不需要进行切片操作。为 int 型掩码。默认 0。
- **new_axis_mask** (int, 可选) - 切片的新增维度掩码。默认 0。
- **shrink_axis_mask** (int, 可选) - 切片的收缩维度掩码。为 int 型掩码。默认 0。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[[[1, 1, 1],
...                             [2, 2, 2]],
...                             [[3, 3, 3],
...                             [4, 4, 4]],
...                             [[5, 5, 5],
```

(下一页继续)

(续上页)

```

... [6, 6, 6]])
>>> mindspore.ops.strided_slice(input, [1, 0, 0], [2, 1, 3], [1, 1, 1])
Tensor(shape=[1, 1, 3], dtype=Int64, value=
[[[3, 3, 3]])
>>> mindspore.ops.strided_slice(input, [1, 0, 0], [2, 2, 3], [1, 1, 1])
Tensor(shape=[1, 2, 3], dtype=Int64, value=
[[[3, 3, 3],
  [4, 4, 4]])
>>> mindspore.ops.strided_slice(input, [1, -1, 0], [2, -3, 3], [1, -1, 1])
Tensor(shape=[1, 2, 3], dtype=Int64, value=
[[[4, 4, 4],
  [3, 3, 3]])

```

mindspore.ops.sum

`mindspore.ops.sum(input, dim=None, keepdim=False, *, dtype=None)`

计算 `tensor` 在指定维度上元素的和。

说明: `Tensor` 类型的 `dim` 仅用作兼容旧版本, 不推荐使用。

参数:

- **input** (`Tensor`) - 输入 `tensor`。
- **dim** (`Union[None, int, tuple(int), list(int), Tensor]`) - 指定维度, 默认 `None`。
- **keepdim** (`bool`) - 输出 `tensor` 是否保留维度, 默认 `False`。

说明: 如果 `dim` 为 `None`, 对 `tensor` 中的所有元素求和; 如果 `dim` 为 `tuple`, `list` 或 `Tensor`, 将对 `dim` 中所有维度求和。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> x = mindspore.tensor([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
... [4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
... [7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]],
mindspore.float32)
>>> out = mindspore.ops.sum(input=x)
>>> print(out)
270.0
>>> out = mindspore.ops.sum(input=x, dim=1)

```

(下页继续)

(续上页)

```

>>> print(out)
[[ 6.  6.  6.  6.  6.  6.]
 [15. 15. 15. 15. 15. 15.]
 [24. 24. 24. 24. 24. 24.]]
>>> out = mindspore.ops.sum(input=x, dim=2)
>>> print(out)
[[ 6. 12. 18.]
 [24. 30. 36.]
 [42. 48. 54.]]
>>> out = mindspore.ops.sum(input=x, dim=[1, 2])
>>> print(out)
[ 36. 90. 144.]
>>> out = mindspore.ops.sum(input=x, dim=2, keepdim=True)
>>> print(out)
[[[ 6.]
  [12.]
  [18.]]
 [[24.]
  [30.]
  [36.]]
 [[42.]
  [48.]
  [54.]]]
>>> print(out.ndim)
3

```

mindspore.ops.swapaxes

`mindspore.ops.swapaxes` (*input, axis0, axis1*)

交换 `tensor` 的两个轴。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **axis0** (int) - 第一个轴。
- **axis1** (int) - 第二个轴。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> input = mindspore.tensor(mindspore.ops.ones([2, 3, 4]))
>>> output = mindspore.ops.swapaxes(input, 0, 2)
>>> print(output.shape)
(4, 3, 2)

```

mindspore.ops.swapdims

`mindspore.ops.swapdims(input, dim0, dim1)`

交换 `tensor` 的两个维度。该函数和 `mindspore.ops.swapaxes()` 功能一致。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **dim0** (int) - 第一个维度。
- **dim1** (int) - 第二个维度。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor(mindspore.ops.ones([2, 3, 4]))
>>> output = mindspore.ops.swapdims(input, 0, 2)
>>> print(output.shape)
(4, 3, 2)
```

mindspore.ops.tensor_scatter_add

`mindspore.ops.tensor_scatter_add(input_x, indices, updates)`

返回一个新 `tensor`, 根据指定索引和更新值对 `input_x` 进行加法更新。

$$output[indices] = input\_x + update$$

说明: 如果 `indices` 中的值超出输入 `input_x` 索引范围:

- GPU 平台上相应的 `updates` 不会更新到 `input_x` 且不会抛出索引错误。
- CPU 平台上直接抛出索引错误。
- Ascend 平台不支持越界检查, 若越界可能会造成未知错误。

参数:

- **input_x** (Tensor) - 输入 `tensor`。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]], mindspore.float32)
>>> indices = mindspore.tensor([[0, 0], [0, 0]], mindspore.int32)
>>> updates = mindspore.tensor([1.0, 2.2], mindspore.float32)
>>> output = mindspore.ops.tensor_scatter_add(input_x, indices, updates)
>>> print(output)
[[ 3.1  0.3  3.6]
 [ 0.4  0.5 -3.2]]
```

mindspore.ops.tensor_scatter_div

mindspore.ops.tensor_scatter_div (*input_x, indices, updates*)

返回一个新 **tensor**，根据指定索引和更新值对 *input_x* 进行除法更新。

$$output[indices] = input\_x \div update$$

说明:

- 如果 *indices* 中的值超出输入 *input_x* 索引范围：
 - GPU 平台上相应的 *updates* 不会更新到 *input_x* 且不会抛出索引错误。
 - CPU 平台上直接抛出索引错误。
 - Ascend 平台不支持越界检查，若越界可能会造成未知错误。
- 算子无法处理除 0 异常，用户需保证 *updates* 中没有 0 值。

参数:

- **input_x** (Tensor) - 输入 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]], mindspore.float32)
>>> indices = mindspore.tensor([[0, 0], [0, 0]], mindspore.int32)
>>> updates = mindspore.tensor([1.0, 2.0], mindspore.float32)
>>> output = mindspore.ops.tensor_scatter_div(input_x, indices, updates)
>>> print(output)
[[-0.05  0.3  3.6 ]
 [ 0.4   0.5 -3.2 ]]
```

mindspore.ops.tensor_scatter_max

`mindspore.ops.tensor_scatter_max(input_x, indices, updates)`

返回一个新 `tensor`，根据指定索引和更新值对 `input_x` 进行最大值更新。

$$output[indices] = \max(input\_x, update)$$

说明：如果 `indices` 中的值超出输入 `input_x` 索引范围：

- GPU 平台上相应的 `updates` 不会更新到 `input_x` 且不会抛出索引错误。
- CPU 平台上直接抛出索引错误。
- Ascend 平台不支持越界检查，若越界可能会造成未知错误。

参数：

- **input_x** (Tensor) - 输入 `tensor`。
- **indices** (Tensor) - 指定索引，其秩至少为 2。
- **updates** (Tensor) - 更新值。

返回：

Tensor

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> input_x = mindspore.tensor([[1, 2, 3], [4, 5, 6]])
>>> indices = mindspore.tensor([[0, 0], [1, 1]])
>>> updates = mindspore.tensor([5, 5])
>>> mindspore.ops.tensor_scatter_max(input_x, indices, updates)
Tensor(shape=[2, 3], dtype=Int64, value=
[[5, 2, 3],
 [4, 5, 6]])
```

mindspore.ops.tensor_scatter_min

`mindspore.ops.tensor_scatter_min(input_x, indices, updates)`

返回一个新 `tensor`，根据指定索引和更新值对 `input_x` 进行最小值更新。

$$output[indices] = \min(input\_x, update)$$

说明：如果 `indices` 中的值超出输入 `input_x` 索引范围：

- GPU 平台上相应的 `updates` 不会更新到 `input_x` 且不会抛出索引错误。
- CPU 平台上直接抛出索引错误。
- Ascend 平台不支持越界检查，若越界可能会造成未知错误。

参数:

- **input_x** (Tensor) - 输入 tensor。
- **indices** (Tensor) - 指定索引, 其秩至少为 2。
- **updates** (Tensor) - 更新值。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([[1, 2, 3], [4, 5, 6]], mindspore.float32)
>>> indices = mindspore.tensor([[0, 0], [1, 1]])
>>> updates = mindspore.tensor([5, 1], mindspore.float32)
>>> mindspore.ops.tensor_scatter_min(input_x, indices, updates)
Tensor(shape=[2, 3], dtype=Float32, value=
[[ 1.00000000e+00,  2.00000000e+00,  3.00000000e+00],
 [ 4.00000000e+00,  1.00000000e+00,  6.00000000e+00]])
```

mindspore.ops.tensor_scatter_mul

`mindspore.ops.tensor_scatter_mul(input_x, indices, updates)`

根据指定索引和更新值对 *input_x* 进行乘法更新。

$$output[indices] = input\_x \times update$$

说明:

- 如果 *indices* 的某些值超出 *input_x* 的维度范围, 则相应的 *updates* 不会更新为 *input_x*, 而不是抛出索引错误。
-

参数:

- **input_x** (Tensor) - 输入 tensor。
- **indices** (Tensor) - 指定索引, 其秩至少为 2。
- **updates** (Tensor) - 更新值。

返回:

Tensor

支持平台:

GPU CPU

样例:


```
>>> import mindspore
>>> input_x = mindspore.tensor([[1, 2, 3], [4, 5, 6]])
>>> indices = mindspore.tensor([[0, 0], [1, 1]])
>>> updates = mindspore.tensor([5, 5])
>>> mindspore.ops.tensor_scatter_mul(input_x, indices, updates)
Tensor(shape=[2, 3], dtype=Int64, value=
[[ 5,  2,  3],
 [ 4, 25,  6]])
```

mindspore.ops.tensor_scatter_sub

mindspore.ops.tensor_scatter_sub (*input_x, indices, updates*)

返回一个新 **tensor**，根据指定索引和更新值对 *input_x* 进行减法更新。

$$output[indices] = input\_x - update$$

说明：如果 *indices* 中的值超出输入 *input_x* 索引范围：GPU 平台上相应的 *updates* 不会更新到 *input_x* 且不会抛出索引错误；CPU 平台上直接抛出索引错误；Ascend 平台不支持越界检查，若越界可能会造成未知错误。

参数：

- **input_x** (Tensor) - 输入 **tensor**。
- **indices** (Tensor) - 指定索引，其秩至少为 2。
- **updates** (Tensor) - 更新值。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3], [4, 5, 6]], mindspore.float32)
>>> indices = mindspore.tensor([[0, 0], [1, 1]])
>>> updates = mindspore.tensor([5, 5], mindspore.float32)
>>> mindspore.ops.tensor_scatter_sub(input, indices, updates)
Tensor(shape=[2, 3], dtype=Float32, value=
[[-4.00000000e+00,  2.00000000e+00,  3.00000000e+00],
 [ 4.00000000e+00,  0.00000000e+00,  6.00000000e+00]])
```

mindspore.ops.tensor_scatter_elements

`mindspore.ops.tensor_scatter_elements(input_x, indices, updates, axis=0, reduction='none')`

返回一个新 `tensor`，根据指定索引和更新值对 `input_x` 中的元素进行指定操作（替换或相加）。

不支持隐式类型转换。

举例：一个三维输入 `tensor` 的返回为

```

output[indices[i][j][k]][j][k] = updates[i][j][k] # if axis == 0, reduction == "none"

output[i][indices[i][j][k]][k] += updates[i][j][k] # if axis == 1, reduction == "add"

output[i][j][indices[i][j][k]] = updates[i][j][k] # if axis == 2, reduction == "none"

```

警告：

- 如果 `indices` 中有多个索引向量对应于同一位置，则输出中该位置值是不确定的。
- 在 Ascend 平台上，目前仅支持 `reduction` 设置为 "none" 的实现。
- 在 Ascend 平台上，`input_x` 仅支持 `float16` 和 `float32` 两种数据类型。
- 这是一个实验性 API，后续可能修改或删除。

说明：如果 `indices` 的值超出 `input_x` 索引上下界，则相应的 `updates` 不会更新到 `input_x`，也不会抛出索引错误。仅当 `indices` 的 `shape` 和 `updates` 的 `shape` 相同时支持求反向梯度。

参数：

- **input_x** (Tensor) - 输入 `tensor`，秩大于等于 1。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。
- **axis** (int) - 指定索引所在的轴。默认 0。
- **reduction** (str) - 指定操作。支持 `none`，`add`。默认 `none`。
 - 如果为 `none`，将把 `updates` 中的元素赋值给 `input_x`。
 - 如果为 `add`，将把 `updates` 中的元素与 `input_x` 中的元素相加。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> input_x = mindspore.tensor([[1, 2, 3, 4, 5]])
>>> indices = mindspore.tensor([[2, 4]])
>>> updates = mindspore.tensor([[8, 8]])
>>> output = mindspore.ops.tensor_scatter_elements(input_x, indices, updates, axis=1,
↪reduction="none")

```

(下页继续)

(续上页)

```
>>> print(output)
[[1 2 8 4 8]]
>>> output = mindspore.ops.tensor_scatter_elements(input_x, indices, updates, axis=1,
↪reduction="add")
>>> print(output)
[[ 1  2 11  4 13]]
```

mindspore.ops.tensor_split

`mindspore.ops.tensor_split(input, indices_or_sections, axis=0)`

根据指定索引或份数，将输入 `tensor` 拆分成多个子 `tensor`。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 指定索引或份数。
 - 如果是 `int` 类型，输入 `tensor` 将被拆分成 `indices_or_sections` 份。
 - * 如果 `input.shape[axis]` 能被 `indices_or_sections` 整除，那么子切片为相同大小 `input.shape[axis]/n`。
 - * 如果 `input.shape[axis]` 不能被 `indices_or_sections` 整除，那么前 `input.shape[axis] mod n` 个切片的大小为 `input.shape[axis]//n + 1`，其余切片的大小为 `input.shape[axis]//n`。
 - 如果是 `tuple(int)` 或 `list(int)` 类型，则表示索引，输入 `tensor` 在索引处被拆分。
- **axis** (int, 可选) - `indices_or_sections` 所在的轴。默认 0。

返回：

由多个 `tensor` 组成的 `tuple`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([0, 1, 2, 3, 4, 5, 6, 7])
>>> mindspore.ops.tensor_split(input, 3)
(Tensor(shape=[3], dtype=Int64, value= [0, 1, 2]),
 Tensor(shape=[3], dtype=Int64, value= [3, 4, 5]),
 Tensor(shape=[2], dtype=Int64, value= [6, 7]))
>>> input = mindspore.tensor([0, 1, 2, 3, 4, 5, 6])
>>> mindspore.ops.tensor_split(input, 3)
(Tensor(shape=[3], dtype=Int64, value= [0, 1, 2]),
 Tensor(shape=[2], dtype=Int64, value= [3, 4]),
 Tensor(shape=[2], dtype=Int64, value= [5, 6]))
>>> mindspore.ops.tensor_split(input, (1, 6))
(Tensor(shape=[1], dtype=Int64, value= [0]),
 Tensor(shape=[5], dtype=Int64, value= [1, 2, 3, 4, 5]),
 Tensor(shape=[1], dtype=Int64, value= [6]))
>>> input = mindspore.tensor([[ 0,  1,  2,  3,  4,  5,  6],
...                           [ 7,  8,  9, 10, 11, 12, 13]])
>>> mindspore.ops.tensor_split(input, 3, axis=1)
(Tensor(shape=[2, 3], dtype=Int64, value=
 [[0, 1, 2],
```

(下页继续)

(续上页)

```

    [7, 8, 9]]),
    Tensor(shape=[2, 2], dtype=Int64, value=
    [[ 3,  4],
     [10, 11]]),
    Tensor(shape=[2, 2], dtype=Int64, value=
    [[ 5,  6],
     [12, 13]]))
>>> mindspore.ops.tensor_split(input, (1, 6), axis=1)
(Tensor(shape=[2, 1], dtype=Int64, value=
[[0],
 [7]]),
 Tensor(shape=[2, 5], dtype=Int64, value=
[[ 1,  2,  3,  4,  5],
 [ 8,  9, 10, 11, 12]]),
 Tensor(shape=[2, 1], dtype=Int64, value=
[[ 6],
 [13]]))

```

mindspore.ops.tile

`mindspore.ops.tile(input, dims)`

通过复制 *dims* 次输入 *tensor* 中的元素来创建新 *tensor*。

返回 *tensor* 的第 *i* 维度有 *input.shape[i] * dims[i]* 个元素，并且 *input* 的值沿第 *i* 维度被复制 *dims[i]* 次。

说明:

- 在 Ascend 平台上，*dims* 参数的个数不大于 8，当前不支持超过 4 个维度同时做被复制的场景。
- 如果 *input.dim = d*，将其相应位置的 *shape* 相乘，返回的 *shape* 为 $(x_1 * y_1, x_2 * y_2, \dots, x_S * y_S)$ 。
- 如果 *input.dim < d*，在 *input* 的 *shape* 的前面填充 1，直到它们的长度一致。例如将 *input* 的 *shape* 设置为 $(1, \dots, x_1, x_2, \dots, x_S)$ ，然后可以将其相应位置的 *shape* 相乘，返回的 *shape* 为 $(1 * y_1, \dots, x_R * y_R, x_S * y_S)$ 。
- 如果 *input.dim > d*，在 *dims* 的前面填充 1，直到它们的长度一致。例如将 *dims* 设置为 $(1, \dots, y_1, y_2, \dots, y_S)$ ，然后可以将其相应位置的 *shape* 相乘，返回的 *shape* 为 $(x_1 * 1, \dots, x_R * y_R, x_S * y_S)$ 。

参数:

- input** (Tensor) - 输入 *tensor*。
- dims** (tuple[int]) - 指定每维度的复制次数。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> input = mindspore.tensor([[1, 2], [3, 4]])
>>> mindspore.ops.tile(input, (2, 3))

```

(下页继续)

(续上页)

```
Tensor(shape=[4, 6], dtype=Int64, value=
[[1, 2, 1, 2, 1, 2],
 [3, 4, 3, 4, 3, 4],
 [1, 2, 1, 2, 1, 2],
 [3, 4, 3, 4, 3, 4]])
>>> mindspore.ops.tile(input, (2, 3, 2))
Tensor(shape=[2, 6, 4], dtype=Int64, value=
[[[1, 2, 1, 2],
  [3, 4, 3, 4],
  [1, 2, 1, 2],
  [3, 4, 3, 4],
  [1, 2, 1, 2],
  [3, 4, 3, 4]]],
 [[1, 2, 1, 2],
  [3, 4, 3, 4],
  [1, 2, 1, 2],
  [3, 4, 3, 4],
  [1, 2, 1, 2],
  [3, 4, 3, 4]]])
```

mindspore.ops.tril

`mindspore.ops.tril(input, diagonal=0)`

将指定对角线上方的元素设置为 0。

参数：

- **input** (Tensor) - 输入 tensor，其秩至少为 2。
- **diagonal** (int, 可选) - 二维 tensor 的指定对角线。默认 0，表示主对角线。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[ 1,  2,  3,  4],
...                           [ 5,  6,  7,  8],
...                           [10, 11, 12, 13],
...                           [14, 15, 16, 17]])
>>> mindspore.ops.tril(input)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  0,  0,  0],
 [ 5,  6,  0,  0],
 [10, 11, 12,  0],
 [14, 15, 16, 17]])
>>> mindspore.ops.tril(input, 1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  2,  0,  0],
 [ 5,  6,  7,  0],
 [10, 11, 12, 13],
 [14, 15, 16, 17]])
```

(下页继续)

(续上页)

```
>>> mindspore.ops.tril(input, -1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 0,  0,  0,  0],
 [ 5,  0,  0,  0],
 [10, 11,  0,  0],
 [14, 15, 16,  0]])
>>> input = mindspore.tensor([[[ 1,  2,  3],
...                             [ 5,  6,  7],
...                             [10, 11, 12]],
...                             [[ 1,  2,  3],
...                             [ 5,  6,  7],
...                             [10, 11, 12]]])
>>> mindspore.ops.tril(input)
Tensor(shape=[2, 3, 3], dtype=Int64, value=
[[[ 1,  0,  0],
 [ 5,  6,  0],
 [10, 11, 12]],
 [[ 1,  0,  0],
 [ 5,  6,  0],
 [10, 11, 12]]])
```

mindspore.ops.triu

`mindspore.ops.triu(input, diagonal=0)`

将指定对角线下方的元素设置为 0。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 tensor。
- **diagonal** (int, 可选) - 二维 tensor 的指定对角线。默认 0，表示主对角线。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[[ 1,  2,  3,  4],
...                             [ 5,  6,  7,  8],
...                             [10, 11, 12, 13],
...                             [14, 15, 16, 17]])
>>> mindspore.ops.triu(input)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  2,  3,  4],
 [ 0,  6,  7,  8],
 [ 0,  0, 12, 13],
 [ 0,  0,  0, 17]])
```

(下页继续)

(续上页)

```

>>> mindspore.ops.triu(input, 1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 0,  2,  3,  4],
 [ 0,  0,  7,  8],
 [ 0,  0,  0, 13],
 [ 0,  0,  0,  0]])
>>> mindspore.ops.triu(input, -1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  2,  3,  4],
 [ 5,  6,  7,  8],
 [ 0, 11, 12, 13],
 [ 0,  0, 16, 17]])
>>> input = mindspore.tensor([[[ 1,  2,  3],
...                             [ 5,  6,  7],
...                             [10, 11, 12]],
...                             [[ 1,  2,  3],
...                             [ 5,  6,  7],
...                             [10, 11, 12]]])
>>> mindspore.ops.triu(input)
Tensor(shape=[2, 3, 3], dtype=Int64, value=
[[[ 1,  2,  3],
 [ 0,  6,  7],
 [ 0,  0, 12]],
 [[ 1,  2,  3],
 [ 0,  6,  7],
 [ 0,  0, 12]])

```

mindspore.ops.transpose

`mindspore.ops.transpose(input, input_perm)`

根据指定的维度排列顺序对输入 `tensor` 进行维度转置。

说明: GPU 和 CPU 平台上, 如果 `input_perm` 的元素值为负数, 则其实际值为 `input_perm[i] + rank(input)`。Ascend 平台不支持 `input_perm` 元素值为负。

参数:

- **input** (Tensor) - 输入 tensor。
- **input_perm** (tuple[int]) - 指定轴的新排列。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> input = mindspore.tensor([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]], mindspore.
↳float32)
>>> output = mindspore.ops.transpose(input, (0, 2, 1))

```

(下页继续)

(续上页)

```
>>> print(output)
[[[ 1.  4.]
   [ 2.  5.]
   [ 3.  6.]]
 [[ 7. 10.]
   [ 8. 11.]
   [ 9. 12.]]]
```

mindspore.ops.unbind

`mindspore.ops.unbind(input, dim=0)`

移除 `tensor` 的指定维度，返回一个沿该维度所有切片的元组。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **dim** (int) - 指定维度。默认 0。

返回：

由多个 `tensor` 组成的 `tuple`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> output = mindspore.ops.unbind(x, dim=0)
>>> print(output)
(Tensor(shape=[3], dtype=Int64, value=[1, 2, 3]), Tensor(shape=[3], dtype=Int64, value=[4, 5,
→ 6]),
 Tensor(shape=[3], dtype=Int64, value=[7, 8, 9]))
```

mindspore.ops.unique

`mindspore.ops.unique(input)`

对输入 `tensor` 中元素去重。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

由两个 `tensor` 组成的 `tuple(output, indices)`。

- **output** (Tensor) - 去重后的输出。
- **indices** (Tensor) - 输入 `tensor` 的元素在 `output` 中的索引。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1, 2, 5, 2], mindspore.int32)
>>> output, indices = mindspore.ops.unique(x)
>>> print(output)
[1 2 5]
>>> print(indices)
[0 1 2 1]
```

mindspore.ops.unique_consecutive

mindspore.ops.unique_consecutive (*input*, *return_inverse=False*, *return_counts=False*, *dim=None*)

对输入 `tensor` 中每组连续且重复的元素去重, 仅保留首个元素。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **return_inverse** (bool, 可选) - 是否返回输入 `tensor` 中各元素映射到输出中的位置。默认 `False`。
- **return_counts** (bool, 可选) - 是否返回每个唯一值出现的次数。默认 `False`。
- **dim** (int, 可选) - 指定去重的维度。默认 `None`, 表示对输入进行展平。

返回:

Tensor 或由多个 `tensor` 组成的 `tuple(output, inverse_indices, counts)`。

- **output** (Tensor) - 去重后的输出。
- **inverse_indices** (Tensor, optional) - 输入 `tensor` 的元素在 `output` 中的索引。
- **counts** (Tensor, optional) - 每个唯一值出现的次数。

支持平台:

Ascend GPU CPU

样例:

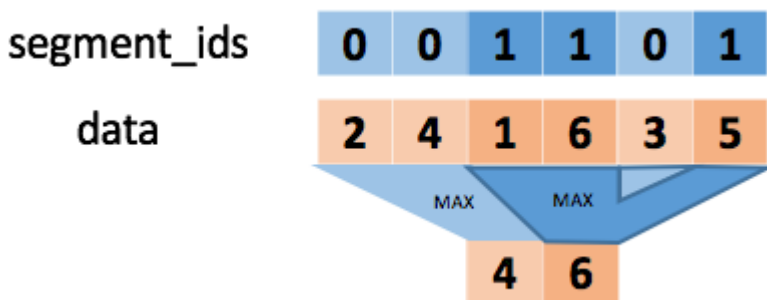
```
>>> import mindspore
>>> x = mindspore.tensor([1, 1, 2, 2, 3, 1, 1, 2], mindspore.int32)
>>> output, inverse_indices, counts = mindspore.ops.unique_consecutive(x, True, True, None)
>>> print(output)
[1 2 3 1 2]
>>> print(inverse_indices)
[0 0 1 1 2 3 3 4]
>>> print(counts)
[2 2 1 2 1]
```

mindspore.ops.unsorted_segment_max

`mindspore.ops.unsorted_segment_max(x, segment_ids, num_segments)`

分段计算输入 tensor 的最大值。

`unsorted_segment_max` 的计算过程如下图所示：



$$\text{output}_i = \max_{j \dots} \text{data}[j \dots]$$

`max` 返回元素 $j \dots$ 中的最大值，其中 $\text{segment\_ids}[j \dots] == i$ 。

说明：

- 如果 `segment_ids` 中不存在 `segment_id i`，则输出 `output[i]` 将使用 `x` 的数据类型的最小值填充。
- `segment_ids` 必须是一个非负 tensor。

参数：

- **x** (Tensor) - 输入 tensor。
- **segment_ids** (Tensor) - 标识每个元素所属段。
- **num_segments** (Union[int, Tensor]) - 分段数量，可以为 int 或零维的 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

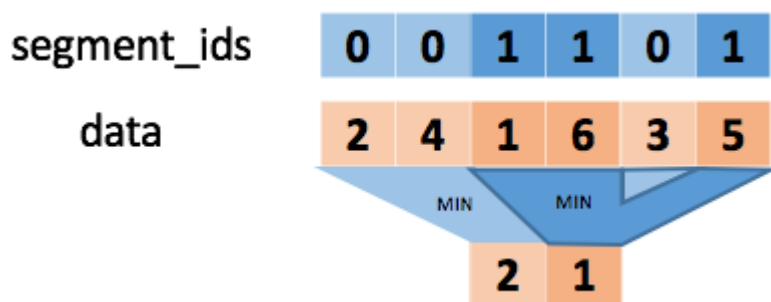
```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [4, 2, 1]])
>>> segment_ids = mindspore.tensor([0, 1, 1])
>>> num_segments = 2
>>> mindspore.ops.unsorted_segment_max(x, segment_ids, num_segments)
Tensor(shape=[2, 3], dtype=Int64, value=
[[1, 2, 3],
 [4, 5, 6]])
```

mindspore.ops.unsorted_segment_min

`mindspore.ops.unsorted_segment_min(x, segment_ids, num_segments)`

分段计算输入 tensor 的最小值。

`unsorted_segment_min` 的计算过程如下图所示：



$$\text{output}_i = \min_{j \dots} \text{data}[j \dots]$$

`min` 返回元素 $j \dots$ 中的最小值，其中 `segment_ids[j...] == i`。

说明：

- 如果 `segment_ids` 中不存在 `segment_id i`，则将使用 `x` 的数据类型的最大值填充输出 `output[i]`。
- `segment_ids` 必须是一个非负 tensor。

参数：

- **x** (Tensor) - 输入 tensor。
- **segment_ids** (Tensor) - 标识每个元素所属段。
- **num_segments** (Union[int, Tensor], 可选) - 分段数量，可以为 int 或零维的 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

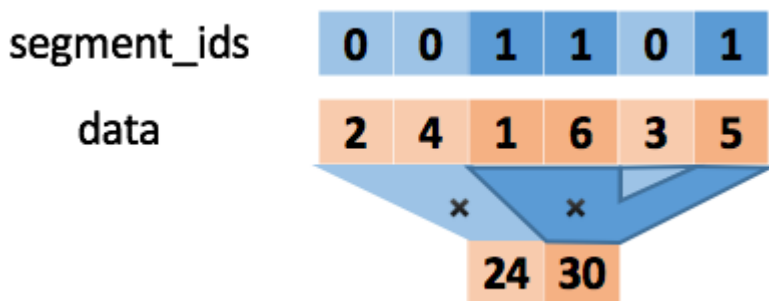
```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [4, 2, 1]])
>>> segment_ids = mindspore.tensor([0, 1, 1])
>>> num_segments = 2
>>> mindspore.ops.unsorted_segment_min(x, segment_ids, num_segments)
Tensor(shape=[2, 3], dtype=Int64, value=
[[1, 2, 3],
 [4, 2, 1]])
```

mindspore.ops.unsorted_segment_prod

`mindspore.ops.unsorted_segment_prod(x, segment_ids, num_segments)`

分段计算输入 tensor 的乘积。

`unsorted_segment_prod` 的计算过程如下图所示：



说明：

- 如果 `segment_ids` 中不存在 `segment_id i`，则将使用 1 填充输出 `output[i]`。
- `segment_ids` 必须是一个非负 tensor。

参数：

- **x** (Tensor) - 输入 tensor。
- **segment_ids** (Tensor) - 标识每个元素所属段。
- **num_segments** (Union[int, Tensor], 可选) - 分段数量，可以为 int 或零维的 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

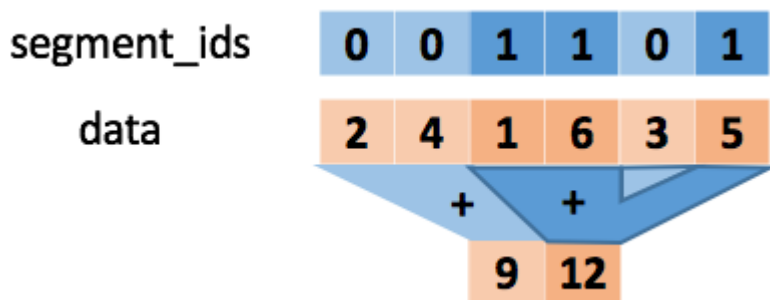
```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3], [4, 5, 6], [4, 2, 1]])
>>> segment_ids = mindspore.tensor([0, 1, 0])
>>> num_segments = 2
>>> mindspore.ops.unsorted_segment_prod(x, segment_ids, num_segments)
Tensor(shape=[2, 3], dtype=Int64, value=
[[4, 4, 3],
 [4, 5, 6]])
```

mindspore.ops.unsorted_segment_sum

`mindspore.ops.unsorted_segment_sum(input_x, segment_ids, num_segments)`

分段计算输入 tensor 的和。

`unsorted_segment_sum` 的计算过程如下图所示：



$$\text{output}[i] = \sum_{\text{segment\_ids}[j]==i} \text{data}[j, \dots]$$

其中 $j, \dots$ 是代表元素索引的 tuple。`segment_ids` 确定输入 tensor 元素的分段。`segment_ids` 不需要排序，也不需要覆盖 `num_segments` 范围内的所有值。

说明：

- 如果 `segment_ids` 中不存在 `segment_id i`，则对输出 `output[i]` 填充 0。
- 在 Ascend 平台上，如果 `segment_id` 的值小于 0 或大于输入 tensor 的 shape 的长度，将触发执行错误。

如果给定的 `segment_ids i` 的和为空，则 `output[i] = 0`。如果 `segment_ids` 元素为负数，将忽略该值。`num_segments` 必须等于不同 `segment_id` 的数量。

参数：

- **input_x** (Tensor) - 输入 tensor。
- **segment_ids** (Tensor) - 标识每个元素所属段。
- **num_segments** (Union[int, Tensor], 可选) - 分段数量，可以为 int 或零维的 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input_x = mindspore.tensor([1, 2, 3, 4])
>>> segment_ids = mindspore.tensor([0, 0, 1, 2])
>>> num_segments = 3
>>> mindspore.ops.unsorted_segment_sum(input_x, segment_ids, num_segments)
Tensor(shape=[3], dtype=Int64, value= [3, 3, 4])
>>> input_x = mindspore.tensor([1, 2, 3, 4, 2, 5])
>>> segment_ids = mindspore.tensor([0, 0, 1, 2, 3, 4])
```

(下页继续)

(续上页)

```
>>> num_segments = 6
>>> mindspore.ops.unsorted_segment_sum(input_x, segment_ids, num_segments)
Tensor(shape=[6], dtype=Int64, value= [3, 3, 4, 2, 5, 0])
```

mindspore.ops.unsqueeze

`mindspore.ops.unsqueeze(input, dim)`

在输入 `tensor` 的指定维度上增加一维。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **dim** (int) - 指定维度。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 3, 4])
>>> mindspore.ops.unsqueeze(input, 0)
Tensor(shape=[1, 4], dtype=Int64, value=
[[1, 2, 3, 4]])
>>> mindspore.ops.unsqueeze(input, 1)
Tensor(shape=[4, 1], dtype=Int64, value=
[[1],
 [2],
 [3],
 [4]])
```

mindspore.ops.unstack

`mindspore.ops.unstack(input_x, axis=0)`

按指定轴对输入 `tensor` 进行拆分。

参数：

- **input_x** (Tensor) - 输入 `tensor`。
- **axis** (int) - 指定轴。默认 0。

返回：

多个 `tensor` 组成的 `tuple`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 1, 1, 1], [2, 2, 2, 2]])
>>> mindspore.ops.unstack(input, 0)
(Tensor(shape=[4], dtype=Int64, value= [1, 1, 1, 1]),
 Tensor(shape=[4], dtype=Int64, value= [2, 2, 2, 2]))
```

mindspore.ops.view_as_real

mindspore.ops.view_as_real (*input*)

返回一个最后一维大小为 2 的实数 **tensor**，由输入 **tensor** 中复数元素的实部和虚部组成。

参数：

- **input** (Tensor) - 输入复数 **tensor**。

返回：

一个实数 **tensor**。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([2+1j, 2+3j, 2-1j, 2])
>>> output = mindspore.ops.view_as_real(input)
>>> print(output)
[[ 2.  1.]
 [ 2.  3.]
 [ 2. -1.]
 [ 2.  0.]]
```

mindspore.ops.vsplit

mindspore.ops.vsplit (*input, indices_or_sections*)

根据 *indices_or_sections* 将至少有两维的输入 **tensor** 垂直分割成多个子 **tensor**。

等同于 *axis = 0* 时的 *ops.tensor_split*。

参数：

- **input** (Tensor) - 输入 **tensor**。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 参考 *mindspore.ops.tensor_split()* 中的 *indices_or_sections* 参数。

返回：

由多个 **tensor** 组成的 **tuple**。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[ 0,  1,  2,  3],
...                           [ 4,  5,  6,  7],
...                           [ 8,  9, 10, 11],
...                           [12, 13, 14, 15]])
>>> mindspore.ops.vsplit(input, 2)
(Tensor(shape=[2, 4], dtype=Int64, value=
[[0, 1, 2, 3],
 [4, 5, 6, 7]]),
 Tensor(shape=[2, 4], dtype=Int64, value=
[[ 8,  9, 10, 11],
 [12, 13, 14, 15]]))
>>> mindspore.ops.vsplit(input, [3, 6])
(Tensor(shape=[3, 4], dtype=Int64, value=
[[ 0,  1,  2,  3],
 [ 4,  5,  6,  7],
 [ 8,  9, 10, 11]]),
 Tensor(shape=[1, 4], dtype=Int64, value=
[[12, 13, 14, 15]]),
 Tensor(shape=[0, 4], dtype=Int64, value=
))
```

mindspore.ops.vstack

`mindspore.ops.vstack(inputs)`

将多个 `tensor` 沿着垂直方向进行堆叠。

等同于将输入按第一个轴进行拼接。一维 `tensor` (N ,) 重新排列为 $(1, N)$ ，然后按第一个轴进行拼接。

参数：

- **inputs** (Union(List[`tensor`], Tuple[`tensor`])) - 一维或二维输入 `tensors`。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x1 = mindspore.tensor([1, 2, 3])
>>> x2 = mindspore.tensor([4, 5, 6])
>>> mindspore.ops.vstack((x1, x2))
Tensor(shape=[2, 3], dtype=Int64, value=
[[1, 2, 3],
 [4, 5, 6]])
>>> x1 = mindspore.tensor([[1],[2],[3]])
>>> x2 = mindspore.tensor([[4],[5],[6]])
>>> mindspore.ops.vstack([x1, x2])
Tensor(shape=[6, 1], dtype=Int64, value=
[[1],
 [2],
 [3],
 [4],
```

(下页继续)

(续上页)

```
[5],
[6]])
```

mindspore.ops.where

`mindspore.ops.where(condition, input, other)`

返回一个 `tensor`，其中元素根据 `condition` 从 `input` 或 `other` 中选择。

支持广播。

$$output_i = \begin{cases} input_i, & \text{if } condition_i \\ other_i, & \text{otherwise} \end{cases}$$

参数：

- **condition** (Tensor[bool]) - 如果为 `True`，选取 `input` 中的元素，否则选取 `other` 中的元素。
- **input** (Union[Tensor, Scalar]) - 在 `condition` 为 `True` 的索引处选择的值。
- **other** (Union[Tensor, Scalar]) - 当 `condition` 为 `False` 的索引处选择的值。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[0, 1],
...                             [2, 3]])
>>> other = mindspore.tensor([[1, 1],
...                             [1, 1]])
>>> condition = input < 3
>>> mindspore.ops.where(condition, input, other)
Tensor(shape=[2, 2], dtype=Int64, value=
[[0, 1],
 [2, 1]])
```

mindspore.ops.cross

`mindspore.ops.cross(input, other, dim=None)`

按指定维度计算两个输入 `tensor` 的叉积。

说明: `input` 和 `other` 必须有相同的 `shape`，且指定的 `dim` 上 `size` 必须为 3。如果不指定 `dim`，则使用第一个 `size` 为 3 的维度。

参数：

- **input** (Tensor) - 第一个输入 `tensor`。
- **other** (Tensor) - 第二个输入 `tensor`。

- **dim** (int, 可选) - 指定维度。默认 None。

返回：

Tensor

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
>>> other = mindspore.tensor([[4, 5, 6], [4, 5, 6], [4, 5, 6]])
>>> mindspore.ops.cross(input, other)
Tensor(shape=[3, 3], dtype=Int64, value=
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]])
>>> mindspore.ops.cross(input, other, dim=1)
Tensor(shape=[3, 3], dtype=Int64, value=
[[-3, 6, -3],
 [-3, 6, -3],
 [-3, 6, -3]])
```

mindspore.ops.renorm

`mindspore.ops.renorm(input, p, axis, maxnorm)`

返回一个 tensor，其中按指定轴计算每个子 tensor 的 p 范数小于等于 $maxnorm$ 。如果大于，返回子 tensor 上的原始值除以子 tensor 的 p 范数，然后再乘以 $maxnorm$ 。

参数：

- **input** (Tensor) - 输入 tensor。
- **p** (int) - 范数计算的幂。
- **axis** (int) - 指定计算轴。
- **maxnorm** (float32) - 指定的最大范数。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1., 1, 1], [2, 2, 2], [3, 3, 3]])
>>> output = mindspore.ops.norm(input, 1, 1)
>>> print(output)
[3. 6. 9.]
>>> output = mindspore.ops.renorm(input, 1, 0, 6.)
>>> print(output)
[[1. 1. 1.]
```

(下页继续)

(续上页)

```
[2. 2. 2.]
[2. 2. 2.]]
```

2.1.4 类型转换

接口名	概述	支持平台
<code>mindspore.ops.cast</code>	转换输入 <code>tensor</code> 的数据类型。	Ascend GPU CPU
<code>mindspore.ops.is_tensor</code>	判断输入对象是否为 <code>mindspore.Tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.scalar_to_tensor</code>	将标量转换为指定数据类型的 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.tuple_to_array</code>	将 <code>tuple</code> 转换为 <code>tensor</code> 。	Ascend GPU CPU

mindspore.ops.cast

`mindspore.ops.cast` (*input*, *dtype*)
转换输入 `tensor` 的数据类型。

说明：将复数转换为 `bool` 类型的时候，不考虑复数的虚部，只要实部不为零，返回 `True`，否则返回 `False`。

- 参数：
- **input** (`Union[Tensor, Number]`) - 输入 `tensor` 或者数值型数据。
 - **dtype** (`dtype.Number`) - 转换后的数据类型。仅支持常量值。

返回：
`Tensor`
支持平台：
Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> dtype = mindspore.float64
>>> output = mindspore.ops.cast(input, dtype)
>>> print(output.dtype)
Float64
>>> print(output)
[1. 2. 3.]
```

mindspore.ops.is_tensor

`mindspore.ops.is_tensor(obj)`

判断输入对象是否为`mindspore.Tensor`。

参数:

- **obj** (Object) - 输入对象。

返回:

Bool

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> a = mindspore.tensor([1.9, 2.2, 3.1])
>>> mindspore.ops.is_tensor(a)
True
```

mindspore.ops.scalar_to_tensor

`mindspore.ops.scalar_to_tensor(input_x, dtype=mstype.float32)`

将标量转换为指定数据类型的 `tensor`。

参数:

- **input_x** (Union[bool, int, float]) - 输入标量。只能是常量值。
- **dtype** (mindspore.dtype) - 指定数据类型。只能是常量值。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> data = 1
>>> output = mindspore.ops.scalar_to_tensor(data, mindspore.float32)
>>> print(output)
1.0
```

mindspore.ops.tuple_to_array

`mindspore.ops.tuple_to_array(input_x)`
将 tuple 转换为 tensor。

说明：如果 tuple 中第一个数据类型为 int，则输出 tensor 的数据类型为 int。否则，输出 tensor 的数据类型为 float。

参数：

- **input_x** (tuple) - 数值型组成的 tuple。仅支持常量值。

返回：

Tensor
支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input_x = (1, 2, 3)
>>> output = mindspore.ops.tuple_to_array(input_x)
>>> print(type(output))
<class 'mindspore.common.tensor.Tensor'>
>>> print(output)
[1 2 3]
```

2.1.5 梯度剪裁

接口名	概述	支持平台
<code>mindspore.ops.clip_by_global_norm</code>	通过权重梯度总和的比率来裁剪多个 tensor 的值。	Ascend GPU CPU
<code>mindspore.ops.clip_by_value</code>	将输入的 Tensor 值裁剪到指定的最小值和最大值之间。	Ascend GPU CPU
<code>mindspore.ops.clip_by_norm</code>	基于范数对输入 Tensor 进行裁剪。	Ascend GPU CPU

mindspore.ops.clip_by_global_norm

`mindspore.ops.clip_by_global_norm(x, clip_norm=1.0, use_norm=None)`
通过权重梯度总和的比率来裁剪多个 tensor 的值。

说明：

- 在半自动并行模式或自动并行模式下，如果输入是梯度，那么将会自动汇聚所有设备上的梯度的平方和。

参数：

- **x** (Union(tuple[Tensor], list[Tensor])) - 要被剪裁的数据。
- **clip_norm** (Union(float, int)) - 裁剪比率，应大于 0。默认 1.0。

- **use_norm** (None) - 全局范数。目前只支持 None，默认 None。

返回：

多个 **tensor** 组成的 **tuple**。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x1 = mindspore.tensor([[2., 3.], [1., 2.]], dtype=mindspore.float32)
>>> x2 = mindspore.tensor([[1., 4.], [3., 1.]], dtype=mindspore.float32)
>>> input_x = (x1, x2)
>>> out = mindspore.ops.clip_by_global_norm(input_x, 1.0)
>>> print(out)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 2.98142403e-01,  4.47213590e-01],
 [ 1.49071202e-01,  2.98142403e-01]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 1.49071202e-01,  5.96284807e-01],
 [ 4.47213590e-01,  1.49071202e-01]]))
```

mindspore.ops.clip_by_value

mindspore.ops.clip_by_value (*x*, *clip_value_min*=None, *clip_value_max*=None)

将输入的 **Tensor** 值裁剪到指定的最小值和最大值之间。

限制 *x* 的范围，其最小值为 *clip_value_min*，最大值为 *clip_value_max*。

$$out_i = \begin{cases} clip\_value\_max & \text{if } x_i \geq clip\_value\_max \\ x_i & \text{if } clip\_value\_min < x_i < clip\_value\_max \\ clip\_value\_min & \text{if } x_i \leq clip\_value\_min \end{cases}$$

说明：

- *clip_value_min* 和 *clip_value_max* 不能同时为 None；
- 当 *clip_value_min* 为 None，*clip_value_max* 不为 None 时，Tensor 中大于 *clip_value_max* 的元素会变为 *clip_value_max*；
- 当 *clip_value_min* 不为 None，*clip_value_max* 为 None 时，Tensor 中小于 *clip_value_min* 的元素会变为 *clip_value_min*；
- 当 *clip_value_min* 大于 *clip_value_max* 时，Tensor 中所有元素的值会被置为 *clip_value_max*；
- *x*，*clip_value_min* 和 *clip_value_max* 的数据类型需支持隐式类型转换，且不能为布尔型。

参数：

- **x** (Union(Tensor, list[Tensor], tuple[Tensor])) - *clip_by_value* 的输入，类型为 **Tensor**、**Tensor** 的列表或元组。支持任意维度的 **Tensor**。
- **clip_value_min** (Union(Tensor, float, int)) - 指定最小值。默认值为 None。
- **clip_value_max** (Union(Tensor, float, int)) - 指定最大值。默认值为 None。

返回：

Tensor、Tensor 的列表或元组，表示裁剪后的 Tensor。其 shape 和数据类型与 x 相同。

异常：

- **ValueError** - 如果 `clip_value_min` 和 `clip_value_max` 都为 None。
- **TypeError** - 如果 x 的数据类型不在 Tensor、list[Tensor] 或 tuple[Tensor] 中。
- **TypeError** - 如果 `clip_value_min` 的数据类型不为 None、Tensor、float 或 int。
- **TypeError** - 如果 `clip_value_max` 的数据类型不为 None、Tensor、float 或 int。

支持平台：

Ascend GPU CPU

样例：

```
>>> # case 1: the data type of x is Tensor
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> min_value = Tensor(5, mindspore.float32)
>>> max_value = Tensor(20, mindspore.float32)
>>> x = Tensor(np.array([[1., 25., 5., 7.], [4., 11., 6., 21.]]), mindspore.float32)
>>> output = ops.clip_by_value(x, min_value, max_value)
>>> print(output)
[[ 5. 20.  5.  7.]
 [ 5. 11.  6. 20.]]
>>> # case 2: the data type of x is list[Tensor]
>>> min_value = 5
>>> max_value = 20
>>> x = Tensor(np.array([[1., 25., 5., 7.], [4., 11., 6., 21.]]), mindspore.float32)
>>> y = Tensor(np.array([[1., 25., 5., 7.], [4., 11., 6., 21.]]), mindspore.float32)
>>> output = ops.clip_by_value([x,y], min_value, max_value)
>>> for out in output:
...     print(out)
[[ 5. 20.  5.  7.]
 [ 5. 11.  6. 20.]]
[[ 5. 20.  5.  7.]
 [ 5. 11.  6. 20.]]
```

`mindspore.ops.clip_by_norm`

`mindspore.ops.clip_by_norm(x, max_norm, norm_type=2.0, error_if_nonfinite=False)`

基于范数对输入 Tensor 进行裁剪。计算时先将所有输入元素的范数连接成向量，再计算该向量的范数。

说明：接口适用于梯度裁剪场景，输入仅支持 float 类型入参。

参数：

- **x** (Union[Tensor, list[Tensor], tuple[Tensor]]) - 需要裁剪的输入。
- **max_norm** (Union[float, int]) - 该组网络参数的范数上限。
- **norm_type** (Union[float, int], 可选) - 范数类型。默认 2.0。
- **error_if_nonfinite** (bool, 可选) - 若为 True，当 x 中元素的总范数为 nan、inf 或 -inf 时抛出异常；若为 False，则不抛出异常。默认 False。

返回：

Tensor、Tensor 的列表或元组，表示裁剪后的 Tensor。

异常：

- **RuntimeError** - 如果 x 的总范数为 nan、inf 或-inf。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> x = Tensor([[0.8748, 0.1425, 0.0076], [0.7721, 0.4084, 0.0552], [4.6376, 0.2914, 2.
↪1120]])
>>> out = ops.clip_by_norm(x, max_norm=1)
>>> print(out)
[[0.16650201 0.02712224 0.00144652]
 [0.14695495 0.07773139 0.0105063 ]
 [0.8826814 0.0554626 0.40198016]]
```

2.2 数学运算函数

2.2.1 逐元素运算

接口名	概述	支持平台
<code>mindspore.ops.abs</code>	逐元素计算输入 tensor 的绝对值。	Ascend GPU CPU
<code>mindspore.ops.absolute</code>	<code>mindspore.ops.abs()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.accumulate_n</code>	逐元素计算列表中各个 tensor 的和。	Ascend GPU
<code>mindspore.ops.acos</code>	逐元素计算输入 tensor 的反余弦。	Ascend GPU CPU
<code>mindspore.ops.arccos</code>	<code>mindspore.ops.acos()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.acosh</code>	逐元素计算输入 tensor 的反双曲余弦。	Ascend GPU CPU
<code>mindspore.ops.add</code>	逐元素计算两个输入 tensor 的和。	Ascend GPU CPU
<code>mindspore.ops.addcddiv</code>	<i>tensor2</i> 对 <i>tensor1</i> 逐元素相除，将结果乘以标量值 <i>value</i> ，并加到 <i>input</i> 。	Ascend GPU CPU
<code>mindspore.ops.addcmul</code>	<i>tensor1</i> 与 <i>tensor2</i> 的逐元素相乘，将结果乘以标量值 <i>value</i> ，并加到 <i>input</i> 。	Ascend GPU CPU
<code>mindspore.ops.addmv</code>	将矩阵 <i>mat</i> 和向量 <i>vec</i> 相乘，结果与输入 <i>input</i> 相加。	Ascend GPU CPU
<code>mindspore.ops.addn</code>	逐元素计算列表中各个 tensor 的和。	Ascend GPU CPU
<code>mindspore.ops.angle</code>	逐元素计算输入 tensor 的辐角。	Ascend GPU CPU
<code>mindspore.ops.arccosh</code>	<code>mindspore.ops.acosh()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.arcsin</code>	<code>mindspore.ops.asin()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.arcsinh</code>	<code>mindspore.ops.asinh()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.arctan</code>	<code>mindspore.ops.atan()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.arctanh</code>	<code>mindspore.ops.atanh()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.arctan2</code>	<code>mindspore.ops.atan2()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.asin</code>	逐元素计算输入 tensor 的反正弦。	Ascend GPU CPU
<code>mindspore.ops.asinh</code>	逐元素计算输入 tensor 的反双曲正弦。	Ascend GPU CPU
<code>mindspore.ops.atan</code>	逐元素计算输入 tensor 的反正切值。	Ascend GPU CPU

下页继续

表 6 - 续上页

<code>mindspore.ops.atan2</code>	逐元素计算 <code>input/other</code> 的反正切值。	Ascend GPU CPU
<code>mindspore.ops.atanh</code>	逐元素计算输入 <code>tensor</code> 的反双曲正切值。	Ascend GPU CPU
<code>mindspore.ops.atleast_1d</code>	当输入中的 <code>tensor</code> 为零维时, 返回它的一维 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.atleast_2d</code>	当输入中的 <code>tensor</code> 小于二维时, 返回它的二维 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.atleast_3d</code>	当输入中的 <code>tensor</code> 小于三维时, 返回它的三维 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.bessel_i0</code>	逐元素计算输入 <code>tensor</code> 的第一类零阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_i0e</code>	逐元素计算输入 <code>tensor</code> 的指数缩放第一类零阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_i1</code>	逐元素计算输入 <code>tensor</code> 的第一类一阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_i1e</code>	逐元素计算输入 <code>tensor</code> 的指数缩放第一类一阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_j0</code>	逐元素计算输入 <code>tensor</code> 的第一类零阶贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_j1</code>	逐元素计算输入 <code>tensor</code> 的第一类一阶贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_k0</code>	逐元素计算输入 <code>tensor</code> 的第二类零阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_k0e</code>	逐元素计算输入 <code>tensor</code> 的指数缩放第二类零阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_k1</code>	逐元素计算输入 <code>tensor</code> 的第二类一阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_k1e</code>	逐元素计算输入 <code>tensor</code> 的指数缩放第二类一阶修正贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_y0</code>	逐元素计算输入 <code>tensor</code> 的第二类零阶贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bessel_y1</code>	逐元素计算输入 <code>tensor</code> 的第二类一阶贝塞尔函数值。	GPU CPU
<code>mindspore.ops.bitwise_and</code>	逐元素对两个输入 <code>tensor</code> 做与运算。	Ascend GPU CPU
<code>mindspore.ops.bitwise_left_shift</code>	逐元素对输入 <code>input</code> 进行左移位运算, 移动的位数由 <code>other</code> 指定。	Ascend GPU CPU
<code>mindspore.ops.bitwise_or</code>	逐元素对两个输入 <code>tensor</code> 做或运算。	Ascend GPU CPU
<code>mindspore.ops.bitwise_right_shift</code>	逐元素对输入 <code>input</code> 进行右移位运算, 移动的位数由 <code>other</code> 指定。	Ascend GPU CPU
<code>mindspore.ops.bitwise_xor</code>	逐元素对两个输入 <code>tensor</code> 做异或运算。	Ascend GPU CPU
<code>mindspore.ops.ceil</code>	逐元素对输入 <code>tensor</code> 做向上取整。	Ascend GPU CPU
<code>mindspore.ops.clamp</code>	将输入 <code>tensor</code> 的所有元素限制在范围 <code>[min, max]</code> 内。	Ascend GPU CPU
<code>mindspore.ops.clip</code>	<code>mindspore.ops.clamp()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.combinations</code>	返回输入 <code>tensor</code> 中所有长度为 <code>r</code> 的子序列。	Ascend GPU CPU
<code>mindspore.ops.copysign</code>	创建一个 <code>float tensor</code> , 由 <code>x</code> 的绝对值和 <code>other</code> 的符号组成。	Ascend GPU CPU
<code>mindspore.ops.cos</code>	逐元素计算输入 <code>tensor</code> 的余弦。	Ascend GPU CPU
<code>mindspore.ops.cosh</code>	逐元素计算输入 <code>tensor</code> 的双曲余弦。	Ascend GPU CPU
<code>mindspore.ops.cosine_similarity</code>	沿指定维度计算两个输入 <code>tensor</code> 之间的余弦相似度。	Ascend GPU CPU

下页继续

表 6 – 续上页

<code>mindspore.ops.cov</code>	返回输入 tensor 的协方差矩阵, 其中输入 tensor 的行表示变量, 列表示观测值。	Ascend GPU CPU
<code>mindspore.ops.diff</code>	按指定轴计算输入 tensor 的 n 阶前向差分。	Ascend GPU CPU
<code>mindspore.ops.deg2rad</code>	逐元素地将输入 tensor 从度数制转换为弧度制。	Ascend GPU CPU
<code>mindspore.ops.digamma</code>	逐元素计算输入 tensor 的伽玛函数的对数导数。	GPU CPU
<code>mindspore.ops.div</code>	逐元素计算第一个输入 tensor 除以第二个输入 tensor 。	Ascend GPU CPU
<code>mindspore.ops.divide</code>	<code>mindspore.ops.div()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.erf</code>	逐元素计算输入 tensor 的高斯误差。	Ascend GPU CPU
<code>mindspore.ops.erfc</code>	逐元素计算输入 tensor 的互补误差。	Ascend GPU CPU
<code>mindspore.ops.erfinv</code>	逐元素计算输入 tensor 的逆误差。	Ascend GPU CPU
<code>mindspore.ops.exp</code>	逐元素计算输入 tensor 的指数。	Ascend GPU CPU
<code>mindspore.ops.exp2</code>	逐元素计算输入 tensor 以 2 为底的指数。	Ascend GPU CPU
<code>mindspore.ops.expm1</code>	逐元素计算输入 tensor 的指数, 然后减 1。	Ascend GPU CPU
<code>mindspore.ops.floor</code>	逐元素对输入 tensor 做向下取整。	Ascend GPU CPU
<code>mindspore.ops.floor_div</code>	<code>mindspore.ops.floor_divide()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.floor_divide</code>	逐元素计算 <i>input</i> 除以 <i>other</i> , 并向下取整。	Ascend GPU CPU
<code>mindspore.ops.floor_mod</code>	逐元素计算第一个输入除以第二个输入, 并向下取余。	Ascend GPU CPU
<code>mindspore.ops.float_power</code>	逐元素计算以第一个输入为底, 第二个输入为指数。	GPU CPU
<code>mindspore.ops.fmax</code>	逐元素计算输入 tensor 的最大值。	CPU
<code>mindspore.ops.fmod</code>	逐元素计算第一个输入除以第二个输入的余数。	Ascend GPU CPU
<code>mindspore.ops.frac</code>	返回输入 tensor 中每个元素的小数部分。	Ascend GPU CPU
<code>mindspore.ops.gcd</code>	逐元素计算两个输入 tensor 的最大公约数。	Ascend GPU CPU
<code>mindspore.ops.hypot</code>	给定直角三角形的边, 逐元素计算其斜边。	Ascend GPU CPU
<code>mindspore.ops.igamma</code>	计算正则化的下层不完全伽马函数。	Ascend GPU CPU
<code>mindspore.ops.igammac</code>	计算正则化的上层不完全伽马函数。	Ascend GPU CPU
<code>mindspore.ops.imag</code>	返回输入 tensor 每个元素的虚部。	Ascend GPU CPU
<code>mindspore.ops.i0</code>	详情请参考 <code>mindspore.ops.bessel_i0()</code> 。	GPU CPU
<code>mindspore.ops.inv</code>	逐元素计算输入 tensor 的倒数。	Ascend GPU CPU
<code>mindspore.ops.invert</code>	逐元素按位翻转。	Ascend GPU CPU
<code>mindspore.ops.lcm</code>	逐元素计算两个输入 tensor 的最小公倍数。	Ascend GPU CPU
<code>mindspore.ops.ldexp</code>	逐元素将输入 tensor 乘以 2^{other} 。	Ascend GPU CPU
<code>mindspore.ops.lerp</code>	基于权重参数计算两个 tensor 之间的线性插值。	Ascend GPU CPU
<code>mindspore.ops.log</code>	逐元素计算输入 tensor 的自然对数。	Ascend GPU CPU
<code>mindspore.ops.log2</code>	逐元素计算输入 tensor 以 2 为底的对数。	Ascend GPU CPU
<code>mindspore.ops.log10</code>	逐元素计算 tensor 以 10 为底的对数。	Ascend GPU CPU
<code>mindspore.ops.log1p</code>	逐元素计算 $(\text{tensor} + 1)$ 的自然对数。	Ascend GPU CPU
<code>mindspore.ops.logaddexp</code>	计算输入的指数和的对数。	Ascend GPU CPU
<code>mindspore.ops.logaddexp2</code>	计算以 2 为底的输入的指数和的对数。	Ascend GPU CPU
<code>mindspore.ops.logical_and</code>	逐元素计算两个 tensor 的逻辑与运算。	Ascend GPU CPU
<code>mindspore.ops.logical_not</code>	逐元素计算输入 tensor 的逻辑非运算。	Ascend GPU CPU
<code>mindspore.ops.logical_or</code>	逐元素计算两个 tensor 的逻辑或运算。	Ascend GPU CPU

下页继续

表 6 - 续上页

<code>mindspore.ops.logical_xor</code>	逐元素计算两个 tensor 的逻辑异或运算。	Ascend CPU
<code>mindspore.ops.logit</code>	逐元素计算 tensor 的逻辑回归函数值。	Ascend GPU CPU
<code>mindspore.ops.mul</code>	两个 Tensor 逐元素相乘。	Ascend GPU CPU
<code>mindspore.ops.multiply</code>	<code>mindspore.ops.mul()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.mvlgamma</code>	逐元素计算 p 维多元对数伽马函数值。	Ascend GPU CPU
<code>mindspore.ops.neg</code>	计算输入 input 的相反数并返回。	Ascend GPU CPU
<code>mindspore.ops.negative</code>	<code>mindspore.ops.neg()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.nextafter</code>	逐元素计算 input 指向 other 的下一个可表示浮点值。	Ascend GPU CPU
<code>mindspore.ops.polar</code>	将极坐标转化为笛卡尔坐标。	Ascend GPU CPU
<code>mindspore.ops.polygamma</code>	计算 input 的多伽马函数的 n 阶导数。	GPU CPU
<code>mindspore.ops.positive</code>	返回输入 tensor 。	Ascend GPU CPU
<code>mindspore.ops.pow</code>	计算 input 中每个元素的 <i>exponent</i> 次幂。	Ascend GPU CPU
<code>mindspore.ops.rad2deg</code>	逐元素将 x 从弧度转换为度数。	Ascend GPU CPU
<code>mindspore.ops.ravel</code>	沿着 0 轴方向, 将多维 Tensor 展开成一维。	Ascend GPU CPU
<code>mindspore.ops.real</code>	返回输入 tensor 的实数部分。	Ascend GPU CPU
<code>mindspore.ops.reciprocal</code>	返回输入 tensor 每个元素的倒数。	Ascend GPU CPU
<code>mindspore.ops.remainder</code>	逐元素计算输入 tensor 的除法余数。	Ascend GPU CPU
<code>mindspore.ops.rot90</code>	沿指定维度的平面内将 n-D tensor 旋转 90 度。	Ascend GPU CPU
<code>mindspore.ops.round</code>	对输入数据四舍五入到最接近的整数。	Ascend GPU CPU
<code>mindspore.ops.rsqrt</code>	逐元素计算输入 tensor 的平方根倒数。	Ascend GPU CPU
<code>mindspore.ops.sgn</code>	<code>mindspore.ops.sign()</code> 在复数上的扩展。	Ascend GPU CPU
<code>mindspore.ops.sign</code>	按 sign 公式逐元素计算输入 tensor 。	Ascend GPU CPU
<code>mindspore.ops.signbit</code>	判断每个元素的符号, 如果元素值小于 0 则对应输出的位置为 True , 否则为 False 。	Ascend GPU CPU
<code>mindspore.ops.sin</code>	逐元素计算输入 tensor 的正弦。	Ascend GPU CPU
<code>mindspore.ops.sinc</code>	计算输入的归一化正弦值。	Ascend GPU CPU
<code>mindspore.ops.sinh</code>	逐元素计算输入 tensor 的双曲正弦。	Ascend GPU CPU
<code>mindspore.ops.sqrt</code>	逐元素计算 tensor 的平方根。	Ascend GPU CPU
<code>mindspore.ops.square</code>	逐元素计算 tensor 的平方。	Ascend GPU CPU
<code>mindspore.ops.sub</code>	逐元素计算第一个输入减去第二个输入的值。	Ascend GPU CPU
<code>mindspore.ops.subtract</code>	从 input 中减去经 <i>alpha</i> 缩放的 other 。	Ascend GPU CPU
<code>mindspore.ops.t</code>	转置二维 tensor , 维度小于二的 tensor 按原样返回。	Ascend GPU CPU
<code>mindspore.ops.tan</code>	逐元素计算输入 tensor 的正切值。	Ascend GPU CPU
<code>mindspore.ops.tanhshrink</code>	逐元素计算输入 tensor 的 Tanhshrink 激活函数值。	Ascend GPU CPU
<code>mindspore.ops.trapz</code>	计算沿指定维度的梯形法则积分。	Ascend GPU CPU
<code>mindspore.ops.tril_indices</code>	返回一个二维的 tensor , 包含 <i>row * col</i> 矩阵的下三角元素的索引。	Ascend GPU CPU
<code>mindspore.ops.triu_indices</code>	返回一个二维的 tensor , 包含 <i>row * col</i> 矩阵的上三角元素的索引。	Ascend GPU CPU
<code>mindspore.ops.true_divide</code>	<code>mindspore.ops.div()</code> 在 <code>rounding_mode=None</code> 时的别名。	Ascend GPU CPU
<code>mindspore.ops.trunc</code>	返回一个 tensor , 包含输入元素的截断整数值。	Ascend GPU CPU
<code>mindspore.ops.truncate_div</code>	将 x 和 y 逐元素相除, 结果将向 0 取整。	Ascend GPU CPU
<code>mindspore.ops.truncate_mod</code>	将 x 和 y 逐元素取模。	Ascend GPU CPU
<code>mindspore.ops.xdivy</code>	x 和 y 逐元素相除。	Ascend GPU CPU
<code>mindspore.ops.xlogy</code>	逐元素对 other 取对数, 再与 input 相乘。	Ascend GPU CPU
<code>mindspore.ops.zeta</code>	逐元素计算 Hurwitz zeta 的值。	Ascend GPU CPU

mindspore.ops.abs

`mindspore.ops.abs(input)`

逐元素计算输入 `tensor` 的绝对值。

$$out_i = |input_i|$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.abs(mindspore.tensor([-1.0, 1.0, 0.0]))
>>> print(output)
[1. 1. 0.]
```

mindspore.ops.absolute

`mindspore.ops.absolute(input)`

`mindspore.ops.abs()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.accumulate_n

`mindspore.ops.accumulate_n(x)`

逐元素计算列表中各个 `tensor` 的和。

`mindspore.ops.accumulate_n()` 与 `mindspore.ops.addn()` 类似, 但 `accumulate_n` 不会等待其所有输入就绪后再求和, 可节省内存。

参数:

- **x** (Union(tuple[`Tensor`], list[`Tensor`])) - 输入 `tensors`。

返回:

Tensor

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> y = mindspore.tensor([4, 5, 6], mindspore.float32)
>>> output = mindspore.ops.accumulate_n([x, y, x, y])
>>> print(output)
[10. 14. 18.]
```

mindspore.ops.acos

`mindspore.ops.acos(input)`

逐元素计算输入 `tensor` 的反余弦。

$$out_i = \cos^{-1}(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.acos(mindspore.tensor([0.74, 0.04, 0.30, 0.56]))
>>> print(output)
[0.737726 1.5307857 1.2661036 0.9764105]
```

mindspore.ops.arccos

`mindspore.ops.arccos(input)`

`mindspore.ops.acos()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.acosh

`mindspore.ops.acosh(input)`

逐元素计算输入 `tensor` 的反双曲余弦。

$$out_i = \cosh^{-1}(input_i)$$

说明: 给定一个输入 `tensor input`, 该函数计算每个元素的反双曲余弦。输入范围为 `[1, inf]`。

参数:

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> output = mindspore.ops.acosh(mindspore.tensor([1.0, 1.5, 3.0, 100.0]))
>>> print(output)
[0.          0.9624237  1.7627472  5.298292 ]
```

mindspore.ops.add

`mindspore.ops.add(input, other)`

逐元素计算两个输入 tensor 的和。

$$out_i = input_i + other_i$$

说明：

- 两个输入不能同时为 bool 类型。[True, Tensor(True, bool_), Tensor(np.array([True]), bool_)] 等都为 bool 类型。
 - 支持广播，支持隐式类型转换、类型提升。
 - 当输入为 tensor 时，维度应大于等于 1。
-

参数：

- **input** (Union[Tensor, number.Number, bool]) - 第一个输入 tensor。
- **other** (Union[Tensor, number.Number, bool]) - 第二个输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: x and y are both tensor.
>>> x = mindspore.tensor([1., 2., 3.])
>>> y = mindspore.tensor([4., 5., 6.])
>>> output = mindspore.ops.add(x, y)
>>> print(output)
[5. 7. 9.]
>>> # case 2: x is a scalar and y is a tensor
>>> x = mindspore.tensor(1, mindspore.int32)
```

(下页继续)

(续上页)

```
>>> y = mindspore.tensor([4., 5., 6.])
>>> output = mindspore.ops.add(x, y)
>>> print(output)
[5. 6. 7.]
>>> # the data type of x is int32, the data type of y is float32,
>>> # and the output is the data format of higher precision float32.
>>> print(output.dtype)
Float32
```

mindspore.ops.addcdiv

`mindspore.ops.addcdiv(input, tensor1, tensor2, value=1)`

tensor2 对 *tensor1* 逐元素相除，将结果乘以标量值 *value*，并加到 *input*。

$$y[i] = input[i] + value[i] * (tensor1[i] / tensor2[i])$$

参数：

- **input** (Tensor) - 输入 tensor。
- **tensor1** (Tensor) - 被除数（分子）。
- **tensor2** (Tensor) - 除数（分母）。
- **value** (Union[Tensor, number]) - (*tensor1* / *tensor2*) 的乘数。默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([1, 1, 1, 1], mindspore.float32)
>>> x1 = mindspore.tensor([1, 2, 3, 4], mindspore.float32)
>>> x2 = mindspore.tensor([4, 4, 2, 1], mindspore.float32)
>>> y = mindspore.ops.addcdiv(x, x1, x2, 0.1)
>>> print(y)
[1.025 1.05 1.15 1.4 ]
```

mindspore.ops.addcmul

`mindspore.ops.addcmul(input, tensor1, tensor2, value=1)`

tensor1 与 *tensor2* 的逐元素相乘，将结果乘以标量值 *value*，并加到 *input*。

$$output[i] = input[i] + value[i] * (tensor1[i] * tensor2[i])$$

参数：

- **input** (Tensor) - 输入 tensor。
- **tensor1** (Tensor) - 将被相乘的 *tensor1*。
- **tensor2** (Tensor) - 将被相乘的 *tensor2*。

- **value** (Union[Tensor, number]) - $(\text{tensor1} * \text{tensor2})$ 的乘数。默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 1, 1]], mindspore.float32)
>>> x1 = mindspore.tensor([[1], [2], [3]], mindspore.float32)
>>> x2 = mindspore.tensor([[1, 2, 3]], mindspore.float32)
>>> value = mindspore.tensor([1], mindspore.float32)
>>> y = mindspore.ops.addcmul(x, x1, x2, value)
>>> print(y)
[[ 2.  3.  4.]
 [ 3.  5.  7.]
 [ 4.  7. 10.]]
```

mindspore.ops.addmv

`mindspore.ops.addmv(input, mat, vec, *, beta=1, alpha=1)`

将矩阵 *mat* 和向量 *vec* 相乘，结果与输入 *input* 相加。

说明：

- 如果 *mat* 是一个大小为 (N, M) 的 tensor，且 *vec* 是一个大小为 M 的一维 tensor，那么 *input* 必须是可广播的，并且是一个大小为 N 的一维 tensor。
- 若 *beta* 为 0，那么 *input* 将被忽略。

$$\text{output} = \text{input} + (\text{mat} @ \text{vec})$$

参数：

- **input** (Tensor) - 输入 tensor。
- **mat** (Tensor) - 将被乘的矩阵 tensor。
- **vec** (Tensor) - 将被乘的向量 tensor。

关键字参数：

- **beta** (scalar[int, float, bool], 可选) - *input* 的尺度因子，默认 1。
- **alpha** (scalar[int, float, bool], 可选) - $(\text{mat} @ \text{vec})$ 的尺度因子，默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：


```
>>> import mindspore
>>> input = mindspore.tensor([2., 3.])
>>> mat = mindspore.tensor([[2., 5., 3.], [4., 2., 2.]])
>>> vec = mindspore.tensor([3., 2., 4.])
>>> output = mindspore.ops.addmv(input, mat, vec)
>>> print(output)
[30. 27.]
```

mindspore.ops.addn

mindspore.ops.addn(x)

逐元素计算列表中各个 **tensor** 的和。

参数:

- **x** (Union(tuple[**Tensor**], list[**Tensor**])) - 输入 **tensors**。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1, 2, 3])
>>> y = mindspore.tensor([4, 5, 6])
>>> mindspore.ops.addn([x, y, x, y])
Tensor(shape=[3], dtype=Int64, value= [10, 14, 18])
```

mindspore.ops.angle

mindspore.ops.angle(input)

逐元素计算输入 **tensor** 的辐角。

$$output_i = angle(input_i)$$

参数:

- **input** (**Tensor**) - 输入 **tensor**。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1 + 1j, -2 + 2j, 3 - 3j], mindspore.complex64)
>>> output = mindspore.ops.angle(input)*180/3.14159
>>> print(output)
[135.0001 135.0001 -45.00004]
```

mindspore.ops.arccosh

`mindspore.ops.arccosh(input)`
`mindspore.ops.acosh()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.arcsin

`mindspore.ops.arcsin(x)`
`mindspore.ops.asin()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.arcsinh

`mindspore.ops.arcsinh(input)`
`mindspore.ops.asinh()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.arctan

`mindspore.ops.arctan(input)`
`mindspore.ops.atan()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.arctanh

`mindspore.ops.arctanh(input)`
`mindspore.ops.atanh()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.arctan2

`mindspore.ops.arctan2(input, other)`
`mindspore.ops.atan2()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.asin

`mindspore.ops.asin(input)`

逐元素计算输入 `tensor` 的反正弦。

$$out_i = \sin^{-1}(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.asin(mindspore.tensor([0.74, 0.04, 0.30, 0.56]))
>>> print(output)
[0.8330704  0.04001067  0.30469266  0.5943858 ]
```

mindspore.ops.asinh

`mindspore.ops.asinh(input)`

逐元素计算输入 `tensor` 的反双曲正弦。

$$out_i = \sinh^{-1}(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.asinh(mindspore.tensor([-5.0, 1.5, 3.0, 100.0]))
>>> print(output)
[-2.3124382  1.1947632  1.8184465  5.298342 ]
```

mindspore.ops.atan

`mindspore.ops.atan(input)`

逐元素计算输入 `tensor` 的反正切值。

$$out_i = \tan^{-1}(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.atan(mindspore.tensor([1.0, 0.0]))
>>> print(output)
[0.7853982 0.          ]
```

mindspore.ops.atan2

`mindspore.ops.atan2(input, other)`

逐元素计算 `input/other` 的反正切值。

返回 $\theta \in [-\pi, \pi]$, 使得 $input = r * \sin(\theta)$, $other = r * \cos(\theta)$, 其中 $r = \sqrt{input^2 + other^2}$ 。

参数:

- **input** (Tensor, Number.number) - 输入 `tensor` 或常数。
- **other** (Tensor, Number.number) - 输入 `tensor` 或常数。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.atan2(mindspore.tensor([0., 1.]), mindspore.tensor([1., 1.]))
>>> print(output)
[0.          0.7853982]
```

mindspore.ops.atanh

`mindspore.ops.atanh(input)`

逐元素计算输入 `tensor` 的反双曲正切值。

$$out_i = \tanh^{-1}(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.atanh(mindspore.tensor([0, -0.5]))
>>> print(output)
[ 0.          -0.54930615]
```

mindspore.ops.atleast_1d

`mindspore.ops.atleast_1d(inputs)`

当输入中的 `tensor` 为零维时, 返回它的一维 `tensor`。其他维度的 `tensor`, 直接返回原 `tensor`。

参数:

- **inputs** (Union[Tensor, list[Tensor]]) - 输入 `tensor` 或 `tensor` 列表。

返回:

`tensor` 或 `tensor` 列表。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: Input is a zero-dimensional tensor.
>>> x = mindspore.tensor(1)
>>> mindspore.ops.atleast_1d(x)
Tensor(shape=[1], dtype=Int64, value= [1])
>>>
>>> # case 2: Input is a one-dimensional tensor.
>>> y = mindspore.tensor([0, 1])
>>> mindspore.ops.atleast_1d(y)
Tensor(shape=[2], dtype=Int64, value= [0, 1])
>>>
>>> # case 3: Input is a list containing tensors of various dimensions.
>>> mindspore.ops.atleast_1d([x, y])
(Tensor(shape=[1], dtype=Int64, value= [1]), Tensor(shape=[2], dtype=Int64, value= [0, 1]))
```

mindspore.ops.atleast_2d

`mindspore.ops.atleast_2d(inputs)`

当输入中的 `tensor` 小于二维时, 返回它的二维 `tensor`。其他维度的 `tensor`, 直接返回原 `tensor`。

参数:

- **inputs** (`Union[Tensor, list[Tensor]]`) - 输入 `tensor` 或 `tensor` 列表。

返回:

`tensor` 或 `tensor` 列表。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: Input is a zero-dimensional tensor.
>>> x = mindspore.tensor(1)
>>> mindspore.ops.atleast_2d(x)
Tensor(shape=[1, 1], dtype=Int64, value= [[1]])
>>>
>>> # case 2: Input is a 2-dimensional tensor.
>>> y = mindspore.tensor([[0, 1], [2, 3]])
>>> mindspore.ops.atleast_2d(y)
Tensor(shape=[2, 2], dtype=Int64, value=
[[0, 1],
 [2, 3]])
>>>
>>> # case 3: Input is a list containing tensors of various dimensions.
>>> mindspore.ops.atleast_2d([x, y])
(Tensor(shape=[1, 1], dtype=Int64, value=
[[1]]),
 Tensor(shape=[2, 2], dtype=Int64, value=
[[0, 1],
 [2, 3]]))
```

mindspore.ops.atleast_3d

`mindspore.ops.atleast_3d(inputs)`

当输入中的 `tensor` 小于三维时, 返回它的三维 `tensor`。其他维度的 `tensor`, 直接返回原 `tensor`。

参数:

- **inputs** (`Union[Tensor, list[Tensor]]`) - 输入 `tensor` 或 `tensor` 列表。

返回:

`tensor` 或 `tensor` 列表。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> # case 1: Input is a zero-dimensional tensor.
>>> x = mindspore.tensor(1)
>>> mindspore.ops.atleast_3d(x)
Tensor(shape=[1, 1, 1], dtype=Int64, value= [[[1]]])
>>>
>>> # case 2: Input is a 3-dimensional tensor.
>>> y = mindspore.tensor([[[0, 1], [2, 3]]])
>>> mindspore.ops.atleast_3d(y)
Tensor(shape=[1, 2, 2], dtype=Int64, value=
[[[0, 1],
  [2, 3]]])
>>>
>>> # case 3: Input is a list containing tensors of various dimensions.
>>> mindspore.ops.atleast_3d([x, y])
(Tensor(shape=[1, 1, 1], dtype=Int64, value=
[[[1]]]),
 Tensor(shape=[1, 2, 2], dtype=Int64, value=
[[[0, 1],
  [2, 3]]])

```

mindspore.ops.bessel_i0

mindspore.ops.bessel_i0(x)

逐元素计算输入 tensor 的第一类零阶修正贝塞尔函数值。

$$I_0(x) = J_0(ix) = \sum_{m=0}^{\infty} \frac{x^{2m}}{2^{2m}(m!)^2}$$

参数:

- **x** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

GPU CPU

样例:

```

>>> import mindspore
>>> x = mindspore.tensor([-1., -0.5, 0.5, 1.])
>>> output = mindspore.ops.bessel_i0(x)
>>> print(output)
[1.266066  1.0634835 1.0634835 1.266066]

```

mindspore.ops.bessel_i0e

`mindspore.ops.bessel_i0e(x)`

逐元素计算输入 `tensor` 的指数缩放第一类零阶修正贝塞尔函数值。

$$I_0 e(x) = e^{(-|x|)} * I_0(x) = e^{(-|x|)} * \sum_{m=0}^{\infty} \frac{x^{2m}}{2^{2m}(m!)^2}$$

参数:

- **x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([-1., -0.5, 0.5, 1.])
>>> output = mindspore.ops.bessel_i0e(x)
>>> print(output)
[0.46575961  0.64503527  0.64503527  0.46575961]
```

mindspore.ops.bessel_i1

`mindspore.ops.bessel_i1(x)`

逐元素计算输入 `tensor` 的第一类一阶修正贝塞尔函数值。

$$I_1(x) = i^{-1} J_1(ix) = \sum_{m=0}^{\infty} \frac{x^{2m+1}}{2^{2m+1} m!(m+1)!}$$

参数:

- **x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([-1., -0.5, 0.5, 1.])
>>> output = mindspore.ops.bessel_i1(x)
>>> print(output)
[-0.5651591  -0.25789431  0.25789431  0.5651591]
```


mindspore.ops.bessel_i1e

`mindspore.ops.bessel_i1e(x)`

逐元素计算输入 `tensor` 的指数缩放第一类一阶修正贝塞尔函数值。

$$I_1 e(x) = e^{(-|x|)} * I_1(x) = e^{(-|x|)} * \sum_{m=0}^{\infty} \frac{x^{2m+1}}{2^{2m+1} m!(m+1)!}$$

参数:

- `x` (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([-1., -0.5, 0.5, 1.])
>>> output = mindspore.ops.bessel_i1e(x)
>>> print(output)
[-0.20791042 -0.15642083  0.15642083  0.20791042]
```

mindspore.ops.bessel_j0

`mindspore.ops.bessel_j0(x)`

逐元素计算输入 `tensor` 的第一类零阶贝塞尔函数值。

$$J_0(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin \theta) d\theta = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{2^{2m} (m!)^2}$$

参数:

- `x` (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_j0(x)
>>> print(output)
[0.93846981  0.76519769  0.22389078 -0.39714981]
```

mindspore.ops.bessel_j1

mindspore.ops.bessel_j1(x)

逐元素计算输入 tensor 的第一类一阶贝塞尔函数值。

$$J_1(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \theta - \theta) d\theta = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m+1}}{2^{2m+1} m! (m+1)!}$$

参数:

- **x** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_j1(x)
>>> print(output)
[0.24226846  0.44005059  0.57672481 -0.06604333]
```

mindspore.ops.bessel_k0

mindspore.ops.bessel_k0(x)

逐元素计算输入 tensor 的第二类零阶修正贝塞尔函数值。

$$K_0(x) = \lim_{\nu \rightarrow 0} \left(\frac{\pi}{2} \right) \frac{I_{-\nu}(x) - I_{\nu}(x)}{\sin(\nu\pi)} = \int_0^\infty e^{-x \cosh t} dt$$

参数:

- **x** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_k0(x)
>>> print(output)
[0.92441907  0.42102444  0.11389387  0.01115968]
```

mindspore.ops.bessel_k0e

`mindspore.ops.bessel_k0e(x)`

逐元素计算输入 `tensor` 的指数缩放第二类零阶修正贝塞尔函数值。

$$K_0 e(x) = e^{(-|x|)} * K_0(x) = e^{(-|x|)} * \int_0^\infty e^{-x \cosh t} dt$$

参数:

- `x` (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_k0e(x)
>>> print(output)
[1.52410939  1.14446308  0.84156822  0.60929767]
```

mindspore.ops.bessel_k1

`mindspore.ops.bessel_k1(x)`

逐元素计算输入 `tensor` 的第二类一阶修正贝塞尔函数值。

计算公式定义如下:

$$K_1(x) = \lim_{\nu \rightarrow 1} \left(\frac{\pi}{2} \right) \frac{I_{-\nu}(x) - I_{\nu}(x)}{\sin(\nu\pi)} = \int_0^\infty e^{-x \cosh t} \cosh(t) dt$$

参数:

- `x` (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_k1(x)
>>> print(output)
[1.65644112  0.60190723  0.13986588  0.0124835]
```

mindspore.ops.bessel_k1e

mindspore.ops.bessel_k1e(x)

逐元素计算输入 tensor 的指数缩放第二类一阶修正贝塞尔函数值。

$$K_1 e(x) = e^{(-|x|)} * K_1(x) = e^{(-|x|)} * \int_0^{\infty} e^{-x \cosh t} \cosh(t) dt$$

参数:

- **x** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_k1e(x)
>>> print(output)
[2.73100971  1.63615349  1.03347685  0.68157595]
```

mindspore.ops.bessel_y0

mindspore.ops.bessel_y0(x)

逐元素计算输入 tensor 的第二类零阶贝塞尔函数值。

$$Y_0(x) = \lim_{n \rightarrow 0} \frac{J_n(x) \cos n\pi - J_{-n}(x)}{\sin n\pi}$$

参数:

- **x** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_y0(x)
>>> print(output)
[-0.44451874  0.08825696  0.51037567 -0.01694074]
```

mindspore.ops.bessel_y1

`mindspore.ops.bessel_y1(x)`

逐元素计算输入 `tensor` 的第二类一阶贝塞尔函数值。

$$Y_1(x) = \lim_{n \rightarrow 1} \frac{J_n(x) \cos n\pi - J_{-n}(x)}{\sin n\pi}$$

参数:

- **x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([0.5, 1., 2., 4.])
>>> output = mindspore.ops.bessel_y1(x)
>>> print(output)
[-1.47147239 -0.78121282 -0.10703243  0.39792571]
```

mindspore.ops.bitwise_and

`mindspore.ops.bitwise_and(input, other)`

逐元素对两个输入 `tensor` 做与运算。如果 `input` 和 `other` 数据类型不同，遵循隐式类型转换规则、类型提升规则。

参数:

- **input** (Tensor) - 第一个输入 `tensor`。
- **other** (Tensor) - 第二个输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1, -2, 3])
>>> other = mindspore.tensor([1, 0, 3])
>>> mindspore.ops.bitwise_and(input, other)
Tensor(shape=[3], dtype=Int64, value= [1, 0, 3])
>>> # Same as calling via the | operator:
>>> input & other
Tensor(shape=[3], dtype=Int64, value= [1, 0, 3])
```

mindspore.ops.bitwise_left_shift

`mindspore.ops.bitwise_left_shift(input, other)`

逐元素对输入 *input* 进行左移位运算, 移动的位数由 *other* 指定。

$$out_i = input_i \ll other_i$$

参数:

- **input** (Union[Tensor, int, bool]) - 被左移的输入 tensor。
- **other** (Union[Tensor, int, bool]) - 左移的位数。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 4, 8])
>>> mindspore.ops.bitwise_left_shift(input, 1)
Tensor(shape=[4], dtype=Int64, value= [ 2,  4,  8, 16])
```

mindspore.ops.bitwise_or

`mindspore.ops.bitwise_or(input, other)`

逐元素对两个输入 tensor 做或运算。如果 *input* 和 *other* 数据类型不同, 遵循隐式类型转换规则、类型提升规则。

参数:

- **input** (Tensor) - 第一个输入 tensor。
- **other** (Tensor) - 第二个输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1, -2, 3])
>>> other = mindspore.tensor([1, 0, 3])
>>> mindspore.ops.bitwise_or(input, other)
Tensor(shape=[3], dtype=Int64, value= [-1, -2,  3])
>>> # Same as calling via the | operator:
>>> input | other
Tensor(shape=[3], dtype=Int64, value= [-1, -2,  3])
```

`mindspore.ops.bitwise_right_shift`

`mindspore.ops.bitwise_right_shift(input, other)`

逐元素对输入 *input* 进行右移位运算, 移动的位数由 *other* 指定。

$$out_i = input_i >> other_i$$

参数:

- **input** (Union[Tensor, int, bool]) - 被右移的输入 tensor。
- **other** (Union[Tensor, int, bool]) - 右移的位数。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 4, 8])
>>> mindspore.ops.bitwise_right_shift(input, 1)
Tensor(shape=[4], dtype=Int64, value= [0, 1, 2, 4])
```

`mindspore.ops.bitwise_xor`

`mindspore.ops.bitwise_xor(input, other)`

逐元素对两个输入 tensor 做异或运算。如果 *input* 和 *other* 数据类型不同, 遵循隐式类型转换规则、类型提升规则。

参数:

- **input** (Tensor) - 第一个输入 tensor。
- **other** (Tensor) - 第二个输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1, -2, 3])
>>> other = mindspore.tensor([1, 0, 3])
>>> mindspore.ops.bitwise_xor(input, other)
Tensor(shape=[3], dtype=Int64, value= [-2, -2,  0])
>>> # Same as calling via the ^ operator:
>>> input ^ other
Tensor(shape=[3], dtype=Int64, value= [-2, -2,  0])
```

mindspore.ops.ceil

`mindspore.ops.ceil(input)`

逐元素对输入 `tensor` 做向上取整。

$$out_i = \lceil input_i \rceil$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1.1, 2.5, -1.5])
>>> output = mindspore.ops.ceil(input)
>>> print(output)
[ 2.  3. -1.]
```

mindspore.ops.clamp

`mindspore.ops.clamp(input, min=None, max=None)`

将输入 `tensor` 的所有元素限制在范围 `[min, max]` 内。

$$out_i = \begin{cases} max & \text{if } input_i \geq max \\ input_i & \text{if } min < input_i < max \\ min & \text{if } input_i \leq min \end{cases}$$

说明:

- `min` 和 `max` 不能同时为 `None`;
 - 当 `min` 为 `None` 时, 无最小值限制
 - 当 `max` 为 `None` 时, 无最大值限制.
 - 当 `min` 大于 `max` 时, Tensor 中所有元素的值会被置为 `max`;
-

参数:

- **input** (Tensor) - 输入 `tensor`。
- **min** (Union(Tensor, float, int), 可选) - 指定最小值。默认 `None`。
- **max** (Union(Tensor, float, int), 可选) - 指定最大值。默认 `None`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: `min` and `max` are integer
>>> input = mindspore.tensor([[1, 25, 5, 7], [4, 11, 6, 21]])
>>> mindspore.ops.clamp(input, 5, 20)
Tensor(shape=[2, 4], dtype=Int64, value=
[[ 5, 20,  5,  7],
 [ 5, 11,  6, 20]])
>>>
>>> # case 2: If `min` and `max` are tensors, their shapes need to be broadcastable with
↳input.
>>> min = mindspore.tensor([2, 4, 6, 8])
>>> max = mindspore.tensor([10, 12, 14, 18])
>>> mindspore.ops.clamp(input, min, max)
Tensor(shape=[2, 4], dtype=Int64, value=
[[ 2, 12,  6,  8],
 [ 4, 11,  6, 18]])
```

mindspore.ops.clip

`mindspore.ops.clip(input, min=None, max=None)`
`mindspore.ops.clamp()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.combinations

`mindspore.ops.combinations(input, r=2, with_replacement=False)`

返回输入 `tensor` 中所有长度为 `r` 的子序列。

当 `with_replacement` 为 `False` 时, 功能与 Python 的 `itertools.combinations` 类似; 当为 `True` 时, 功能与 `itertools.combinations_with_replacement` 一致。

参数:

- **input** (Tensor) - 一维输入 `tensor`。
- **r** (int, 可选) - 进行组合的子序列长度。默认 2。
- **with_replacement** (bool, 可选) - 是否允许组合存在重复值。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> input = mindspore.tensor([1, 2, 3])
>>> mindspore.ops.combinations(input)
Tensor(shape=[3, 2], dtype=Int64, value=
[[1, 2],
 [1, 3],
 [2, 3]])
>>> mindspore.ops.combinations(input, r=3)
Tensor(shape=[1, 3], dtype=Int64, value=
[[1, 2, 3]])
>>> mindspore.ops.combinations(input, with_replacement=True)
Tensor(shape=[6, 2], dtype=Int64, value=
[[1, 1],
 [1, 2],
 [1, 3],
 [2, 2],
 [2, 3],
 [3, 3]])
>>> # It has the same results as using itertools.combinations.
>>> import itertools
>>> input = [1, 2, 3]
>>> list(itertools.combinations(input, r=2))
[(1, 2), (1, 3), (2, 3)]
>>> list(itertools.combinations(input, r=3))
[(1, 2, 3)]
>>> list(itertools.combinations_with_replacement(input, r=2))
[(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)]

```

mindspore.ops.copysign

`mindspore.ops.copysign(x, other)`

创建一个 float tensor, 由 x 的绝对值和 $other$ 的符号组成。

支持广播。

参数:

- **x** (Union[Tensor]) - 输入 tensor
- **other** (Union[int, float, Tensor]) - 决定返回值符号的 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> # case 1: When other is a tensor
>>> x = mindspore.tensor([[0.3, -0.7],
...                       [0.5, 0.5]])
>>> other = mindspore.tensor([-0.4, 0.6])
>>> output = mindspore.ops.copysign(x, other)
>>> print(output)
[[-0.3  0.7]

```

(下页继续)

(续上页)

```

[-0.5  0.5]]
>>> # case 2: When other is a scalar
>>> output = mindspore.ops.copysign(x, 2)
>>> print(output)
[[0.3 0.7]
 [0.5 0.5]]

```

mindspore.ops.cos

`mindspore.ops.cos(input)`

逐元素计算输入 tensor 的余弦。

$$out_i = \cos(x_i)$$

警告：如果使用 float64，可能会存在精度丢失的问题。

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> # The datatype of output will be float32 when datatype of input is in [bool, int8, uint8,
↪ int16, int32, int64] (Datatype only supported on Ascend).
>>> input = mindspore.tensor([0, 1, 2], mindspore.int32)
>>> mindspore.ops.cos(input)
Tensor(shape=[3], dtype=Float32, value= [ 1.00000000e+00,  5.40302306e-01, -4.16146837e-01])
>>>
>>> # Otherwise output has the same dtype as the `input`.
>>> input = mindspore.tensor([0.74, 0.04, 0.30, 0.56], mindspore.float64)
>>> mindspore.ops.cos(input)
Tensor(shape=[4], dtype=Float64, value= [ 7.38468559e-01,  9.99200107e-01,  9.55336489e-01,
↪ 8.47255111e-01])

```

mindspore.ops.cosh

`mindspore.ops.cosh(input)`

逐元素计算输入 `tensor` 的双曲余弦。

$$out_i = \cosh(input_i)$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([0.74, 0.04, 0.30, 0.56])
>>> output = mindspore.ops.cosh(input)
>>> print(output)
[1.2865248 1.0008001 1.0453385 1.1609408]
```

mindspore.ops.cosine_similarity

`mindspore.ops.cosine_similarity(x1, x2, dim=1, eps=1e-08)`

沿指定维度计算两个输入 `tensor` 之间的余弦相似度。

说明：当前暂不支持对输入进行广播。

$$\text{similarity} = \frac{x_1 \cdot x_2}{\max(\|x_1\|_2 \cdot \|x_2\|_2, \epsilon)}$$

参数：

- **x1** (Tensor) - 第一个输入 `tensor`。
- **x2** (Tensor) - 第二个输入 `tensor`。
- **dim** (int, 可选) - 指定维度。默认 1。
- **eps** (float, 可选) - 极小值，用于避免除零的情况。默认 1e-08。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[[-0.0256, 0.0127, -0.2475, 0.2316, 0.8037],
...                           [0.5809, -1.2712, -0.7038, -0.2558, 0.7494]]])
>>> other = mindspore.tensor([[[-0.6115, -0.1965, -0.8484, 0.2389, 0.2409],
...                           [1.8940, -2.1997, 0.1915, 0.0856, 0.7542]]])
>>> output = mindspore.ops.cosine_similarity(input, other)
>>> print(output)
[0.4843164 0.81647635]
```

mindspore.ops.cov

`mindspore.ops.cov(input, *, correction=1, fweights=None, aweights=None)`

返回输入 `tensor` 的协方差矩阵，其中输入 `tensor` 的行表示变量，列表示观测值。协方差矩阵的对角线为输入 `tensor` 每个变量的方差，非对角线上的元素为两两变量之间的协方差。当输入为零维或一维 `tensor` 时，返回其方差。

变量 a 和 b 的无偏样本协方差由下式给出：

$$\text{cov}_w(a, b) = \frac{\sum_{i=1}^N (a_i - \bar{a})(b_i - \bar{b})}{N - 1}$$

其中 $\bar{a}$ 和 $\bar{b}$ 分别是 a 和 b 的简单均值。

如果提供了 `fweights` 和/或 `aweights`，则计算无偏加权协方差，由下式给出：

$$\text{cov}_w(a, b) = \frac{\sum_{i=1}^N w_i (a_i - \mu_a^*)(b_i - \mu_b^*)}{\sum_{i=1}^N w_i - 1}$$

其中 w 基于提供的 `fweights` 或 `aweights` 中的任意一个参数进行表示，如果两个参数都有提供，则 $w = fweights \times aweights$ ，并且 $\mu_x^* = \frac{\sum_{i=1}^N w_i x_i}{\sum_{i=1}^N w_i}$ 表示变量的加权平均值。

警告：`fweights` 和 `aweights` 的值不能为负数，负数权重场景结果未定义。

说明：当前暂不支持复数。

参数：

- **input** (Tensor) - 零维、一维或二维输入 `tensor`。

关键字参数：

- **correction** (int, 可选) - 样本大小与样本自由度之间的差值。`correction = 0` 将返回简单平均值。默认为 Bessel 校正 `correction = 1`，即使指定了 `fweights` 和 `aweights`，也会返回无偏估计。
- **fweights** (Tensor, 可选) - 标量或一维 `tensor`，表示每一个观测向量的重复次数（频率）。必须为整数类型。元素数必须等于输入 `input` 的列数。若为 `None` 则忽略。默认 `None`。
- **aweights** (Tensor, 可选) - 标量或一维 `tensor`，表示每一个观测向量的重要性（权重），重要性越高对应值越大。必须为浮点数类型。元素数必须等于输入 `input` 的列数。若为 `None` 则忽略。默认 `None`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[0., 5., 7.],
...                           [3., 5., 0.]])
>>> output = mindspore.ops.cov(input)
>>> print(output)
[[13.         -3.5         ]
 [-3.5         6.3333335]]
>>> output = mindspore.ops.cov(input, correction=0)
>>> print(output)
[[ 8.666667  -2.3333333]
 [-2.3333333  4.2222223]]
>>> fw = mindspore.tensor([5, 2, 4], dtype=mindspore.int64)
>>> aw = mindspore.tensor([0.4588, 0.9083, 0.7616], mindspore.float32)
>>> output = mindspore.ops.cov(input, fweights=fw, aweights=aw)
>>> print(output)
[[10.146146 -3.47241 ]
 [-3.47241  4.716825]]
```

mindspore.ops.diff

`mindspore.ops.diff(x, n=1, axis=-1, prepend=None, append=None)`

按指定轴计算输入 tensor 的 n 阶前向差分。

一阶差分的计算公式为: $out[i] = x[i+1] - x[i]$ 。高阶差分通过递归调用 `mindspore.ops.diff()` 实现。

说明: 不支持空 Tensor, 如果传入了空 Tensor, 会出现 `ValueError`。空 Tensor 指的是, Tensor 的任意一维为零。比如 shape 为 (0,) , (1,2,0,4) 的 Tensor 都为空 Tensor。

参数:

- **x** (Tensor) - 输入 tensor。
- **n** (int, 可选) - 计算差分的阶数, 目前只支持 1。默认 1。
- **axis** (int, 可选) - 指定轴。默认 -1。
- **prepend** (Tensor, 可选) - 在计算差分之前, 沿 *axis* 添加到 *x* 之前的值。它们的维度和必须与输入的维度相同, 除维度 *axis* 外, 其余维度的形状必须与输入张量一致。默认 None。
- **append** (Tensor, 可选) - 在计算差分之前, 沿 *axis* 添加到 *x* 之后的值。它们的维度必须与输入的维度相同, 除维度 *axis* 外, 其余维度的形状必须与输入张量一致。默认 None。

返回:

一个数据类型与 *x* 相同, shape 与 *x* 相比在第 *axis* 维上缩小 *n* 的 tensor。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> x = mindspore.tensor([1, 3, 2])
>>> # case 1: By default, compute the first-order differences along axis -1.
>>> mindspore.ops.diff(x)
Tensor(shape=[2], dtype=Int64, value= [ 2, -1])
>>>
>>> # case 2: When argument prepend is setting:
>>> n = mindspore.tensor([4, 5])
>>> mindspore.ops.diff(x, prepend=n)
Tensor(shape=[4], dtype=Int64, value= [ 1, -4,  2, -1])
>>>
>>> # case 3: When argument append is setting:
>>> mindspore.ops.diff(x, append=n)
Tensor(shape=[4], dtype=Int64, value= [ 2, -1,  2,  1])
>>>
>>> # case 4: When input is 2-D dimensional tensor, compute forward difference along_
different axis.
>>> x = mindspore.tensor([[1, 2, 3], [3, 4, 5]])
>>> mindspore.ops.diff(x, axis=0)
Tensor(shape=[1, 3], dtype=Int64, value=
[[2, 2, 2]])
>>> mindspore.ops.diff(x, axis=1)
Tensor(shape=[2, 2], dtype=Int64, value=
[[1, 1],
 [1, 1]])

```

mindspore.ops.deg2rad

`mindspore.ops.deg2rad(x)`

逐元素地将输入 `tensor` 从度数制转换为弧度制。

参数：

- **x** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> input = mindspore.tensor([[90.0, -90.0], [180.0, -180.0], [270.0, -270.0]])
>>> output = mindspore.ops.deg2rad(input)
>>> print(output)
[[ 1.5707964 -1.5707964]
 [ 3.1415927 -3.1415927]
 [ 4.712389  -4.712389 ]]

```

mindspore.ops.digamma

`mindspore.ops.digamma(input)`

逐元素计算输入 `tensor` 的伽玛函数的对数导数。

$$P(x) = \frac{d}{dx}(\ln(\Gamma(x)))$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1.5, 0.5, 9])
>>> output = mindspore.ops.digamma(input)
>>> print(output)
[ 0.03648992 -1.9635109  2.1406415 ]
```

mindspore.ops.div

`mindspore.ops.div(input, other, *, rounding_mode=None)`

逐元素计算第一个输入 `tensor` 除以第二个输入 `tensor`。

$$out_i = input_i / other_i$$

说明：

- 两个输入 `tensor` 的 `shape` 须可以进行广播。
 - 两个输入 `tensor` 不能同时为 `bool` 类型。
 - 两个输入 `tensor` 遵循隐式类型转换规则，使数据类型保持一致。
-

参数：

- **input** (Union[Tensor, Number, bool]) - 第一个输入 `tensor`。
- **other** (Union[Tensor, Number, bool]) - 第二个输入 `tensor`。

关键字参数：

- **rounding_mode** (str, 可选) - 应用于计算结果的舍入类型。默认 `None`。三种类型被定义为：
 - `None`：相当于 Python 中的 *true division* 或 NumPy 中的 *true_divide*。

- **"floor"**: 向下取整。相当于 Python 中的 *floor division* 或 NumPy 中的 *floor_divide*。
- **"trunc"**: 向 0 取整。相当于 C 语言风格的整数除法。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[ -0.3711, -1.9353, -0.4605, -0.2917],
...                           [ 0.1815, -1.0111, 0.9805, -1.5923],
...                           [ 0.1062, 1.4581, 0.7759, -1.2344],
...                           [-0.1830, -0.0313, 1.1908, -1.4757]])
>>> other = mindspore.tensor([ 0.8032, 0.2930, -0.8113, -0.2308])
>>> output = mindspore.ops.div(input, other)
>>> print(output)
[[-0.4620269 -6.605119 0.5676076 1.2638649 ]
 [ 0.22597112 -3.4508533 -1.2085541 6.899047 ]
 [ 0.13222112 4.97645 -0.95636636 5.348354 ]
 [-0.22783864 -0.10682594 -1.4677677 6.3938475 ]]
>>> output = mindspore.ops.div(input, other, rounding_mode='floor')
>>> print(output)
[[-1. -7.  0.  1.]
 [ 0. -4. -2.  6.]
 [ 0.  4. -1.  5.]
 [-1. -1. -2.  6.]]
```

mindspore.ops.divide

`mindspore.ops.divide(input, other, *, rounding_mode=None)`

`mindspore.ops.div()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.erf

`mindspore.ops.erf(input)`

逐元素计算输入 tensor 的高斯误差。

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # The datatype of output will be float32 when datatype of input is in [int64,
↳bool](Datatype only supported on Ascend).
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.int64)
>>> mindspore.ops.erf(input)
Tensor(shape=[5], dtype=Float32, value= [-8.42700793e-01,  0.00000000e+00,  8.42700793e-01,  ↳
↳9.95322265e-01,  9.99977910e-01])
>>>
>>> # Otherwise output has the same dtype as the input.
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.float64)
>>> mindspore.ops.erf(input)
Tensor(shape=[5], dtype=Float64, value= [-8.42700793e-01,  0.00000000e+00,  8.42700793e-01,  ↳
↳9.95322265e-01,  9.99977910e-01])
```

mindspore.ops.erfc

`mindspore.ops.erfc(input)`

逐元素计算输入 tensor 的互补误差。

$$\text{erfc}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # The datatype of output will be float32 when datatype of input is in [int64,
↳bool](Datatype only supported on Ascend).
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.int64)
>>> mindspore.ops.erfc(input)
Tensor(shape=[5], dtype=Float32, value= [ 1.84270079e+00,  1.00000000e+00,  1.57299207e-01,  ↳
↳4.67773498e-03,  2.20904970e-05])
>>>
>>> # Otherwise output has the same dtype as the input.
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.float64)
>>> mindspore.ops.erfc(input)
Tensor(shape=[5], dtype=Float64, value= [ 1.84270079e+00,  1.00000000e+00,  1.57299207e-01,  ↳
↳4.67773498e-03,  2.20904970e-05])
```

mindspore.ops.erfinv

`mindspore.ops.erfinv(input)`

逐元素计算输入 `tensor` 的逆误差。逆误差函数在 $(-1, 1)$ 范围内定义为:

$$\text{erfinv}(\text{erf}(x)) = x$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # When the `input` is int8, int16, int32, int64, uint8 or bool, the return value type is float32.
>>> input = mindspore.tensor([0, 0.5, -0.9], mindspore.int64)
>>> mindspore.ops.erfinv(input)
Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00])
>>> # Otherwise, the return value type is the same as the input type.
>>> input = mindspore.tensor([0, 0.5, -0.9], mindspore.float32)
>>> mindspore.ops.erfinv(input)
Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  4.76936132e-01, -1.16308689e+00])
```

mindspore.ops.exp

`mindspore.ops.exp(input)`

逐元素计算输入 `tensor` 的指数。

$$\text{out}_i = e^{x_i}$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.0, 1.0, 3.0], mindspore.float32)
>>> output = mindspore.ops.exp(input)
>>> print(output)
[ 1.          2.7182817 20.085537]
```

mindspore.ops.exp2

`mindspore.ops.exp2(input)`

逐元素计算输入 `tensor` 以 2 为底的指数。

$$out_i = 2^{input_i}$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.0, 1.0, 3.0], mindspore.float32)
>>> output = mindspore.ops.exp2(input)
>>> print(output)
[1. 2. 8.]
```

mindspore.ops.expm1

`mindspore.ops.expm1(input)`

逐元素计算输入 `tensor` 的指数，然后减 1。

$$out_i = e^{x_i} - 1$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.0, 1.0, 3.0], mindspore.float32)
>>> output = mindspore.ops.expm1(input)
>>> print(output)
[ 0.          1.7182817 19.085537 ]
```

mindspore.ops.floor

`mindspore.ops.floor(input)`

逐元素对输入 `tensor` 做向下取整。

$$out_i = \lfloor input_i \rfloor$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1.1, 2.5, -1.5])
>>> output = mindspore.ops.floor(input)
>>> print(output)
[ 1.  2. -2.]
```

mindspore.ops.floor_div

`mindspore.ops.floor_div(x, y)`

`mindspore.ops.floor_divide()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.floor_divide

`mindspore.ops.floor_divide(input, other)`

逐元素计算 `input` 除以 `other`，并向下取整。

如果 `input` 和 `other` 数据类型不同，遵循隐式类型转换规则。输入必须是两个 `tensor` 或一个 `tensor` 和一个 `scalar`。当输入是两个 `tensor` 时，其 `shape` 须可以进行广播，并且数据类型不能同时为 `bool`。

$$out_i = \text{floor}\left(\frac{input_i}{other_i}\right)$$

警告: 这是一个实验性 API，后续可能修改或删除。

参数:

- **input** (Union[Tensor, Number, bool]) - 第一个输入 `tensor`。
- **other** (Union[Tensor, Number, bool]) - 第二个输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: Two tensors with boolean and integer data type.
>>> input = mindspore.tensor([True, True, False])
>>> other = mindspore.tensor([1, 2, 4])
>>> output = mindspore.ops.floor_divide(input, other)
>>> print(output)
[1 0 0]
>>>
>>> # case 2: One tensor and one scalar.
>>> input = mindspore.tensor([1, 2, 4])
>>> other = mindspore.tensor(1.5)
>>> output = mindspore.ops.floor_divide(input, other)
>>> print(output)
[0. 1. 2.]
>>>
>>> # case 3: When inputs have different data types, type promotion rules are followed.
>>> input = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> other = mindspore.tensor([1.1, 2.5, -1.5], mindspore.float32)
>>> output = mindspore.ops.floor_divide(input, other)
>>> print(output)
[ 0.  0. -3.]
```

mindspore.ops.floor_mod

`mindspore.ops.floor_mod(x, y)`

逐元素计算第一个输入除以第二个输入，并向下取余。

如果两个输入数据类型不同，遵循隐式类型转换规则。输入必须是两个 `tensor` 或一个 `tensor` 和一个 `scalar`。当输入是两个 `tensor` 时，其 `shape` 须可以进行广播，并且数据类型不能同时为 `bool`。

$$out_i = \text{floor}(x_i // y_i)$$

警告:

- 输入 `y` 的元素不能等于 0，否则将返回当前数据类型的最大值。
- 当输入元素数量超过 2048 时，算子的精度不能保证千分之二的要求。
- 由于架构不同，该算子在 NPU 和 CPU 上的计算结果可能不一致。
- 如果 `shape` 表示为 $(D1, D2, \dots, Dn)$ ，那么 $D1 * D2 \cdots Dn \leq 1000000, n \leq 8$ 。

参数:

- `x` (`Union[Tensor, Number, bool]`) - 第一个输入 `tensor`。
- `y` (`Union[Tensor, Number, bool]`) - 第二个输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: Two tensors with boolean and integer data type.
>>> input = mindspore.tensor([True, True, False])
>>> other = mindspore.tensor([1, 2, 4])
>>> output = mindspore.ops.floor_mod(input, other)
>>> print(output)
[0 1 0]
>>>
>>> # case 2: One tensor and one scalar.
>>> input = mindspore.tensor([1, 2, 4])
>>> other = mindspore.tensor(1.5)
>>> output = mindspore.ops.floor_mod(input, other)
>>> print(output)
[1. 0.5 1. ]
>>>
>>> # case 3: When inputs have different data types, type promotion rules are followed.
>>> input = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> other = mindspore.tensor([1.1, 2.5, -1.5], mindspore.float32)
>>> output = mindspore.ops.floor_mod(input, other)
>>> print(output)
[ 1.  2. -0.5]
```

mindspore.ops.float_power

mindspore.ops.float_power (*input, exponent*)

逐元素计算以第一个输入为底，第二个输入为指数。如果输入为实数类型，则转换为 `mindspore.float64` 计算。

说明：目前不支持复数运算。

参数:

- **input** (`Union[Tensor, Number]`) - 第一个输入。
- **exponent** (`Union[Tensor, Number]`) - 第二个输入。如果 *input* 是 `Number`，则该参数必须是 `Tensor` 类型。

返回:

类型为 `mindspore.float64` 的 `tensor`。

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: When exponent is scalar:
>>> input = mindspore.tensor([-1.5, 0., 2.])
>>> output = mindspore.ops.float_power(input, 2)
```

(下页继续)

(续上页)

```
>>> print(output)
[2.25 0.  4.  ]
>>>
>>> # case 2: When exponent is a tensor:
>>> exponent = mindspore.tensor([0, 1, 2])
>>> output = mindspore.ops.float_power(input, exponent)
>>> print(output)
[1. 0. 4.]
```

mindspore.ops.fmax

`mindspore.ops.fmax(input, other)`
逐元素计算输入 tensor 的最大值。

$$output_i = \max(x1_i, x2_i)$$

说明:

- 支持隐式类型转换、类型提升。
- 输入 *input* 和 *other* 的 shape 必须能相互广播。
- 如果其中一个比较值是 NaN，则返回另一个比较值。

参数:

- **input** (Tensor) - 第一个输入。
- **other** (Tensor) - 第二个输入。

返回:

Tensor

支持平台:

CPU

样例:

```
>>> import mindspore
>>> x1 = mindspore.tensor([1.0, 5.0, 3.0], mindspore.float32)
>>> x2 = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float32)
>>> output = mindspore.ops.fmax(x1, x2)
>>> print(output)
[4. 5. 6.]
```


mindspore.ops.fmod

`mindspore.ops.fmod(input, other)`

逐元素计算第一个输入除以第二个输入的余数。

支持广播、类型提升。

$$out = input - n * other$$

其中 n 是 $input/other$ 结果中的整数部分。返回值的符号和 $input$ 相同，在数值上小于 $other$ 。

参数：

- **input** (Union[Tensor, Number]) - 被除数。
- **other** (Union[Tensor, Number]) - 除数。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: When other is scalar:
>>> input = mindspore.tensor([-3., -2, -1, 3, 4, 5])
>>> output = mindspore.ops.fmod(input, 2)
>>> print(output)
[-1.  0. -1.  1.  0.  1.]
>>>
>>> # case 2: When other is a tensor:
>>> other = mindspore.tensor([-1., 1, 2, 2.5, 1.5, 1.7])
>>> output = mindspore.ops.fmod(input, other)
>>> print(output)
[ 0.          0.         -1.          0.5         1.         1.5999999]
```

mindspore.ops.frac

`mindspore.ops.frac(x)`

返回输入 tensor 中每个元素的小数部分。

参数：

- **x** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([2, 4.2, -2.5])
>>> output = mindspore.ops.frac(input)
>>> print(output)
[ 0.          0.19999981 -0.5          ]
```

mindspore.ops.gcd

`mindspore.ops.gcd(input, other)`

逐元素计算两个输入 `tensor` 的最大公约数。

支持广播、类型提升。数据类型支持：int16（使用 Ascend 后端时支持，graph mode 只在 jit level 为 O0 时支持）、int32、int64 之一。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 第一个输入 `tensor`。
- **other** (Tensor) - 第二个输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([7, 8, 9])
>>> other = mindspore.tensor([14, 6, 12])
>>> mindspore.ops.gcd(input, other)
Tensor(shape=[3], dtype=Int64, value= [7, 2, 3])
```

mindspore.ops.hypot

`mindspore.ops.hypot(input, other)`

给定直角三角形的边，逐元素计算其斜边。

支持广播、类型提升。

$$out_i = \sqrt{input_i^2 + other_i^2}$$

参数：

- **input** (Tensor) - 第一个输入 `tesnor`。
- **other** (Tensor) - 第二个输入 `tesnor`。

返回：

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([3., 5., 7.])
>>> other = mindspore.tensor([4., 12., 24.])
>>> output = mindspore.ops.hypot(input, other)
>>> print(output)
[ 5. 13. 25.]
```

mindspore.ops.igamma

`mindspore.ops.igamma(input, other)`

计算正则化的下层不完全伽马函数。

如果我们将 `input` 比作 a ，`other` 比作 x ，则正则化的下层不完全伽马函数可以表示成：

$$P(a, x) = \text{Gamma}(a, x) / \text{Gamma}(a) = 1 - Q(a, x)$$

其中，

$$\text{Gamma}(a, x) = \int_0^x t^{a-1} \exp^{-t} dt$$

为下层不完全伽马函数。

$Q(a, x)$ 则为正则化的上层完全伽马函数。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 第一个输入 tensor。
- **other** (Tensor) - 第二个输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([2.0, 4.0, 6.0, 8.0])
>>> other = mindspore.tensor([2.0, 3.0, 4.0, 5.0])
>>> output = mindspore.ops.igamma(input, other)
>>> print(output)
[0.5939941  0.35276785 0.21486954 0.13337176]
```

mindspore.ops.igammac

`mindspore.ops.igammac(input, other)`

计算正则化的上层不完全伽马函数。

将 *input* 记作 a ，*other* 记作 x ，则正则化的上层不完全伽马函数可以表示为：

$$Q(a, x) = \text{Gamma}(a, x) / \text{Gamma}(a) = 1 - P(a, x)$$

其中，

$$\text{Gamma}(a, x) = \int_x^{\infty} t^{a-1} \exp(-t) dt$$

表示上层不完全伽马函数。

$P(a, x)$ 表示正则化的下层不完全伽马函数。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 第一个输入 tensor。
- **other** (Tensor) - 第二个输入 tensor。

返回：

Tensor，数据类型与 *input* 和 *other* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([2.0, 4.0, 6.0, 8.0])
>>> other = mindspore.tensor([2.0, 3.0, 4.0, 5.0])
>>> output = mindspore.ops.igammac(input, other)
>>> print(output)
[0.40600574 0.6472322 0.78513044 0.8666282 ]
```

mindspore.ops.imag

`mindspore.ops.imag(input)`

返回输入 tensor 每个元素的虚部。如果元素为实数，则返回零。

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.22+0.3511j, -0.55-0.6796j, -1.92-0.033j, -0.38-0.2991j])
>>> output = mindspore.ops.imag(input)
>>> print(output)
[ 0.3511 -0.6796 -0.033 -0.2991]
```

mindspore.ops.i0

`mindspore.ops.i0(input)`

详情请参考 `mindspore.ops.bessel_i0()`。当前接口的参数 `input` 与参考接口中的参数 `x` 实际为同一参数。

支持平台:

GPU CPU

mindspore.ops.inv

`mindspore.ops.inv(x)`

逐元素计算输入 `tensor` 的倒数。

$$out_i = \frac{1}{x_i}$$

参数:

- **x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.25, 0.4, 0.31, 0.52])
>>> output = mindspore.ops.inv(input)
>>> print(output)
[4.         2.5         3.2258065 1.923077 ]
```

mindspore.ops.invert

`mindspore.ops.invert(x)`

逐元素按位翻转。如: 01010101 变为 10101010。

$$out_i = \sim x_i$$

参数:

- **x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> mindspore.ops.invert(mindspore.tensor([25, 4, 13, 9], mindspore.int16))
Tensor(shape=[4], dtype=Int16, value= [-26,  -5, -14, -10])
```

mindspore.ops.lcm

`mindspore.ops.lcm(input, other)`

逐元素计算两个输入 `tensor` 的最小公倍数。

支持广播, 支持隐式类型转换、类型提升。

参数:

- **input** (Tensor) - 第一个输入 `tensor`。
- **other** (Tensor) - 第二个输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> mindspore.ops.lcm(mindspore.tensor([7, 8, 9]), mindspore.tensor([14, 6, 12]))
Tensor(shape=[3], dtype=Int64, value= [14, 24, 36])
```

mindspore.ops.ldexp

`mindspore.ops.ldexp(x, other)`

逐元素将输入 `tensor` 乘以 2^{other} 。

该函数通常用于将尾数 x 和由指数 `other` 创建的 2 的整数次幂相乘来构造浮点数:

$$out_i = x_i * (2^{other_i})$$

参数:

- **x** (Tensor) - 尾数 `tensor`。
- **other** (Tensor) - 指数 `tensor`, 通常为整数。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1.], mindspore.float32)
>>> other = mindspore.tensor([1, 2, 3, 4], mindspore.int32)
>>> out = mindspore.ops.ldexp(x, other)
>>> print(out)
[ 2.  4.  8. 16.]
>>> x = mindspore.tensor([[1.], [2]], mindspore.float32)
>>> other = mindspore.tensor([[1.], [2]], mindspore.int32)
>>> out = mindspore.ops.ldexp(x, other)
>>> print(out)
[[2.]
 [8.]]
```

mindspore.ops.lerp

`mindspore.ops.lerp(input, end, weight)`

基于权重参数计算两个 `tensor` 之间的线性插值。

$$output_i = input_i + weight_i * (end_i - input_i)$$

说明:

- 输入 `input` 和 `end` 的 `shape` 必须是可广播的。
- 如果 `weight` 为 `tensor`, 则 `weight`、`input` 和 `end` 的 `shape` 必须是可广播的。
- 在 Ascend 平台上, 若 `weight` 为浮点数, 则 `input` 与 `end` 类型应为 `float32`。

参数:

- **input** (`Tensor`) - 起始点。
- **end** (`Tensor`) - 终止点。
- **weight** (`Union[float, Tensor]`) - 线性插值公式的权重参数。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> start = mindspore.tensor([1., 2., 3., 4.], mindspore.float32)
>>> end = mindspore.tensor([10., 10., 10., 10.], mindspore.float32)
>>> output = mindspore.ops.lerp(start, end, 0.5)
>>> print(output)
[5.5 6.  6.5 7. ]
>>> output = mindspore.ops.lerp(start, end, mindspore.tensor([0.5, 0.5, 0.5, 0.5], mindspore.
↪float32))
>>> print(output)
[5.5 6.  6.5 7. ]
```

mindspore.ops.log

`mindspore.ops.log(input)`

逐元素计算输入 `tensor` 的自然对数。

$$y_i = \log_e(x_i)$$

警告：如果输入值在 (0, 0.01] 或 [0.95, 1.05] 范围内，则输出精度可能会存在误差。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.ops.log(x)
>>> print(output)
[0.          0.6931472  1.3862944]
```

mindspore.ops.log2

`mindspore.ops.log2(input)`

逐元素计算输入 `tensor` 以 2 为底的对数。

$$y_i = \log_2(input_i)$$

警告：如果 `log2` 的输入值范围在 (0, 0.01] 或 [0.95, 1.05] 区间，输出精度可能会受影响。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：


```
>>> import mindspore
>>> x = mindspore.tensor([2, 4, 8], mindspore.float16)
>>> output = mindspore.ops.log2(x)
>>> print(output)
[1. 2. 3.]
```

mindspore.ops.log10

`mindspore.ops.log10(input)`

逐元素计算 `tensor` 以 10 为底的对数。

$$y_i = \log_{10}(input_i)$$

警告：如果 `log10` 的输入值范围在 (0, 0.01] 或 [0.95, 1.05] 区间，输出精度可能会受影响。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([2, 4, 10], mindspore.float16)
>>> output = mindspore.ops.log10(x)
>>> print(output)
[0.301 0.602 1. ]
```

mindspore.ops.log1p

`mindspore.ops.log1p(input)`

逐元素计算 $(\text{tensor} + 1)$ 的自然对数。

$$out_i = \log_e(input_i + 1)$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.ops.log1p(x)
>>> print(output)
[0.6931472 1.0986123 1.609438 ]
```

mindspore.ops.logaddexp

`mindspore.ops.logaddexp(input, other)`

计算输入的指数和的对数。该函数可以很好地解决统计学中计算得到的事件概率过小（超过了正常浮点数表达的范围）的问题。

$$out_i = \log(\exp(input_i) + \exp(other_i))$$

参数：

- **input** (Tensor) - 输入 tensor。
- **other** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x1 = mindspore.tensor([1, 2, 3], mindspore.float16)
>>> x2 = mindspore.tensor(2, mindspore.float16)
>>> output = mindspore.ops.logaddexp(x1, x2)
>>> print(output)
[2.312 2.693 3.312]
```

mindspore.ops.logaddexp2

`mindspore.ops.logaddexp2(input, other)`

计算以 2 为底的输入的指数和的对数。

$$out_i = \log_2(2^{input_i} + 2^{other_i})$$

参数：

- **input** (Tensor) - 输入 tensor。
- **other** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x1 = mindspore.tensor([1, 2, 3], mindspore.float16)
>>> x2 = mindspore.tensor(2, mindspore.float16)
>>> output = mindspore.ops.logaddexp2(x1, x2)
>>> print(output)
[2.586 3.      3.586]
```

mindspore.ops.logical_and

`mindspore.ops.logical_and(input, other)`

逐元素计算两个 `tensor` 的逻辑与运算。

$$out_i = input_i \wedge other_i$$

说明:

- 支持广播。
- 支持隐式类型转换。

参数:

- **input** (`Union[Tensor, bool]`) - 第一个输入。
- **other** (`Union[Tensor, bool]`) - 第二个输入。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> y = mindspore.tensor([True, True, False], mindspore.bool_)
>>> output = mindspore.ops.logical_and(x, y)
>>> print(output)
[ True False False]
>>> x = mindspore.tensor(1, mindspore.bool_)
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.ops.logical_and(x, y)
>>> print(output)
False
>>> x = True
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.ops.logical_and(x, y)
>>> print(output)
False
>>> x = True
>>> y = mindspore.tensor([True, False], mindspore.bool_)
>>> output = mindspore.ops.logical_and(x, y)
```

(下页继续)

(续上页)

```
>>> print(output)
[True False]
```

mindspore.ops.logical_not

`mindspore.ops.logical_not(input)`
逐元素计算输入 `tensor` 的逻辑非运算。

$$out_i = \neg input_i$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> output = mindspore.ops.logical_not(x)
>>> print(output)
[False True False]
```

mindspore.ops.logical_or

`mindspore.ops.logical_or(input, other)`
逐元素计算两个 `tensor` 的逻辑或运算。

$$out_i = input_i \vee other_i$$

说明:

- 支持广播。
- 支持隐式类型转换。

参数:

- **input** (Union[Tensor, bool]) - 第一个输入。
- **other** (Union[Tensor, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> y = mindspore.tensor([True, True, False], mindspore.bool_)
>>> output = mindspore.ops.logical_or(x, y)
>>> print(output)
[ True True  True]
>>> x = mindspore.tensor(1, mindspore.bool_)
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.ops.logical_or(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.ops.logical_or(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor([True, False], mindspore.bool_)
>>> output = mindspore.ops.logical_or(x, y)
>>> print(output)
[True True]
```

mindspore.ops.logical_xor

`mindspore.ops.logical_xor(input, other)`

逐元素计算两个 `tensor` 的逻辑异或运算。

$$out_i = input_i \oplus other_i$$

说明:

- 支持广播。
- 支持隐式类型转换。

参数:

- **input** (Tensor) - 第一个输入 `tensor`。
- **other** (Tensor) - 第二个输入 `tensor`。

返回:

Tensor

支持平台:

Ascend CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> y = mindspore.tensor([True, True, False], mindspore.bool_)
```

(下页继续)

(续上页)

```

>>> output = mindspore.ops.logical_xor(x, y)
>>> print(output)
[False True True]
>>> x = mindspore.tensor(1, mindspore.bool_)
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.ops.logical_xor(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.ops.logical_xor(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor([True, False], mindspore.bool_)
>>> output = mindspore.ops.logical_xor(x, y)
>>> print(output)
[False True]

```

mindspore.ops.logit

`mindspore.ops.logit(input, eps=None)`

逐元素计算 `tensor` 的逻辑回归函数值。

公式如下：

$$y_i = \ln\left(\frac{z_i}{1-z_i}\right)$$

$$z_i = \begin{cases} input_i & \text{if } eps \text{ is None} \\ eps & \text{if } input_i < eps \\ input_i & \text{if } eps \leq input_i \leq 1 - eps \\ 1 - eps & \text{if } input_i > 1 - eps \end{cases}$$

参数：

- **input** (Tensor) - 输入 `tensor`。
- **eps** (float, 可选) - 用于输入限制边界的 `epsilon` 值。当 `eps` 不是 `None` 时，输入的数值界限被定义 `[eps, 1-eps]`，否则没有界限。默认 `None`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> x = mindspore.tensor([0.1, 0.2, 0.3], mindspore.float32)
>>> output = mindspore.ops.logit(x, eps=1e-5)
>>> print(output)
[-2.1972246 -1.3862944 -0.8472978]

```

mindspore.ops.mul

`mindspore.ops.mul(input, other)`

两个 Tensor 逐元素相乘。

$$out_i = input_i * other_i$$

说明:

- 当两个输入具有不同的 shape 时，它们的 shape 必须能够广播为一个共同的 shape。
- 两个输入不能同时为 bool 类型。[True, Tensor(True, bool_), Tensor(np.array([True]), bool_)] 等都属于 bool 类型。
- 支持隐式类型转换、类型提升。

参数:

- **input** (Union[Tensor, number.Number, bool]) - 第一个输入。
- **other** (Union[Tensor, number.Number, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> # case 1: The shape of two inputs are different
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> output = mindspore.ops.mul(x, 100)
>>> print(output)
[100. 200. 300.]
>>> # case 2: The shape of two inputs are the same
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> y = mindspore.tensor([4.0, 5.0, 6.0], mindspore.float32)
>>> output = mindspore.ops.mul(x, y)
>>> print(output)
[ 4. 10. 18.]
```

mindspore.ops.multiply

`mindspore.ops.multiply(input, other)`

`mindspore.ops.mul()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.mvlgamma

`mindspore.ops.mvlgamma(input, p)`

逐元素计算 p 维多元对数伽马函数值。

Mvlgamma 计算公式如下：

$$\log(\Gamma_p(input)) = C + \sum_{i=1}^p \log(\Gamma(input - \frac{i-1}{2}))$$

其中 $C = \log(\pi) \times \frac{p(p-1)}{4}$ ， $\Gamma(\cdot)$ 为 Gamma 函数。

参数：

- **input** (Tensor) - 多元对数伽马函数的输入 tensor。*input* 中每个元素的值必须大于 $(p-1)/2$ 。
- **p** (int) - 维度数量，必须大于等于 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[3, 4, 5], [4, 2, 6]], mindspore.float32)
>>> y = mindspore.ops.mvlgamma(x, p=3)
>>> print(y)
[[2.694925  5.402975  9.140645]
 [5.402975  1.596312 13.64045]]
```

mindspore.ops.neg

`mindspore.ops.neg(input)`

计算输入 *input* 的相反数并返回。

$$out_i = -input_i$$

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, -1, 2, 0, -3.5], mindspore.float32)
>>> output = mindspore.ops.neg(input)
>>> print(output)
[-1.  -2.   1.  -2.   0.   3.5]
```


mindspore.ops.negative

`mindspore.ops.negative(input)`
`mindspore.ops.neg()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.nextafter

`mindspore.ops.nextafter(input, other)`
 逐元素计算 *input* 指向 *other* 的下一个可表示浮点值。

$$out_i = \begin{cases} input_i + eps, & \text{if } input_i < other_i \\ input_i - eps, & \text{if } input_i > other_i \\ input_i, & \text{if } input_i = other_i \end{cases}$$

其中 *eps* 为输入 *tensor* 数据类型最小可表示增量值。

更多详细信息请参见 [A Self Regularized Non-Monotonic Neural Activation Function](#)。

参数:

- **input** (Tensor) - 第一个输入 *tensor*
- **other** (Tensor) - 第二个输入 *tensor*。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.0], mindspore.float32)
>>> other = mindspore.tensor([0.1], mindspore.float32)
>>> output = mindspore.ops.nextafter(input, other)
>>> print(output)
[1.e-45]
```

mindspore.ops.polar

`mindspore.ops.polar(abs, angle)`
 将极坐标转化为笛卡尔坐标。

返回一个复数 *tensor*，其元素是由输入极坐标构造的笛卡尔坐标。其中极坐标由极径 *abs* 和极角 *angle* 给定。

$$y_i = abs_i * \cos(angle_i) + abs_i * \sin(angle_i) * j$$

参数:

- **abs** (Tensor, float) - 极径。
- **angle** (Tensor, float) - 极角。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> abs = mindspore.tensor([1, 2], mindspore.float32)
>>> angle = mindspore.tensor([np.pi / 2, 5 * np.pi / 4], mindspore.float32)
>>> output = mindspore.ops.polar(abs, angle)
>>> print(output)
[ -4.3711388e-08+1.j          -1.4142137e+00-1.4142134j]
```

mindspore.ops.polygamma

`mindspore.ops.polygamma(n, input)`
计算 *input* 的多伽马函数的 *n* 阶导数。

$$\psi^{(a)}(x) = \frac{d^{(a)}}{dx^{(a)}} \psi(x)$$

其中 $\psi(x)$ 为 digamma 函数。

参数：

- **n** (Tensor) - 多伽马函数求导的阶数。
- **input** (Tensor) - 输入 tensor。

返回：

Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([3.14, -2.71], mindspore.float64)
>>> a = mindspore.tensor(1, mindspore.int64)
>>> output = mindspore.ops.polygamma(a, x)
>>> print(output)
[ 0.37446456 15.49884838]
```

mindspore.ops.positive

`mindspore.ops.positive(input)`

返回输入 `tensor`。

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([-5.0, 1.5, 3.0, 100.0], mindspore.float32)
>>> print(mindspore.ops.positive(x))
[ -5.    1.5    3.   100.]
```

mindspore.ops.pow

`mindspore.ops.pow(input, exponent)`

计算 `input` 中每个元素的 `exponent` 次幂。

说明:

- 支持广播。
- 支持隐式类型转换、类型提升。

$$out_i = input_i^{exponent_i}$$

参数:

- **input** (Union[Tensor, Number]) - 第一个输入。
- **exponent** (Union[Tensor, Number]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.ops.pow(input, exponent=3.0)
>>> print(output)
[ 1.   8.  64.]
>>>
>>> input = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
```

(下页继续)

(续上页)

```
>>> exponent = mindspore.tensor([2.0, 4.0, 3.0], mindspore.float32)
>>> output = mindspore.ops.pow(input, exponent)
>>> print(output)
[ 1. 16. 64.]
```

mindspore.ops.rad2deg

mindspore.ops.rad2deg(*x*)

逐元素将 *x* 从弧度转换为度数。

参数:

- **x** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[6.283, -3.142], [1.570, -6.283], [3.142, -1.570]], mindspore.
    ↳float32)
>>> output = mindspore.ops.rad2deg(x)
>>> print(output)
[[ 359.98935 -180.02333]
 [  89.95438 -359.98935]
 [ 180.02333 -89.95438]]
```

mindspore.ops.ravel

mindspore.ops.ravel(*input*)

沿着 0 轴方向, 将多维 Tensor 展开成一维。

参数:

- **input** (Tensor) - 要展开的 Tensor。

返回:

一维 Tensor, 含有与输入相同的元素。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[0, 1], [2, 1]]).astype(np.float32))
>>> output = ops.ravel(x)
>>> print(output)
[0. 1. 2. 1.]
>>> print(output.shape)
(4,)
```

mindspore.ops.real

`mindspore.ops.real(input)`

返回输入 `tensor` 的实数部分。如果输入是实数，则返回值与输入值相同。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor(1.3+0.4j, mindspore.complex64)
>>> output = mindspore.ops.real(input)
>>> print(output)
1.3
```

mindspore.ops.reciprocal

`mindspore.ops.reciprocal(input)`

返回输入 `tensor` 每个元素的倒数。

$$out_i = \frac{1}{x_i}$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.ops.reciprocal(input)
>>> print(output)
[1.    0.5  0.25]
```

mindspore.ops.remainder

`mindspore.ops.remainder(input, other)`

逐元素计算输入 `tensor` 的除法余数。

支持隐式类型转换、类型提升。

```
remainder(input, other) == input - input.div(other, rounding_mode="floor") * other
```

警告:

- 当输入元素超过 2048 时, 可能会有精度问题。
- 在 Ascend 和 CPU 上的计算结果可能不一致。
- 如果 `shape` 表示为 $(D_1, D_2 \cdots D_n)$, 那么 $D_1 * D_2 \cdots D_n \leq 1000000$, $n \leq 8$ 。

参数:

- **input** (`Union[Tensor, numbers.Number, bool]`) - 第一个输入。
- **other** (`Union[Tensor, numbers.Number, bool]`) - 第二个输入。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> x = mindspore.tensor([-4.0, 5.0, 6.0], mindspore.float16)
>>> y = mindspore.tensor([3.0], mindspore.float16)
>>> output = mindspore.ops.remainder(x, y)
>>> print(output)
[2.  2.  0.]
>>> # case 2: The shape of two inputs are the same
>>> x = mindspore.tensor([-4.0, 5.0, 6.0], mindspore.float16)
>>> y = mindspore.tensor([3.0, 2.0, 3.0], mindspore.float16)
>>> output = mindspore.ops.remainder(x, y)
>>> print(output)
[2.  1.  0.]
```

mindspore.ops.rot90

`mindspore.ops.rot90(input, k, dims)`

沿指定维度的平面内将 n -D tensor 旋转 90 度。如果 $k > 0$ ，旋转方向是从第一轴朝向第二轴，如果 $k < 0$ ，旋转方向从第二轴朝向第一轴。

参数：

- **input** (Tensor) - 输入 tensor。
- **k** (int) - 旋转的次数。
- **dims** (Union[list(int), tuple(int)]) - 指定维度。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 2, 3],
...                       [4, 5, 6]], mindspore.float32)
>>> output = mindspore.ops.rot90(x, k=1, dims=(0, 1))
>>> print(output)
[[3. 6.]
 [2. 5.]
 [1. 4.]]
>>> mindspore.ops.rot90(x, k=1, dims=(1, 0)) == mindspore.ops.rot90(x, k=-1, dims=(0, 1))
Tensor(shape=[3, 2], dtype=Bool, value=
[[ True,  True],
 [ True,  True],
 [ True,  True]])
>>> # when input array has ndim>2
>>> x = mindspore.tensor([[[1, 2, 3],
...                       [4, 5, 6]],
...                       [[7, 8, 9],
...                       [10, 11, 12]]], mindspore.float32)
>>> output = mindspore.ops.rot90(x, k=1, dims=(2, 1))
>>> print(output)
[[[ 4.  1.]
 [ 5.  2.]
 [ 6.  3.]]
 [[10.  7.]
 [11.  8.]
 [12.  9.]]]
```

mindspore.ops.round

`mindspore.ops.round(input, *, decimals=0)`

对输入数据四舍五入到最接近的整数。

$$out_i \approx input_i$$

说明: Ascend 平台支持的输入数据类型包括 `bfloat16` (Atlas 训练系列产品不支持)、`float16`、`float32`、`float64`、`int32`、`int64`。

参数:

- **input** (Tensor) - 输入 tensor。

关键字参数:

- **decimals** (int, 可选) - 要舍入到的小数位数。如果为负数, 则指定小数点左侧的位数。默认 0。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.round(mindspore.tensor([4.7, -2.3, 9.1, -7.7]))
>>> print(output)
[ 5. -2.  9. -8.]
>>> # Values equidistant from two integers are rounded towards the
>>> # the nearest even value (zero is treated as even)
>>> output = mindspore.ops.round(mindspore.tensor([-0.5, 0.5, 1.5, 2.5]))
>>> print(output)
[0. 0.  2.  2.]
>>> # A positive decimals argument rounds to the to that decimal place
>>> output = mindspore.ops.round(mindspore.tensor([0.1234567]), decimals=3)
>>> print(output)
[0.123]
>>> # A negative decimals argument rounds to the left of the decimal
>>> output = mindspore.ops.round(mindspore.tensor([1200.1234567]), decimals=-3)
>>> print(output)
[1000.]
```

mindspore.ops.rsqrt

`mindspore.ops.rsqrt(input)`

逐元素计算输入 tensor 的平方根倒数。

$$out_i = \frac{1}{\sqrt{input_i}}$$

参数:

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([-0.0370, 0.2970, 1.5420, -0.9105])
>>> output = mindspore.ops.rsqrt(input)
>>> print(output)
[          nan 1.8349396 0.8053002          nan]
```

mindspore.ops.sgn

`mindspore.ops.sgn(input)`

`mindspore.ops.sign()` 在复数上的扩展。对于实数输入，此方法与 `mindspore.ops.sign()` 一致。对于复数输入，此方法按照如下公式计算：

$$\text{out}_i = \begin{cases} 0 & |\text{input}_i| == 0 \\ \frac{\text{input}_i}{|\text{input}_i|} & \text{otherwise} \end{cases}$$

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[3+4j, 7-24j, 0, 6+8j, 8], [15+20j, 7-24j, 0, 3+4j, 20]], dtype=
↳mindspore.complex64)
>>> output = mindspore.ops.sgn(input)
>>> print(output)
[[0.6 +0.8j 0.28-0.96j 0. +0.j 0.6 +0.8j 1. +0.j ]
 [0.6 +0.8j 0.28-0.96j 0. +0.j 0.6 +0.8j 1. +0.j ]]
```

mindspore.ops.sign

`mindspore.ops.sign(input)`

按 `sign` 公式逐元素计算输入 tensor。

$$\text{out}_i = \begin{cases} -1 & \text{input}_i < 0 \\ 0 & \text{input}_i = 0 \\ 1 & \text{input}_i > 0 \end{cases}$$

说明：在输入为 NaN 且数据类型为 float64 时，该算子计算结果为 NaN。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[ -1, 0, 2, 4, 6], [2, 3, 5, -6, 0]])
>>> output = mindspore.ops.sign(input)
>>> print(output)
[[-1  0  1  1  1]
 [ 1  1  1 -1  0]]
>>> mindspore.set_device(device_target="CPU")
>>> x = mindspore.tensor([[-1, 0, float('inf'), 4, float('nan')], [2, 3, float('-inf'), -6, 0]])
>>> output = mindspore.ops.sign(x)
>>> print(output)
[[-1.  0.  1.  1.  0.]
 [ 1.  1. -1. -1.  0.]]
```

mindspore.ops.signbit

mindspore.ops.signbit (*input*)

判断每个元素的符号, 如果元素值小于 0 则对应输出的位置为 **True**, 否则为 **False**。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.3, 1.2, 0., -2.5])
>>> output = mindspore.ops.signbit(input)
>>> print(output)
[False False False  True]
```

mindspore.ops.sin

`mindspore.ops.sin(input)`

逐元素计算输入 `tensor` 的正弦。

$$output_i = \sin(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.62, 0.28, 0.43, 0.62], mindspore.float32)
>>> output = mindspore.ops.sin(input)
>>> print(output)
[0.58103514 0.27635565 0.4168708 0.58103514]
```

mindspore.ops.sinc

`mindspore.ops.sinc(input)`

计算输入的归一化正弦值。

$$out_i = \begin{cases} \frac{\sin(\pi input_i)}{\pi input_i} & input_i \neq 0 \\ 1 & input_i = 0 \end{cases}$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.62, 0.28, 0.43, 0.62], mindspore.float32)
>>> output = mindspore.ops.sinc(input)
>>> print(output)
[0.47735003 0.8759357 0.7224278 0.47735003]
```

mindspore.ops.sinh

`mindspore.ops.sinh(input)`

逐元素计算输入 `tensor` 的双曲正弦。

$$output_i = \sinh(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.62, 0.28, 0.43, 0.62], mindspore.float32)
>>> output = mindspore.ops.sinh(input)
>>> print(output)
[0.6604918  0.28367308 0.44337422 0.6604918 ]
```

mindspore.ops.sqrt

`mindspore.ops.sqrt(x)`

逐元素计算 `tensor` 的平方根。

$$out_i = \sqrt{x_i}$$

参数:

- **x** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1.0, 4.0, 9.0], mindspore.float32)
>>> output = mindspore.ops.sqrt(x)
>>> print(output)
[1. 2. 3.]
```

mindspore.ops.square

`mindspore.ops.square(input)`

逐元素计算 `tensor` 的平方。

$$y_i = input_i^2$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> output = mindspore.ops.square(input)
>>> print(output)
[1. 4. 9.]
```

mindspore.ops.sub

`mindspore.ops.sub(input, other)`

逐元素计算第一个输入减去第二个输入的值。

$$out_i = input_i - other_i$$

说明：

- 当两个输入具有不同的 `shape` 时，它们的 `shape` 必须要能广播为一个共同的 `shape`。
- 两个输入不能同时为 `bool` 类型。[True, Tensor(True, bool_), Tensor(np.array([True]), bool_)] 等都为 `bool` 类型。
- 支持隐式类型转换、类型提升。

参数：

- **input** (Union[Tensor, number.Number, bool]) - 第一个输入。
- **other** (Union[Tensor, number.Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([4, 5, 6], mindspore.int32)
>>> output = mindspore.ops.sub(input, other)
>>> print(output)
[-3 -3 -3]
```

mindspore.ops.subtract

`mindspore.ops.subtract(input, other, *, alpha=1)`

从 *input* 中减去经 *alpha* 缩放的 *other*。

支持隐式类型转换、类型提升。

$$output[i] = input[i] - alpha * other[i]$$

参数:

- **input** (Union[Tensor, number.Number]) - 第一个输入。
- **other** (Union[Tensor, number.Number]) - 第二个输入。

关键字参数:

- **alpha** (number) - *other* 的乘数。默认 1。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([4, 5, 6], mindspore.float32)
>>> y = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> z = mindspore.ops.subtract(input, y, alpha=1)
>>> print(z)
[3. 3. 3.]
```

mindspore.ops.t

`mindspore.ops.t(input)`

转置二维 tensor，维度小于二的 tensor 按原样返回。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: Input is a 0-D tensor
>>> mindspore.ops.t(mindspore.tensor(6))
Tensor(shape=[], dtype=Int64, value= 6)
>>>
>>> # case 2: Input is a 1-D tensor
>>> mindspore.ops.t(mindspore.tensor([1, 2]))
Tensor(shape=[2], dtype=Int64, value= [1, 2])
>>>
>>> # case 3: Input is a 2-D tensor
>>> mindspore.ops.t(mindspore.tensor([[1, 2, 3], [2, 3, 4]]))
Tensor(shape=[3, 2], dtype=Int64, value=
[[1, 2],
 [2, 3],
 [3, 4]])
```

mindspore.ops.tan

`mindspore.ops.tan(input)`

逐元素计算输入 `tensor` 的正切值。

$$out_i = \tan(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.tan(mindspore.tensor([-1.0, 0.0, 1.0]))
>>> print(output)
[-1.5574077 0. 1.5574077]
```

mindspore.ops.tanhshrink

`mindspore.ops.tanhshrink(input)`

逐元素计算输入 `tensor` 的 `Tanhshrink` 激活函数值。

$$\text{Tanhshrink}(x) = x - \text{Tanh}(x)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.tanhshrink(mindspore.tensor([1., 2., 3., 2., 1.]))
>>> print(output)
[0.23840582 1.0359724 2.0049443 1.0359724 0.23840582]
```

mindspore.ops.trapz

`mindspore.ops.trapz(y, x=None, *, dx=1.0, dim=-1)`

计算沿指定维度的梯形法则积分。采样点之间的距离由 `tensor x` 或标量 `dx` 指定，默认 1。

$$\int y(x)dx$$

参数:

- **y** (Tensor) - 输入 tensor。
- **x** (Tensor, 可选) - 如果指定，则定义采样点之间的间距。

关键字参数:

- **dx** (float, 可选) - 采样点之间的常数间距，默认 1.0。如果 `x` 被指定，则 `dx` 不生效。
- **dim** (int, 可选) - 指定维度。默认 -1。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> y = mindspore.tensor([[1., 2., 3.], [2., 3., 4.], [3., 2., 1.]])
>>> # case 1: Integrate over a regular grid, with spacing 1.
>>> output = mindspore.ops.trapz(y, dx=1.)
>>> print(output)
[4. 6. 4.]
>>>
>>> # case 2: Integrate over an irregular grid.
>>> x = mindspore.tensor([[1, 2, 3], [1, 3, 5], [1, 4, 7]])
>>> output = mindspore.ops.trapz(y, x)
>>> print(output)
[ 4. 12. 12.]
```


`mindspore.ops.tril_indices`

`mindspore.ops.tril_indices` (*row*, *col*, *offset*=0, *, *dtype*=*mstype.int64*)

返回一个二维的 `tensor`，包含 *row* * *col* 矩阵的下三角元素的索引。第一行是所有索引的行坐标，第二行是所有索引的列坐标。坐标先按行排序，后按列排序。

说明：在 CUDA 上运行时，*row* * *col* 必须小于 2^{59} ，以防止计算时溢出。

参数：

- **row** (int) - 二维 `tensor` 的行数。
- **col** (int) - 二维 `tensor` 的列数。
- **offset** (int, 可选) - 对角线偏移量。默认 0。
 - 当 *offset* 是正整数时，对角线向上方偏移。
 - 当 *offset* 是负整数时，对角线向下方偏移。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 指定数据类型，支持 *mstype.int32* 和 *mstype.int64*，默认 *mstype.int64*。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: By default, offset=0, all elements on and below the main diagonal are retained.
>>> mindspore.ops.tril_indices(3, 2, 0)
Tensor(shape=[2, 5], dtype=Int64, value=
[[0, 1, 1, 2, 2],
 [0, 0, 1, 0, 1]])
>>>
>>> # case 2: Offset=1, the indices on and below the first sub-diagonal above the main
↪diagonal are returned.
>>> mindspore.ops.tril_indices(3, 2, 1)
Tensor(shape=[2, 6], dtype=Int64, value=
[[0, 0, 1, 1, 2, 2],
 [0, 1, 0, 1, 0, 1]])
>>>
>>> # case 3: Offset=-1, the indices on and below the first sub-diagonal below the main
↪diagonal are returned.
>>> mindspore.ops.tril_indices(3, 2, -1)
Tensor(shape=[2, 3], dtype=Int64, value=
[[1, 2, 2],
 [0, 0, 1]])
```

mindspore.ops.triu_indices

`mindspore.ops.triu_indices(row, col, offset=0, *, dtype=mstype.int64)`

返回一个二维的 `tensor`，包含 `row * col` 矩阵的上三角元素的索引。第一行是所有索引的行坐标，第二行是所有索引的列坐标。坐标先按行排序，后按列排序。

说明：在 CUDA 上运行的时候，`row * col` 必须小于 2^{59} 以防止计算时溢出。

参数：

- **row** (int) - 二维 `tensor` 的行数。
- **col** (int) - 二维 `tensor` 的列数。
- **offset** (int, 可选) - 对角线偏移量。默认 0。
 - 当 `offset` 是正整数时，对角线向上方偏移。
 - 当 `offset` 是负整数时，对角线向下方偏移。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型，支持 `mstype.int32` 和 `mstype.int64`，默认 `mstype.int64`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: By default, offset=0, all elements on and below the main diagonal are retained.
>>> mindspore.ops.triu_indices(3, 2, 0)
Tensor(shape=[2, 3], dtype=Int64, value=
[[0, 0, 1],
 [0, 1, 1]])
>>>
>>> # case 2: If offset=-1, the indices on and above the first sub-diagonal below the main_
↪diagonal are returned.
>>> mindspore.ops.triu_indices(3, 2, -1)
Tensor(shape=[2, 5], dtype=Int64, value=
[[0, 0, 1, 1, 2],
 [0, 1, 0, 1, 1]])
```

mindspore.ops.true_divide

`mindspore.ops.true_divide (dividend, divisor)`

`mindspore.ops.div()` 在 `rounding_mode=None` 时的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.trunc

`mindspore.ops.trunc (input)`

返回一个 `tensor`, 包含输入元素的截断整数值。

参数:

- **input** (`Tensor`) - 输入 `tensor`。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.trunc(mindspore.tensor([3.4742, 0.5466, -0.8008, -3.9079]))
>>> print(output)
[3. 0. 0. -3.]
```

mindspore.ops.truncate_div

`mindspore.ops.truncate_div (x, y)`

将 x 和 y 逐元素相除, 结果将向 0 取整。

说明: 支持隐式类型转换, 支持广播。

参数:

- **x** (`Union[Tensor, Number, bool]`) - 第一个输入 `tensor`。
- **y** (`Union[Tensor, Number, bool]`) - 第二个输入 `tensor`。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> mindspore.ops.truncate_div(mindspore.tensor([2, 4, -1]), mindspore.tensor([3, 3, 3]))
Tensor(shape=[3], dtype=Int64, value= [0, 1, 0])
```

`mindspore.ops.truncate_mod`

`mindspore.ops.truncate_mod(x, y)`

将 x 和 y 逐元素取模。

支持隐式类型转换，支持广播。

警告：

- 输入数值不能为 0。
- 当输入含有超过 2048 个元素时，该操作不能保证千分之二的精度要求。
- 由于架构不同，该算子在 NPU 和 CPU 上的计算结果可能不一致。
- 若 shape 为 (D1、D2...、Dn)，则 $D1 * D2 * \dots * DN \leq 1000000$ ， $n \leq 8$ 。

参数：

- **x** (Union[Tensor, Number, bool]) - 第一个输入 tensor。
- **y** (Union[Tensor, Number, bool]) - 第二个输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> output = mindspore.ops.truncate_mod(mindspore.tensor([2., 4., -1.]), mindspore.tensor([3., 3., 3.]))
↩, 3., 3.]))
>>> print(output)
[ 2.  1. -1.]
```

`mindspore.ops.xdiv`

`mindspore.ops.xdiv(x, y)`

x 和 y 逐元素相除。

说明：

- 支持广播，支持隐式类型转换、类型提升。
- 当 x 和 y 的数据类型都为复数时，必须同时为 `complex64` 或 `complex128`。
- x 和 y 不能同时为 `bool` 类型。

参数：

- **x** (Union[Tensor, Number, bool]) - 被除数（分子）。
- **y** (Union[Tensor, Number, bool]) - 除数（分母）。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> output = mindspore.ops.xdivy(mindspore.tensor([2., 4., -1.]), mindspore.tensor([2., 2., 2.]))
>>> print(output)
[ 1.  2. -0.5]
```

mindspore.ops.xlogy

`mindspore.ops.xlogy(input, other)`

逐元素对 *other* 取对数，再与 *input* 相乘。

$$out_i = input_i * \ln other_i$$

说明：

- 支持广播，支持隐式类型转换、类型提升。

警告： Ascend 平台，*input* 和 *other* 必须为 float16 或 float32。

参数：

- **input** (Union[Tensor, numbers.Number, bool]) - 第一个输入 tensor。
- **other** (Union[Tensor, numbers.Number, bool]) - 第二个输入 tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> output = mindspore.ops.xlogy(mindspore.tensor([-5., 0., 4.]), mindspore.tensor([2., 2., 2.]))
>>> print(output)
[-3.465736  0.          2.7725887]
```

mindspore.ops.zeta

mindspore.ops.zeta(input, other)
逐元素计算 Hurwitz zeta 的值。

$$\zeta(x, q) = \sum_{k=0}^{\infty} \frac{1}{(k + q)^x}$$

警告：这是一个实验性 API，后续可能修改或删除。

- 参数：
- **input** (Union[Tensor, int, float]) - 第一个输入 tensor。在公式中表示为 x 。
 - **other** (Union[Tensor, int, float]) - 第二个输入 tensor。在公式中表示为 q 。

返回：

Tensor
支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> z = mindspore.ops.zeta(mindspore.tensor([10.]), mindspore.tensor([1.]))
>>> print(z)
[1.0009946]
```

2.2.2 Reduction 函数

接口名	概述	支持平台
<code>mindspore.ops.all</code>	检查指定轴上是否所有元素均为 <i>True</i> 。	Ascend GPU CPU
<code>mindspore.ops.amax</code>	返回 tensor 在指定轴上的最大值。	Ascend GPU CPU
<code>mindspore.ops.amin</code>	返回 tensor 在指定轴上的最小值。	Ascend GPU CPU
<code>mindspore.ops.aminmax</code>	返回 tensor 在指定轴上的最小值和最大值。	Ascend GPU CPU
<code>mindspore.ops.any</code>	检查指定轴上是否含有 <i>True</i> 。	Ascend GPU CPU
<code>mindspore.ops.argmax</code>	返回 tensor 在指定维度上的最大值索引。	Ascend GPU CPU
<code>mindspore.ops.argmin</code>	返回 tensor 在指定轴上的最大值索引。	Ascend GPU CPU
<code>mindspore.ops.cummax</code>	返回 tensor 在指定轴上的累积最大值及其索引。	Ascend GPU CPU
<code>mindspore.ops.cummin</code>	返回 tensor 在指定轴上的累积最小值及其索引。	Ascend GPU CPU
<code>mindspore.ops.cumprod</code>	返回 tensor 在指定维度上累积的元素乘积。	Ascend GPU CPU
<code>mindspore.ops.cumsum</code>	返回 tensor 在指定轴上累积的元素和。	Ascend GPU CPU
<code>mindspore.ops.histc</code>	计算 tensor 的直方图。	Ascend CPU
<code>mindspore.ops.logcumsumexp</code>	计算 tensor 在指定维度上的累积指数和的对数。	Ascend CPU GPU
<code>mindspore.ops.logsumexp</code>	计算 tensor 在指定维度上的指数和的对数。	Ascend GPU CPU

下页继续

表 7 - 续上页

<code>mindspore.ops.max</code>	返回 tensor 在指定轴上的最大值及其索引。	Ascend GPU CPU
<code>mindspore.ops.mean</code>	计算 tensor 在指定轴上的均值。	Ascend GPU CPU
<code>mindspore.ops.median</code>	返回 tensor 在指定轴上的中位数及其索引。	GPU CPU
<code>mindspore.ops.min</code>	返回 tensor 在指定轴上的最小值及其索引。	Ascend GPU CPU
<code>mindspore.ops.norm</code>	计算 tensor 的矩阵范数或向量范数。	Ascend GPU CPU
<code>mindspore.ops.prod</code>	返回 tensor 在指定轴上的乘积。	Ascend GPU CPU
<code>mindspore.ops.std</code>	计算 Tensor 在指定轴上的标准差。	Ascend GPU CPU
<code>mindspore.ops.std_mean</code>	计算 tensor 在指定轴上的标准差及平均值。	Ascend GPU CPU
<code>mindspore.ops.var</code>	计算 tensor 在指定轴上的方差。	Ascend GPU CPU
<code>mindspore.ops.var_mean</code>	计算 tensor 在指定轴上的方差及平均值。	Ascend GPU CPU

mindspore.ops.all

`mindspore.ops.all` (*input*, *axis=None*, *keep_dims=False*)

检查指定轴上是否所有元素均为 *True*。

参数：

- **input** (Tensor) - 输入 **tensor**。
- **axis** (Union[int, tuple(int), list(int), Tensor], 可选) - 指定轴。如果为 *None*，检查所有元素。默认 *None*。
- **keep_dims** (bool, 可选) - 输出 **tensor** 是否保留维度。默认 *False*。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: By default, mindspore.ops.all tests along all the axes.
>>> mindspore.ops.all(input)
Tensor(shape=[], dtype=Bool, value= False)
>>>
>>> # case 2: Reduces a dimension along axis 1, with keep_dims False.
>>> mindspore.ops.all(input, axis=1)
Tensor(shape=[2], dtype=Bool, value= [False,  True])
>>>
>>> # case 3: Reduces a dimension along axis (0,1), with keep_dims False.
>>> mindspore.ops.all(input, axis=(0,1))
Tensor(shape=[], dtype=Bool, value= False)
>>>
>>> # case 4: Reduces a dimension along axis [0,1], with keep_dims True.
>>> mindspore.ops.all(input, axis=[0,1], keep_dims=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[False]])
```

mindspore.ops.amax

`mindspore.ops.amax(input, axis=None, keepdims=False, *, initial=None, where=None)`

返回 `tensor` 在指定轴上的最大值。

参数:

- **input** (`Tensor[Number]`) - 输入 `tensor`。
- **axis** (`Union[int, tuple(int), list(int), Tensor]`, 可选) - 指定计算的轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **keepdims** (`bool`, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

关键字参数:

- **initial** (`scalar`, 可选) - 最大值的初始值。默认 `None`。
- **where** (`Tensor[bool]`, 可选) - 指定计算最大值的范围，该 `tensor` 的 `shape` 须可被广播到 `input` 的 `shape` 上。必须指定 `initial` 值。默认 `None`，表示计算全部元素。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: By default, compute the maximum of all elements.
>>> mindspore.ops.amax(input)
Tensor(shape=[], dtype=Int64, value= 9)
>>>
>>> # case 2: Compute maximum along axis 1.
>>> mindspore.ops.amax(input, axis=1)
Tensor(shape=[3], dtype=Int64, value= [9, 7, 8])
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.amax(input, axis=1, keepdims=True)
Tensor(shape=[3, 1], dtype=Int64, value=
[[9],
 [7],
 [8]])
>>>
>>> # case 4: Use "where" to include only specific elements in computing the maximum.
>>> where = mindspore.tensor([[0, 0, 1, 0],
...                           [0, 0, 1, 1],
...                           [1, 1, 1, 0]], dtype=mindspore.bool_)
>>> mindspore.ops.amax(input, axis=1, keepdims=True, initial=0, where=where)
Tensor(shape=[3, 1], dtype=Int64, value=
[[4],
 [7],
 [8]])
>>>
>>> # case 5: The shape of "where" must be broadcast compatible with input.
>>> where = mindspore.tensor([[False],
```

(下页继续)

(续上页)

```

...             [False],
...             [False]])
>>> mindspore.ops.amax(input, axis=0, keepdims=True, initial=0, where=where)
Tensor(shape=[1, 4], dtype=Int64, value=
[[0, 0, 0, 0]])

```

mindspore.ops.amin

`mindspore.ops.amin(input, axis=None, keepdims=False, *, initial=None, where=None)`

返回 `tensor` 在指定轴上的最小值。

参数:

- **input** (Tensor[Number]) - 输入 `tensor`。
- **axis** (Union[int, tuple(int), list(int), Tensor], 可选) - 指定计算的轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **keepdims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

关键字参数:

- **initial** (scalar, 可选) - 最小值的初始值。默认 `None`。
- **where** (Tensor[bool], 可选) - 指定计算最小值的范围，该 `tensor` 的 `shape` 须可被广播到 `input` 的 `shape` 上。必须指定 `initial` 值。默认 `None`，表示计算全部元素。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> input = mindspore.tensor([[2, 5, 1, 6],
...                           [3, -7, -2, 4],
...                           [8, -4, 1, -3]])
>>> # case 1: By default, compute the minimum of all elements.
>>> mindspore.ops.amin(input)
Tensor(shape=[], dtype=Int64, value= -7)
>>>
>>> # case 2: Compute minimum along axis 1.
>>> mindspore.ops.amin(input, axis=1)
Tensor(shape=[3], dtype=Int64, value= [ 1, -7, -4])
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.amin(input, axis=1, keepdims=True)
Tensor(shape=[3, 1], dtype=Int64, value=
[[ 1],
 [-7],
 [-4]])
>>>
>>> # case 4: Use "where" to include only specific elements in computing the minimum.
>>> where = mindspore.tensor([[1, 0, 1, 0],
...                           [0, 0, 1, 1],

```

(下页继续)

(续上页)

```

...                               [1, 1, 1, 0]], dtype=mindspore.bool_)
>>> mindspore.ops.amin(input, axis=1, keepdims=True, initial=0, where=where)
Tensor(shape=[3, 1], dtype=Int64, value=
[[ 0],
 [-2],
 [-4]])
>>>
>>> # case 5: The shape of "where" must be broadcast compatible with input.
>>> where = mindspore.tensor([[False],
...                           [False],
...                           [False]])
>>> mindspore.ops.amin(input, axis=0, keepdims=True, initial=0, where=where)
Tensor(shape=[1, 4], dtype=Int64, value=
[[0, 0, 0, 0]])

```

mindspore.ops.aminmax

`mindspore.ops.aminmax(input, *, axis=0, keepdims=False)`

返回 `tensor` 在指定轴上的最小值和最大值。

参数:

- **input** (Tensor) - 输入 `tensor`。

关键字参数:

- **axis** (int, 可选) - 指定计算的轴。如果为 `None`，计算 `input` 中的所有元素。默认 0。
- **keepdims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

两个 `tensor` 组成的 `tuple(min, max)`。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>>
>>> # case 1: By default, compute along axis 0.
>>> mindspore.ops.aminmax(input)
(Tensor(shape=[4], dtype=Int64, value= [5, 1, 3, 4]),
 Tensor(shape=[4], dtype=Int64, value= [9, 3, 7, 6]))
>>>
>>> # case 2: Disregard NaN (Not a Number) values present in the input during computation.
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, float('nan')]])
>>> mindspore.ops.aminmax(input, axis=None)
(Tensor(shape=[], dtype=Float32, value= 1),
 Tensor(shape=[], dtype=Float32, value= 9))
>>>

```

(下页继续)

(续上页)

```
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.aminmax(input, axis=None, keepdims=True)
(Tensor(shape=[1, 1], dtype=Float32, value=
[[ 1.00000000e+00]]),
 Tensor(shape=[1, 1], dtype=Float32, value=
[[ 9.00000000e+00]]))
```

mindspore.ops.any

`mindspore.ops.any(input, axis=None, keep_dims=False)`

检查指定轴上是否含有 *True*。

参数：

- **input** (Tensor) - 输入 tensor。
- **axis** (Union[int, tuple(int), list(int), Tensor], 可选) - 指定轴。如果为 None，检查所有元素。默认 None。
- **keep_dims** (bool, 可选) - 输出 tensor 是否保留维度。默认 False。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: By default, mindspore.ops.any tests along all the axes.
>>> mindspore.ops.any(input)
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 2: Reduces a dimension along axis 1, with keep_dims False.
>>> mindspore.ops.any(input, axis=1)
Tensor(shape=[2], dtype=Bool, value= [ True,  True])
>>>
>>> # case 3: Reduces a dimension along axis (0, 1), with keep_dims False.
>>> mindspore.ops.any(input, axis=(0,1))
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 4: Reduces a dimension along axis [0, 1], with keep_dims True.
>>> mindspore.ops.any(input, axis=[0,1], keep_dims=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[ True]])
```

mindspore.ops.argmax

`mindspore.ops.argmax(input, dim=None, keepdim=False)`

返回 `tensor` 在指定维度上的最大值索引。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **dim** (Union[int, None], 可选) - 指定维度。如果为 `None`，则计算 `input` 中所有元素的最大值索引。默认 `None`。
- **keepdim** (bool, 可选) - 是否保留输出 `tensor` 的维度。默认 `False`。

返回:

Tensor, 包含最大值的索引。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: By default, compute the maximum indice of all elements.
>>> mindspore.ops.argmax(input)
Tensor(shape=[], dtype=Int64, value= 0)
>>>
>>> # case 2: Compute maximum indice along dim 1.
>>> mindspore.ops.argmax(input, dim=1)
Tensor(shape=[3], dtype=Int64, value= [0, 2, 0])
>>>
>>> # case 3: If keepdim=True, the output shape will be same of that of the input.
>>> mindspore.ops.argmax(input, dim=1, keepdim=True)
Tensor(shape=[3, 1], dtype=Int64, value=
[[0],
 [2],
 [0]])
```

mindspore.ops.argmin

`mindspore.ops.argmin(input, axis=None, keepdims=False)`

返回 `tensor` 在指定轴上的最大值索引。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **axis** (Union[int, None], 可选) - 指定轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **keepdims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[2, 5, 1, 6],
...                           [3, -7, -2, 4],
...                           [8, -4, 1, -3]])
>>> # case 1: By default, compute the minimum indice of all elements.
>>> mindspore.ops.argmin(input)
Tensor(shape=[], dtype=Int32, value= 5)
>>>
>>> # case 2: Compute the minimum indices along axis 1.
>>> mindspore.ops.argmin(input, axis=1)
Tensor(shape=[3], dtype=Int32, value= [2, 1, 1])
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.argmin(input, axis=1, keepdims=True)
Tensor(shape=[3, 1], dtype=Int32, value=
[[2],
 [1],
 [1]])
```

mindspore.ops.cummax

`mindspore.ops.cummax(input, axis)`

返回 **tensor** 在指定轴上的累积最大值及其索引。

$$y_i = \max(x_1, x_2, \dots, x_i)$$

参数:

- **input** (Tensor) - 输入 tensor。
- **axis** (int) - 指定计算的轴。

返回:

两个 **tensor** 组成的 **tuple**(max, max_indices)。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[3, 4, 6, 10],
...                           [1, 6, 7, 9],
...                           [4, 3, 8, 7],
...                           [1, 3, 7, 9]])
>>> mindspore.ops.cummax(input, axis=0)
(Tensor(shape=[4, 4], dtype=Int64, value=
[[ 3,  4,  6, 10],
 [ 3,  6,  7, 10],
 [ 4,  6,  8, 10],
 [ 4,  6,  8, 10]]),
Tensor(shape=[4, 4], dtype=Int64, value=
[[0, 0, 0, 0],
 [0, 1, 1, 0],
```

(下页继续)

(续上页)

```
[2, 1, 2, 0],
[2, 1, 2, 0]])
```

mindspore.ops.cummin

`mindspore.ops.cummin(input, axis)`

返回 `tensor` 在指定轴上的累积最小值及其索引。

$$y_i = \min(x_1, x_2, \dots, x_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。
- **axis** (int) - 指定计算的轴。

返回:

两个 `tensor` 组成的 `tuple(min, min_indices)`

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import Tensor, ops
>>> import mindspore
>>> a = Tensor([-0.2284, -0.6628, 0.0975, 0.2680, -1.3298, -0.4220], mindspore.float32)
>>> output = ops.cummin(a, axis=0)
>>> print(output[0])
[-0.2284 -0.6628 -0.6628 -0.6628 -1.3298 -1.3298]
>>> print(output[1])
[0 1 1 1 4 4]
```

mindspore.ops.cumprod

`mindspore.ops.cumprod(input, dim, dtype=None)`

返回 `tensor` 在指定维度上累积的元素乘积。

$$y_i = x_1 * x_2 * x_3 * \dots * x_i$$

参数:

- **input** (Tensor) - 输入 `tensor`。
- **dim** (int) - 指定维度。
- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 `None`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3],
...                             [4, 5, 6]])
>>> mindspore.ops.cumprod(input, dim=0)
Tensor(shape=[2, 3], dtype=Int64, value=
[[ 1,  2,  3],
 [ 4, 10, 18]])
>>> mindspore.ops.cumprod(input, dim=1)
Tensor(shape=[2, 3], dtype=Int64, value=
[[ 1,  2,  6],
 [ 4, 20, 120]])
```

mindspore.ops.cumsum

`mindspore.ops.cumsum(x, axis, dtype=None)`

返回 `tensor` 在指定轴上累积的元素和。

$$y_i = x_1 + x_2 + x_3 + \dots + x_i$$

参数:

- **x** (Tensor) - 输入 `tensor`。
- **axis** (int) - 指定计算的轴。
- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 `None`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3],
...                             [4, 5, 6]], mindspore.int32)
>>> mindspore.ops.cumsum(input, axis=0)
Tensor(shape=[2, 3], dtype=Int32, value=
[[1, 2, 3],
 [5, 7, 9]])
>>> mindspore.ops.cumsum(input, axis=1)
Tensor(shape=[2, 3], dtype=Int32, value=
[[ 1,  3,  6],
 [ 4,  9, 15]])
```

mindspore.ops.histc

`mindspore.ops.histc(input, bins=100, min=0., max=0.)`

计算 `tensor` 的直方图。

元素被分类到 `min` 和 `max` 之间的等宽箱中。如果 `min` 和 `max` 均为 0，则使用数据的最小值和最大值。

低于最小值和高于最大值的元素将被忽略。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **bins** (int, 可选) - 直方图箱的数量。默认 100。
- **min** (int, float, 可选) - 直方图数据范围的最小值。默认 0.。
- **max** (int, float, 可选) - 直方图数据范围的最大值。默认 0.。

返回：

Tensor

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1., 1, 2, 2, 2, 5, 8])
>>> output = mindspore.ops.histc(input, bins=5, min=0, max=9)
>>> print(output)
[2. 3. 1. 0. 1.]
```

mindspore.ops.logcumsumexp

`mindspore.ops.logcumsumexp(input, axis)`

计算 `tensor` 在指定维度上的累积指数和的对数。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **axis** (int) - 指定计算轴。

返回：

Tensor

支持平台：

Ascend CPU GPU

样例：


```
>>> import mindspore
>>> input = mindspore.tensor([[9., 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: Compute the log of cumulative summed exponential along dim 0.
>>> output = mindspore.ops.logcumsumexp(input, 0)
>>> print(output)
[[9.      3.      4.      5.      ]
 [9.01815  3.3132617 7.0485873 5.3132615]
 [9.326563  3.407606  7.065884  6.407606 ]]
>>>
>>> # case 2: Compute the log of cumulative summed exponential along dim 1.
>>> output = mindspore.ops.logcumsumexp(input, 1)
>>> print(output)
[[9.      9.002476  9.009174  9.02716  ]
 [5.      5.0485873 7.1328454 7.175515 ]
 [8.      8.000912  8.007621  8.133643 ]]
```

`mindspore.ops.logsumexp`

`mindspore.ops.logsumexp(input, dim, keepdim=False)`

计算 `tensor` 在指定维度上的指数和的对数。

$$\text{logsumexp}(\text{input}) = \log\left(\sum(e^{\text{input}-\text{input}_{\max}})\right) + \text{input}_{\max}$$

参数:

- **input** (Tensor) - 输入 `tensor`。
- **dim** (Union[int, tuple(int), list(int)]) - 指定维度。如果为 `()`，计算 `input` 中的所有元素。
- **keepdim** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[9., 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: By default, compute the log of summed exponential of all elements.
>>> output = mindspore.ops.logsumexp(input, ())
>>> print(output)
9.475807
>>>
>>> # case 2: Compute the log of summed exponential along dim 0.
>>> output = mindspore.ops.logsumexp(input, 0)
>>> print(output)
[9.326562  3.4076054 7.065884  6.4076056]
>>>
>>> # case 3: If keepdim=True, the output shape will be same of that of the input.
```

(下页继续)

(续上页)

```
>>> output = mindspore.ops.logsumexp(input, 1, True)
>>> print(output)
[[9.02716 ]
 [7.175515]
 [8.133643]]
```

mindspore.ops.max

`mindspore.ops.max(input, axis=None, keepdims=False, *, initial=None, where=None)`

返回 `tensor` 在指定轴上的最大值及其索引。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **axis** (int) - 指定轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **keepdims** (bool) - 输出 `tensor` 是否保留维度。默认 `False`。

关键字参数:

- **initial** (scalar, 可选) - 最大值的初始值。默认 `None`。
- **where** (Tensor[bool], 可选) - 指定计算最大值的范围，该 `tensor` 的 `shape` 须可被广播到 `input` 的 `shape` 上。必须指定 `initial` 值。默认 `None`，表示计算全部元素。

返回:

两个 `tensor` 组成的 `tuple(max, max_indices)`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: By default, compute the maximum of all elements.
>>> mindspore.ops.max(input)
(Tensor(shape=[], dtype=Int64, value= 9),
 Tensor(shape=[], dtype=Int64, value= 0))
>>>
>>> # case 2: Compute maximum along axis 1.
>>> mindspore.ops.max(input, axis=1)
(Tensor(shape=[3], dtype=Int64, value= [9, 7, 8]),
 Tensor(shape=[3], dtype=Int64, value= [0, 2, 0]))
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.max(input, axis=1, keepdims=True)
(Tensor(shape=[3, 1], dtype=Int64, value=
 [[9],
 [7],
 [8]]),
 Tensor(shape=[3, 1], dtype=Int64, value=
 [[0],
 [2],
```

(下页继续)

(续上页)

```

[0]))
>>>
>>> # case 4: Use "where" to include only specific elements in computing the maximum.
>>> where = mindspore.tensor([[0, 0, 1, 0],
...                           [0, 0, 1, 1],
...                           [1, 1, 1, 0]], dtype=mindspore.bool_)
>>> mindspore.ops.max(input, axis=1, keepdims=True, initial=0, where=where)
(Tensor(shape=[3, 1], dtype=Int64, value=
[[4],
 [7],
 [8]]),
 Tensor(shape=[3, 1], dtype=Int64, value=
[[2],
 [2],
 [0]]))
>>>
>>> # case 5: The shape of "where" must be broadcast compatible with input.
>>> where = mindspore.tensor([[False],
...                           [False],
...                           [False]])
>>> mindspore.ops.max(input, axis=0, keepdims=True, initial=0, where=where)
(Tensor(shape=[1, 4], dtype=Int64, value=
[[0, 0, 0, 0]]),
 Tensor(shape=[1, 4], dtype=Int64, value=
[[0, 0, 0, 0]]))

```

mindspore.ops.mean

`mindspore.ops.mean(x, axis=None, keep_dims=False)`

计算 `tensor` 在指定轴上的均值。

参数：

- **x** (Tensor[Number]) - 输入 `tensor`。
- **axis** (Union[int, tuple(int), list(int), Tensor]) - 指定轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **keep_dims** (bool) - 输出 `tensor` 是否保留维度。默认 `False`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: By default, compute the mean of all elements.
>>> mindspore.ops.mean(input)
Tensor(shape=[], dtype=Int64, value= 4)
>>>
>>> # case 2: Compute the mean along axis 1.

```

(下页继续)

(续上页)

```
>>> mindspore.ops.mean(input, axis=1)
Tensor(shape=[3], dtype=Int64, value= [5, 4, 4])
>>>
>>> # case 3: If keep_dims=True, the output shape will be same of that of the input.
>>> mindspore.ops.mean(input, axis=1, keep_dims=True)
Tensor(shape=[3, 1], dtype=Int64, value=
[[5],
 [4],
 [4]])
```

mindspore.ops.median

`mindspore.ops.median(input, axis=-1, keepdims=False)`

返回 `tensor` 在指定轴上的中位数及其索引。

警告:

- 如果 `input` 的中值不唯一，则 `indices` 不一定对应第一个出现的中值。该接口的具体实现方式和后端类型相关，CPU 和 GPU 的返回值可能不相同。

参数:

- **input** (Tensor) - 输入 `tensor`。
- **axis** (int, 可选) - 指定计算的轴。默认 `-1`。
- **keepdims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回:

两个 `tensor` 组成的 `tuple(median, median_indices)`

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: By default, compute the median along axis -1.
>>> mindspore.ops.median(input)
(Tensor(shape=[3], dtype=Int64, value= [4, 4, 3]),
 Tensor(shape=[3], dtype=Int64, value= [2, 3, 2]))
>>>
>>> # case 2: Compute the median along axis 0.
>>> mindspore.ops.median(input, axis=0)
(Tensor(shape=[4], dtype=Int64, value= [8, 2, 4, 5]),
 Tensor(shape=[4], dtype=Int64, value= [2, 1, 0, 0]))
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.median(input, axis=0, keepdims=True)
(Tensor(shape=[1, 4], dtype=Int64, value=
```

(下页继续)

(续上页)

```
[[8, 2, 4, 5]]),
Tensor(shape=[1, 4], dtype=Int64, value=
[[2, 1, 0, 0]])
```

mindspore.ops.min

`mindspore.ops.min(input, axis=None, keepdims=False, *, initial=None, where=None)`

返回 `tensor` 在指定轴上的最小值及其索引。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **axis** (int) - 指定轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **keepdims** (bool) - 输出 `tensor` 是否保留维度。默认 `False`。

关键字参数：

- **initial** (scalar, 可选) - 最小值的初始值。默认 `None`。
- **where** (Tensor[bool], 可选) - 指定计算最小值的范围，该 `tensor` 的 `shape` 须可被广播到 `input` 的 `shape` 上。必须指定 `initial` 值。默认 `None`，表示计算全部元素。

返回：

两个 `tensor` 组成的 `tuple(min, min_indices)`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[2, 5, 1, 6],
...                           [3, -7, -2, 4],
...                           [8, -4, 1, -3]])
>>> # case 1: By default, compute the minimum of all elements.
>>> mindspore.ops.min(input)
(Tensor(shape=[], dtype=Int64, value= -7),
 Tensor(shape=[], dtype=Int64, value= 0))
>>>
>>> # case 2: Compute minimum along axis 1.
>>> mindspore.ops.min(input, axis=1)
(Tensor(shape=[3], dtype=Int64, value= [ 1, -7, -4]),
 Tensor(shape=[3], dtype=Int64, value= [2, 1, 1]))
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.min(input, axis=1, keepdims=True)
(Tensor(shape=[3, 1], dtype=Int64, value=
[[ 1],
 [-7],
 [-4]]),
 Tensor(shape=[3, 1], dtype=Int64, value=
[[2],
 [1],
 [1]]))
>>>
```

(下页继续)

(续上页)

```

>>> # case 4: Use "where" to include only specific elements in computing the minimum.
>>> where = mindspore.tensor([[1, 0, 1, 0],
...                           [0, 0, 1, 1],
...                           [1, 1, 1, 0]], dtype=mindspore.bool_)
>>> mindspore.ops.min(input, axis=1, keepdims=True, initial=0, where=where)
(Tensor(shape=[3, 1], dtype=Int64, value=
[[ 0],
 [-2],
 [-4]]),
Tensor(shape=[3, 1], dtype=Int64, value=
[[0],
 [2],
 [1]]))
>>>
>>> # case 5: The shape of "where" must be broadcast compatible with input.
>>> where = mindspore.tensor([[False],
...                           [False],
...                           [False]])
>>> mindspore.ops.min(input, axis=0, keepdims=True, initial=0, where=where)
(Tensor(shape=[1, 4], dtype=Int64, value=
[[0, 0, 0, 0]]),
Tensor(shape=[1, 4], dtype=Int64, value=
[[0, 0, 0, 0]]))

```

mindspore.ops.norm

`mindspore.ops.norm(A, ord=None, dim=None, keepdim=False, *, dtype=None)`

计算 tensor 的矩阵范数或向量范数。

`ord` 为 norm 的计算模式。支持下列 norm 模式。

<i>ord</i>	矩阵范数	向量范数
<i>None</i> (默认值)	Frobenius norm	2-norm (参考最下方公式)
'fro'	Frobenius norm	不支持
'nuc'	nuclear norm	不支持
<i>inf</i>	$\max(\text{sum}(\text{abs}(x), \text{dim} = 1))$	$\max(\text{abs}(x))$
<i>-inf</i>	$\min(\text{sum}(\text{abs}(x), \text{dim} = 1))$	$\min(\text{abs}(x))$
0	不支持	$\text{sum}(x \neq 0)$
1	$\max(\text{sum}(\text{abs}(x), \text{dim} = 0))$	参考最下方公式
-1	$\min(\text{sum}(\text{abs}(x), \text{dim} = 0))$	参考最下方公式
2	最大奇异值	参考最下方公式
-2	最小奇异值	参考最下方公式
其余 int 或 float 值	不支持	$\text{sum}(\text{abs}(x)^{\text{ord}})^{(1/\text{ord})}$

参数:

- **A** (Tensor) - 输入 tensor。
- **ord** (Union[int, float, inf, -inf, 'fro', 'nuc'], 可选) - 指定要采用的范数类型。默认 `None`。
- **dim** (Union[int, Tuple(int)], 可选) - 指定计算维度。默认 `None`。
 - 当 `dim` 为 int 时, 计算向量范数。
 - 当 `dim` 为一个二元组时, 计算矩阵范数。

- 当 *dim* 为 `None` 且 *ord* 为 `None`，展平 *A* 为一维 `tensor` 并计算向量的 2-范数。
- 当 *dim* 为 `None` 且 *ord* 不为 `None`，*A* 必须为一维或者二维。

- **keepdim** (bool) - 输出 `tensor` 是否保留维度。默认 `False`。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 指定数据类型。如果设置此参数，则会在计算前将 *A* 转换为指定的类型。默认 `None`。

返回：

Tensor

说明：

- 当前暂不支持复数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # Vector norms:
>>> A = mindspore.tensor([3., 4., 12.])
>>> mindspore.ops.norm(A)
Tensor(shape=[], dtype=Float32, value= 13)
>>> mindspore.ops.norm(A, ord=1)
Tensor(shape=[], dtype=Float32, value= 19)
>>> mindspore.ops.norm(A, ord=0)
Tensor(shape=[], dtype=Float32, value= 3)
>>>
>>> # Matrix norms:
>>> A = mindspore.tensor([[1., 2., 3.],
...                       [4., 5., 7.]])
>>> mindspore.ops.norm(A) # Frobenius norm
Tensor(shape=[], dtype=Float32, value= 10.198)
>>> mindspore.ops.norm(A, ord='nuc') # nuclear norm
Tensor(shape=[], dtype=Float32, value= 10.7625)
>>> mindspore.ops.norm(A, ord=1) # 1-norm
Tensor(shape=[], dtype=Float32, value= 10)
>>>
>>> # Batched vector norm:
>>> mindspore.ops.norm(A, dim=1)
Tensor(shape=[2], dtype=Float32, value= [ 3.74165726e+00,  9.48683262e+00])
```

mindspore.ops.prod

`mindspore.ops.prod(input, axis=None, keep_dims=False, dtype=None)`

返回 `tensor` 在指定轴上的乘积。

参数：

- **input** (`Tensor[Number]`) - 输入 `tensor`。
- **axis** (`Union[int, tuple(int), list(int), Tensor]`) - 指定计算轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **keep_dims** (`bool`) - 输出 `tensor` 是否保留维度。默认 `False`。
- **dtype** (`mindspore.dtype`) - 指定数据类型。默认 `None`。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[9, 3, 4, 5],
...                           [5, 2, 7, 4],
...                           [8, 1, 3, 6]])
>>> # case 1: By default, compute the product of all elements.
>>> mindspore.ops.prod(input)
Tensor(shape=[], dtype=Int64, value= 21772800)
>>>
>>> # case 2: Compute the product along axis 1.
>>> mindspore.ops.prod(input, axis=1)
Tensor(shape=[3], dtype=Int64, value= [540, 280, 144])
>>>
>>> # case 3: If keep_dims=True, the output shape will be same of that of the input.
>>> mindspore.ops.prod(input, axis=1, keep_dims=True)
Tensor(shape=[3, 1], dtype=Int64, value=
[[540],
 [280],
 [144]])
```

mindspore.ops.std

`mindspore.ops.std(input, axis=None, ddof=0, keepdims=False)`

计算 `Tensor` 在指定轴上的标准差。

参数：

- **input** (`Tensor[Number]`) - 输入 `Tensor`。
- **axis** (`Union[int, tuple(int)]`，可选) - 指定轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **ddof** (`Union[int, bool]`，可选) - δ 自由度。默认 `0`。
 - 如果为整数，计算中使用的除数是 $N - ddof$ ，其中 N 表示元素的数量。
 - 如果为 `bool` 值，`True` 与 `False` 分别对应 `ddof` 为整数时的 `1` 与 `0`。

- 如果取值为 0、1、True 或 False，支持的平台只有 *Ascend* 和 *CPU*。其他情况下，支持平台是 *Ascend*、*GPU* 和 *CPU*。

- **keepdims** (bool, 可选) - 输出 Tensor 是否保留维度。默认 False。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1., 3, 4, 2],
...                           [4, 2, 5, 3],
...                           [5, 4, 2, 3]])
>>> # case 1: By default, compute the standard deviation of all elements.
>>> output = mindspore.ops.std(input)
>>> print(output)
1.2133516
>>>
>>> # case 2: Compute the standard deviation along axis 0.
>>> output = mindspore.ops.std(input, axis=0)
>>> print(output)
[1.6996732 0.8164966 1.2472192 0.4714045]
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> output = mindspore.ops.std(input, axis=0, keepdims=True)
>>> print(output)
[[1.6996732 0.8164966 1.2472192 0.4714045]]
>>>
>>> # case 4: If ddof=1:
>>> output = mindspore.ops.std(input, axis=0, keepdims=True, ddof=1)
>>> print(output)
[[2.081666 1. 1.5275253 0.57735026]]
>>>
>>> # case 5: If ddof=True, same as ddof=1:
>>> output = mindspore.ops.std(input, axis=0, keepdims=True, ddof=True)
>>> print(output)
[[2.081666 1. 1.5275253 0.57735026]]
>>>
>>> # case 6: If ddof=False, same as ddof=0:
>>> output = mindspore.ops.std(input, axis=0, keepdims=True, ddof=False)
>>> print(output)
[[1.6996732 0.8164966 1.2472192 0.4714045]]
```

mindspore.ops.std_mean

`mindspore.ops.std_mean(input, axis=None, ddof=0, keepdims=False)`

计算 tensor 在指定轴上的标准差及平均值。

参数：

- **input** (Tensor[Number]) - 输入 tensor。
- **axis** (Union[int, tuple(int)], 可选) - 指定轴。如果为 None，计算 *input* 中的所有元素。默认 None。
- **ddof** (Union[int, bool], 可选) - δ 自由度。默认 0。
 - 如果为整数，计算中使用的除数是 $N - ddof$ ，其中 N 表示元素的数量。
 - 如果为 bool 值，True 与 False 分别对应 ddof 为整数时的 1 与 0。
 - 如果取值为 0、1、True 或 False，支持的平台只有 *Ascend* 和 *CPU*。其他情况下，支持平台是 *Ascend*、*GPU* 和 *CPU*。
- **keepdims** (bool, 可选) - 输出 tensor 是否保留维度。默认 False。

返回：

两个 tensor 组成的 tuple(std, mean)。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1., 3, 4, 2],
...                           [4, 2, 5, 3],
...                           [5, 4, 2, 3]])
>>> # case 1: By default, compute the standard deviation and mean of all elements.
>>> mindspore.ops.std_mean(input)
(Tensor(shape=[], dtype=Float32, value= 1.21335),
 Tensor(shape=[], dtype=Float32, value= 3.16667))
>>>
>>> # case 2: Compute the standard deviation and mean along axis 0.
>>> mindspore.ops.std_mean(input, axis=0)
(Tensor(shape=[4], dtype=Float32, value= [ 1.69967318e+00,  8.16496611e-01,  1.24721920e+00, 4.71404493e-01]),
 Tensor(shape=[4], dtype=Float32, value= [ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00, 2.66666675e+00]))
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> mindspore.ops.std_mean(input, axis=0, keepdims=True)
(Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 1.69967318e+00,  8.16496611e-01,  1.24721920e+00,  4.71404493e-01]]),
 Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))
>>>
>>> # case 4: If ddof=1:
>>> mindspore.ops.std_mean(input, axis=0, keepdims=True, ddof=1)
(Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 2.08166599e+00,  1.00000000e+00,  1.52752531e+00,  5.77350259e-01]]),
 Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))
>>>
```

(下页继续)

(续上页)

```
>>> # case 5: If ddof=True, same as ddof=1:
>>> mindspore.ops.std_mean(input, axis=0, keepdims=True, ddof=True)
(Tensor(shape=[1, 4], dtype=Float32, value=
[[ 2.08166599e+00,  1.00000000e+00,  1.52752531e+00,  5.77350259e-01]]),
Tensor(shape=[1, 4], dtype=Float32, value=
[[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))
>>>
>>> # case 6: If ddof=False, same as ddof=0:
>>> mindspore.ops.std_mean(input, axis=0, keepdims=True, ddof=False)
(Tensor(shape=[1, 4], dtype=Float32, value=
[[ 1.69967318e+00,  8.16496611e-01,  1.24721920e+00,  4.71404493e-01]]),
Tensor(shape=[1, 4], dtype=Float32, value=
[[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))
```

mindspore.ops.var

`mindspore.ops.var(input, axis=None, ddof=0, keepdims=False)`

计算 tensor 在指定轴上的方差。

参数:

- **input** (Tensor[Number]) - 输入 tensor。
- **axis** (Union[int, tuple(int)], 可选) - 指定轴。如果为 None，计算 *input* 中的所有元素。默认 None。
- **ddof** (Union[int, bool], 可选) - δ 自由度。默认 0。
 - 如果为整数，计算中使用的除数是 $N - ddof$ ，其中 N 表示元素的数量。
 - 如果为 bool 值，True 与 False 分别对应 ddof 为整数时的 1 与 0。
 - 如果取值为 0、1、True 或 False，支持的平台只有 Ascend 和 CPU。其他情况下，支持平台是 Ascend、GPU 和 CPU。
- **keepdims** (bool, 可选) - 输出 tensor 是否保留维度。默认 False。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[1., 3, 4, 2],
...                           [4, 2, 5, 3],
...                           [5, 4, 2, 3]])
>>> # case 1: By default, compute the variance of all elements.
>>> output = mindspore.ops.var(input)
>>> print(output)
1.4722221
>>>
>>> # case 2: Compute the variance along axis 0.
>>> output = mindspore.ops.var(input, axis=0)
>>> print(output)
[2.8888888 0.6666667 1.5555557 0.2222222]
```

(下页继续)

(续上页)

```

>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> output = mindspore.ops.var(input, axis=0, keepdims=True)
>>> print(output)
[[2.8888888 0.6666667 1.5555557 0.2222222]]
>>>
>>> # case 4: If ddof=1:
>>> output = mindspore.ops.var(input, axis=0, keepdims=True, ddof=1)
>>> print(output)
[[4.3333335 1.          2.3333335 0.3333333]]
>>>
>>> # case 5: If ddof=True, same as ddof=1:
>>> output = mindspore.ops.var(input, axis=0, keepdims=True, ddof=True)
>>> print(output)
[[4.3333335 1.          2.3333335 0.3333333]]
>>>
>>> # case 6: If ddof=False, same as ddof=0:
>>> output = mindspore.ops.var(input, axis=0, keepdims=True, ddof=False)
>>> print(output)
[[2.8888888 0.6666667 1.5555557 0.2222222]]

```

mindspore.ops.var_mean

`mindspore.ops.var_mean(input, axis=None, ddof=0, keepdims=False)`

计算 `tensor` 在指定轴上的方差及平均值。

参数：

- **input** (Tensor[Number]) - 输入 `tensor`。
- **axis** (Union[int, tuple(int)], 可选) - 指定轴。如果为 `None`，计算 `input` 中的所有元素。默认 `None`。
- **ddof** (Union[int, bool], 可选) - δ 自由度。默认 0。
 - 如果为整数，计算中使用的除数是 $N - ddof$ ，其中 N 表示元素的数量。
 - 如果为 `bool` 值，`True` 与 `False` 分别对应 `ddof` 为整数时的 1 与 0。
 - 如果取值为 0、1、`True` 或 `False`，支持的平台只有 *Ascend* 和 *CPU*。其他情况下，支持平台是 *Ascend*、*GPU* 和 *CPU*。
- **keepdims** (bool, 可选) - 输出 `tensor` 是否保留维度。默认 `False`。

返回：

两个 `tensor` 组成的 `tuple(var, mean)`。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> input = mindspore.tensor([[1., 3, 4, 2],
...                           [4, 2, 5, 3],
...                           [5, 4, 2, 3]])
>>> # case 1: By default, compute the variance and mean of all elements.
>>> mindspore.ops.var_mean(input)

```

(下页继续)

(续上页)

```

(Tensor(shape=[], dtype=Float32, value= 1.47222),
 Tensor(shape=[], dtype=Float32, value= 3.16667))
>>>
>>> # case 2: Compute the variance and mean along axis 0.
>>> output = mindspore.ops.var_mean(input, axis=0)
(Tensor(shape=[4], dtype=Float32, value= [ 2.88888884e+00,  6.66666687e-01,  1.55555570e+00,  2.22222194e-01]),
 Tensor(shape=[4], dtype=Float32, value= [ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]))
>>>
>>> # case 3: If keepdims=True, the output shape will be same of that of the input.
>>> output = mindspore.ops.var_mean(input, axis=0, keepdims=True)
(Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 2.88888884e+00,  6.66666687e-01,  1.55555570e+00,  2.22222194e-01]]),
 Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))
>>>
>>> # case 4: If ddof=1:
>>> output = mindspore.ops.var_mean(input, axis=0, keepdims=True, ddof=1)
(Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 4.33333349e+00,  1.00000000e+00,  2.33333349e+00,  3.33333313e-01]]),
 Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))
>>>
>>> # case 5: If ddof=True, same as ddof=1:
>>> output = mindspore.ops.var_mean(input, axis=0, keepdims=True, ddof=True)
(Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 4.33333349e+00,  1.00000000e+00,  2.33333349e+00,  3.33333313e-01]]),
 Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))
>>>
>>> # case 6: If ddof=False, same as ddof=0:
>>> output = mindspore.ops.var_mean(input, axis=0, keepdims=True, ddof=False)
(Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 2.88888884e+00,  6.66666687e-01,  1.55555570e+00,  2.22222194e-01]]),
 Tensor(shape=[1, 4], dtype=Float32, value=
 [[ 3.33333325e+00,  3.00000000e+00,  3.66666675e+00,  2.66666675e+00]]))

```

2.2.3 比较函数

接口名	概述	支持平台
<code>mindspore.ops.argsort</code>	返回按指定轴对 <code>tensor</code> 进行排序后的索引。	Ascend GPU CPU
<code>mindspore.ops.approximate_equal</code>	返回一个布尔型 <code>tensor</code> , 表示两个 <code>tensor</code> 在容忍度内是否逐元素相等。	Ascend GPU CPU
<code>mindspore.ops.bucketize</code>	返回输入 <code>tensor</code> 中每个元素所属桶的索引。	Ascend GPU CPU
<code>mindspore.ops.eq</code>	逐元素计算两个输入是否相等。	Ascend GPU CPU
<code>mindspore.ops.equal</code>	逐元素计算两个输入是否相等。	Ascend GPU CPU
<code>mindspore.ops.ge</code>	逐元素计算 <code>input >= other</code> 的值。	Ascend GPU CPU
<code>mindspore.ops.greater</code>	逐元素计算 <code>input > other</code> 的值。	Ascend GPU CPU
<code>mindspore.ops.greater_equal</code>	逐元素计算 <code>input >= other</code> 的值。	Ascend GPU CPU
<code>mindspore.ops.gt</code>	逐元素计算 <code>input > other</code> 的值。	Ascend GPU CPU

下页继续

表 8 - 续上页

<code>mindspore.ops.intopk</code>	返回第二个输入 tensor 中的元素是否存在于第一个输入 tensor 的前 k 个元素中。	Ascend GPU CPU
<code>mindspore.ops.isclose</code>	返回一个布尔型 tensor , 表示两个 tensor 在容忍度内是否逐元素相等。	Ascend GPU CPU
<code>mindspore.ops.isfinite</code>	返回一个布尔型 tensor , 指示哪些元素是有限数。	Ascend GPU CPU
<code>mindspore.ops.isinf</code>	返回一个布尔型 tensor , 指示哪些元素是正负无穷数。	Ascend CPU GPU
<code>mindspore.ops.isnan</code>	返回一个布尔型 tensor , 指示哪些元素是 NaN。	Ascend GPU CPU
<code>mindspore.ops.isneginf</code>	返回一个布尔型 tensor , 指示哪些元素是负无穷数。	Ascend GPU CPU
<code>mindspore.ops.isposinf</code>	返回一个布尔型 tensor , 指示哪些元素是正无穷数。	Ascend GPU CPU
<code>mindspore.ops.isreal</code>	返回一个布尔型 tensor , 指示哪些元素是实数。	GPU CPU
<code>mindspore.ops.is_complex</code>	返回一个布尔型 tensor , 指示哪些元素是复数。	Ascend GPU CPU
<code>mindspore.ops.is_floating_point</code>	如果 tensor 的数据类型是浮点数, 则返回 True , 否则返回 False 。	Ascend GPU CPU
<code>mindspore.ops.le</code>	逐元素计算 $input \leq other$ 的值。	Ascend GPU CPU
<code>mindspore.ops.less</code>	逐元素计算 $input < other$ 的值。	Ascend GPU CPU
<code>mindspore.ops.less_equal</code>	逐元素计算 $input \leq other$ 的值。	Ascend GPU CPU
<code>mindspore.ops.lt</code>	<code>mindspore.ops.less()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.maximum</code>	逐元素计算两个输入 tensor 中最大值。	Ascend GPU CPU
<code>mindspore.ops.minimum</code>	逐元素计算两个输入 tensor 的最小值。	Ascend GPU CPU
<code>mindspore.ops.msort</code>	返回沿第一个维度对输入 tensor 进行升序排序后的 tensor 。	Ascend GPU CPU
<code>mindspore.ops.ne</code>	逐元素计算两个输入是否不相等。	Ascend GPU CPU
<code>mindspore.ops.not_equal</code>	<code>mindspore.ops.ne()</code> 的别名。	Ascend GPU CPU
<code>mindspore.ops.searchsorted</code>	返回元素能够插入输入 tensor 的位置索引, 以维持原 tensor 的递增顺序。	Ascend GPU CPU
<code>mindspore.ops.topk</code>	按指定维度返回前 k 个最大或最小元素及其对应索引。	Ascend GPU CPU

mindspore.ops.argsort

`mindspore.ops.argsort(input, axis=-1, descending=False)`

返回按指定轴对 **tensor** 进行排序后的索引。

说明: 当前 Ascend 后端只支持对最后一维进行排序。

参数:

- **input** (Tensor) - 输入 **tensor**。
- **axis** (int) - 指定轴。默认 -1。
- **descending** (bool) - 指定排序 (升序或降序)。默认 **False**。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: 1-dimensional sort
>>> input = mindspore.tensor([1, 3, 5, 4, 2, 1])
>>> mindspore.ops.argsort(input)
Tensor(shape=[6], dtype=Int32, value= [0, 5, 4, 1, 3, 2])
>>>
>>> # case 2: multi-dimensional sort
>>> input = mindspore.tensor([[2, 1, 3],
...                           [6, 4, 3]])
>>> mindspore.ops.argsort(input, axis=1)
Tensor(shape=[2, 3], dtype=Int32, value=
[[1, 0, 2],
 [2, 1, 0]])
>>> mindspore.ops.argsort(input, axis=1, descending=True)
Tensor(shape=[2, 3], dtype=Int32, value=
[[2, 0, 1],
 [0, 1, 2]])
```

`mindspore.ops.approximate_equal`

`mindspore.ops.approximate_equal(x, y, tolerance=1e-5)`

返回一个布尔型 `tensor`，表示两个 `tensor` 在容忍度内是否逐元素相等。

支持隐式类型转换、类型提升。

数学公式为：

$$out_i = \begin{cases} \text{if } |x_i - y_i| < \text{tolerance}, & True \\ \text{if } |x_i - y_i| \geq \text{tolerance}, & False \end{cases}$$

两个 `inf` 值和两个 `NaN` 值均不被认为相等。

参数：

- **x** (`Tensor`) - 第一个输入 `tensor`。
- **y** (`Tensor`) - 第二个输入 `tensor`。
- **tolerance** (`float`) - 两个元素被视为相等的最大偏差。默认 `1e-5`。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> mindspore.ops.approximate_equal(mindspore.tensor([1e6, 2e6, float("inf"), float("-inf"),
↪float("nan")]),
...                               mindspore.tensor([1e6, 2e7, float("inf"), float("-inf"),
↪float("nan")]))
Tensor(shape=[6], dtype=Bool, value= [ True, False, False, False, False])
```

(下页继续)

(续上页)

```

>>>
>>> mindspore.ops.approximate_equal(mindspore.tensor([1e6, 2e6, 3e6]),
...                                mindspore.tensor([1.00001e6, 2.00002e6, 3.00009e6]),
...                                tolerance=1e3)
Tensor(shape=[3], dtype=Bool, value= [ True,  True,  True])
>>>
>>> # If `x` and `y` have different datatypes, the lower precision data type will be
...   converted to the
...   relatively highest precision data type.
>>> mindspore.ops.approximate_equal(mindspore.tensor([1, 2], mindspore.int32),
...                                mindspore.tensor([1., 2], mindspore.float32))
Tensor(shape=[2], dtype=Bool, value= [ True,  True])

```

mindspore.ops.bucketize

`mindspore.ops.bucketize(input, boundaries, *, right=False)`

返回输入 `tensor` 中每个元素所属桶的索引。如果 `right` 为 `False`，则左边界开放，对于 `input` 中的每个元素 `x`，返回的索引满足以下规则：

$$\begin{cases} boundaries[i-1] < x \leq boundaries[i], & \text{if } right = False \\ boundaries[i-1] \leq x < boundaries[i], & \text{if } right = True \end{cases}$$

参数：

- **input** (Tensor) - 输入 `tensor`。
- **boundaries** (list) - 存储桶边界值的有序递增列表。

关键字参数：

- **right** (bool, 可选) - 如果为 `False`，则从边界获取输入中每个值的下限索引；如果为 `True`，则改为获取上限索引。默认 `False`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> input = mindspore.tensor([[3, 6, 9], [3, 6, 9]])
>>> boundaries = [1., 3., 5., 7., 9.]
>>> output = mindspore.ops.bucketize(input, boundaries, right=True)
>>> output
Tensor(shape=[2, 3], dtype=Int32, value=
[[2, 3, 5],
 [2, 3, 5]])

```


mindspore.ops.eq

`mindspore.ops.eq(input, other)`
逐元素计算两个输入是否相等。

$$out_i = \begin{cases} \text{True, if } input_i = other_i \\ \text{False, if } input_i \neq other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 输入必须是两个 Tensor，或是一个 Tensor 和一个 Scalar。
- 两个输入的 shape 支持广播。

参数:

- **input** (Union[Tensor, Number]) - 第一个输入。
- **other** (Union[Tensor, Number]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> x = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.eq(x, 2.0)
>>> print(output)
[False True False]
>>> # case 2: The shape of two inputs are the same
>>> x = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> y = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.eq(x, y)
>>> print(output)
[ True True False]
```

mindspore.ops.equal

`mindspore.ops.equal(input, other)`
逐元素计算两个输入是否相等。

$$out_i = \begin{cases} \text{True, if } input_i = other_i \\ \text{False, if } input_i \neq other_i \end{cases}$$

说明:

- 支持隐式类型转换。

- 两个输入的 shape 支持广播。

参数：

- **input** (Union[Tensor, Number]) - 第一个输入。
- **other** (Union[Tensor, Number]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.equal(input, 2.0)
>>> print(output)
[False True False]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.equal(input, other)
>>> print(output)
[ True  True False]
```

mindspore.ops.ge

mindspore.ops.ge(input, other)

逐元素计算 $input \geq other$ 的值。

说明：

- 支持隐式类型转换。
- 输入必须是两个 Tensor，或一个 Tensor 和一个 Scalar。
- 当输入是两个 Tensor 时，它们的数据类型不能同时是 bool，并保证其 shape 可以广播。
- 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
- 支持广播。
- 若输入的 Tensor 可以广播，则会把低维度通过复制该维度的值的方式扩展到另一个输入中对应的高维度。

$$out_i = \begin{cases} \text{True, if } input_i \geq other_i \\ \text{False, if } input_i < other_i \end{cases}$$

参数：

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.ge(input, 2.0)
>>> print(output)
[False True True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.ge(input, other)
>>> print(output)
[ True  True False]
```

mindspore.ops.greater

`mindspore.ops.greater(input, other)`

逐元素计算 $input > other$ 的值。

参数：

- **input** (Union[Tensor, Number]) - 第一个输入。
- **other** (Union[Tensor, Number]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.greater(input, 2.0)
>>> print(output)
[False False True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.greater(input, other)
>>> print(output)
[ False False False]
```

mindspore.ops.greater_equal

`mindspore.ops.greater_equal(input, other)`

逐元素计算 $input \geq other$ 的值。

参数：

- **input** (Union[Tensor, Number]) - 第一个输入。
- **other** (Union[Tensor, Number]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.greater_equal(input, 2.0)
>>> print(output)
[False True True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.greater_equal(input, other)
>>> print(output)
[ True  True False]
```

mindspore.ops.gt

`mindspore.ops.gt(input, other)`

逐元素计算 $input > other$ 的值。

$$out_i = \begin{cases} \text{True, if } input_i > other_i \\ \text{False, if } input_i \leq other_i \end{cases}$$

说明：

- 支持隐式类型转换。
 - 输入必须是两个 Tensor，或一个 Tensor 和一个 Scalar。
 - 当输入是两个 Tensor 时，它们的数据类型不能同时是 bool，并保证其 shape 可以广播。
 - 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
 - 支持广播。
 - 若输入的 Tensor 可以广播，则会把低维度通过复制该维度的值的方式扩展到另一个输入中对应的高维度。
-

参数：

- **input** (Union[Tensor, number.Number, bool]) - 第一个输入。

- **other** (Union[Tensor, number.Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.gt(input, 2.0)
>>> print(output)
[False False True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.gt(input, other)
>>> print(output)
[ False False False]
```

mindspore.ops.intopk

`mindspore.ops.intopk(x1, x2, k)`

返回第二个输入 `tensor` 中的元素是否存在于第一个输入 `tensor` 的前 `k` 个元素中。

参数：

- **x1** (Tensor) - 二维输入 `tensor`。
- **x2** (Tensor) - 一维输入 `tensor`。须满足 `x2.shape[0] = x1.shape[0]`。
- **k** (int) - 前 `k` 个元素。

返回：

一维的 `bool` 类型 `tensor`，与 `x2` `shape` 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x1 = mindspore.tensor([[1, 8, 5, 2, 7], [4, 9, 1, 3, 5]], mindspore.float32)
>>> x2 = mindspore.tensor([1, 3], mindspore.int32)
>>> mindspore.ops.intopk(x1, x2, 3)
Tensor(shape=[2], dtype=Bool, value= [ True, False])
```

mindspore.ops.isclose

`mindspore.ops.isclose(input, other, rtol=1e-05, atol=1e-08, equal_nan=False)`

返回一个布尔型 `tensor`，表示两个 `tensor` 在容忍度内是否逐元素相等。数学公式为：

$$|input - other|atol + rtol \times |other|$$

如果 `inf` 值具有相同的符号，则认为它们相等；如果 `equal_nan` 为 `True`，`NaN` 值被认为相等。

参数：

- **input** (`Tensor`) - 第一个输入 `tensor`。
- **other** (`Tensor`) - 第二个输入 `tensor`。
- **rtol** (`Union[float, int, bool]`, 可选) - 相对容忍度。默认 `1e-05`。
- **atol** (`Union[float, int, bool]`, 可选) - 绝对容忍度。默认 `1e-08`。
- **equal_nan** (`bool`, 可选) - 两个 `NaN` 是否被视为相等。默认 `False`。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> mindspore.ops.isclose(mindspore.tensor([2e6, float("inf"), float("-inf"), float("inf"),
↪float("nan")]),
...                       mindspore.tensor([2e7, float("inf"), float("-inf"), float("-inf"),
↪float("nan")]))
Tensor(shape=[5], dtype=Bool, value= [False,  True,  True, False, False])
>>>
>>> mindspore.ops.isclose(mindspore.tensor([1e6, 2e6, 3e6]),
...                       mindspore.tensor([1.00008e6, 2.00008e7, 3.00008e8]), rtol=1e3)
Tensor(shape=[3], dtype=Bool, value= [ True,  True,  True])
>>>
>>> mindspore.ops.isclose(mindspore.tensor([1e6, 2e6, 3e6]),
...                       mindspore.tensor([1.00001e6, 2.00002e6, 3.00009e6]), atol=1e3)
Tensor(shape=[3], dtype=Bool, value= [ True,  True,  True])
>>> mindspore.ops.isclose(mindspore.tensor([float("nan"), 1, 2]),
...                       mindspore.tensor([float("nan"), 1, 2]), equal_nan=True)
Tensor(shape=[3], dtype=Bool, value= [ True,  True,  True])
```

mindspore.ops.isfinite

`mindspore.ops.isfinite(input)`

返回一个布尔型 `tensor`，指示哪些元素是有限数。

如果一个元素不是 `NaN`、`-INF`、`INF`，则该元素被视为有限数。

参数：

- **input** (`Tensor`) - 输入 `tensor`。

返回：

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1, 3, float("inf"), float("-inf"), float("nan")])
>>> mindspore.ops.isfinite(input)
Tensor(shape=[5], dtype=Bool, value= [ True,  True, False, False, False])
```

mindspore.ops.isinf

`mindspore.ops.isinf(input)`

返回一个布尔型 `tensor`, 指示哪些元素是正负无穷数。

警告:

- 这是一个实验性 API, 后续可能修改或删除。
- 该 API 目前只支持在 Atlas A2 训练系列产品上使用。

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend CPU GPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1, 3, float("inf"), float("-inf"), float("nan")])
>>> mindspore.ops.isinf(input)
Tensor(shape=[5], dtype=Bool, value= [False, False,  True,  True, False])
```

mindspore.ops.isnan

`mindspore.ops.isnan(input)`

返回一个布尔型 `tensor`, 指示哪些元素是 NaN。

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1, 3, float("inf"), float("-inf"), float("nan")])
>>> mindspore.ops.isnan(input)
Tensor(shape=[5], dtype=Bool, value= [False, False, False, False,  True])
```

mindspore.ops.isneginf

`mindspore.ops.isneginf(input)`

返回一个布尔型 `tensor`, 指示哪些元素是负无穷数。

警告: 该 API 目前只支持在 Atlas A2 训练系列产品上使用。

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-1, 3, float("inf"), float("-inf"), float("nan")])
>>> mindspore.ops.isneginf(input)
Tensor(shape=[5], dtype=Bool, value= [False, False, False,  True, False])
```

mindspore.ops.isposinf

`mindspore.ops.isposinf(input)`

返回一个布尔型 `tensor`, 指示哪些元素是正无穷数。

警告: 该 API 目前只支持在 Atlas A2 训练系列产品上使用。

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:


```
>>> import mindspore
>>> input = mindspore.tensor([-1, 3, float("inf"), float("-inf"), float("nan")])
>>> mindspore.ops.isposinf(input)
Tensor(shape=[5], dtype=Bool, value= [False, False,  True, False, False])
```

mindspore.ops.isreal

`mindspore.ops.isreal(input)`

返回一个布尔型 `tensor`，指示哪些元素是实数。

一个复数的虚部是 0 时也被看作是实数。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([False, 0j, 1, 2.1, 1+2j])
>>> mindspore.ops.isreal(input)
Tensor(shape=[5], dtype=Bool, value= [ True,  True,  True,  True, False])
```

mindspore.ops.is_complex

`mindspore.ops.is_complex(input)`

返回一个布尔型 `tensor`，指示哪些元素是复数。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Bool

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 1+1j, 2+2j], mindspore.dtype.complex64)
>>> mindspore.ops.is_complex(input)
True
>>>
>>> input = mindspore.tensor([1, 1+1j, 2+2j], mindspore.dtype.complex128)
>>> mindspore.ops.is_complex(input)
True
>>> input = mindspore.tensor([1, 1+1j, 2+2j], mindspore.dtype.int32)
```

(下页继续)

(续上页)

```
>>> mindspore.ops.is_complex(input)
False
```

mindspore.ops.is_floating_point

`mindspore.ops.is_floating_point(input)`

如果 `tensor` 的数据类型是浮点数, 则返回 `True`, 否则返回 `False`。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Bool

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([False, 0j, 1, 2.1, 1+2j], mindspore.float64)
>>> mindspore.ops.is_floating_point(input)
True
>>>
>>> input = mindspore.tensor([False, 0j, 1, 2.1, 1+2j], mindspore.float32)
>>> mindspore.ops.is_floating_point(input)
True
>>>
>>> input = mindspore.tensor([False, 0j, 1, 2.1, 1+2j], mindspore.float16)
>>> mindspore.ops.is_floating_point(input)
True
>>>
>>> input = mindspore.tensor([False, 0j, 1, 2.1, 1+2j], mindspore.int32)
>>> mindspore.ops.is_floating_point(input)
False
```

mindspore.ops.le

`mindspore.ops.le(input, other)`

逐元素计算 `input <= other` 的值。

$$out_i = \begin{cases} \text{True, if } input_i \leq other_i \\ \text{False, if } input_i > other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 输入必须是两个 Tensor, 或一个 Tensor 和一个 Scalar。
- 当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常数。

参数：

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.le(input, 2.0)
>>> print(output)
[True  True False]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.le(input, other)
>>> print(output)
[ True  True  True]
```

mindspore.ops.less

`mindspore.ops.less(input, other)`
逐元素计算 $input < other$ 的值。

$$out_i = \begin{cases} \text{True, if } input_i < other_i \\ \text{False, if } input_i \geq other_i \end{cases}$$

说明：支持隐式类型转换。

参数：

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.less(input, 2.0)
>>> print(output)
[True False False]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.less(input, other)
>>> print(output)
[ False False  True]
```

mindspore.ops.less_equal

`mindspore.ops.less_equal(input, other)`

逐元素计算 $input \leq other$ 的值。

$$out_i = \begin{cases} \text{True, if } input_i \leq other_i \\ \text{False, if } input_i > other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常数。

参数:

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.less_equal(input, 2.0)
>>> print(output)
[True True False]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.less_equal(input, other)
>>> print(output)
[ True True  True]
```

mindspore.ops.lt

`mindspore.ops.lt(input, other)`
`mindspore.ops.less()` 的别名。

支持平台:

Ascend GPU CPU

mindspore.ops.maximum

`mindspore.ops.maximum(input, other)`
 逐元素计算两个输入 tensor 中最大值。

$$output_i = \max(input_i, other_i)$$

说明:

- 输入 `input` 和 `other` 遵循隐式类型转换规则，使数据类型保持一致。
- 当输入是两个 Tensor 时，它们的数据类型不能同时是 bool，并保证其 shape 可以广播。
- 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
- 支持广播。
- 如果一个元素和 NaN 比较，则返回 NaN。

警告: 如果所有输入都为标量 int 类型，在 Graph 模式下，输出为 int32 类型的 Tensor，在 PyNative 模式下，输出为 int64 类型的 Tensor。

参数:

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1 : same data type
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.float32)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float32)
>>> mindspore.ops.maximum(input, other)
Tensor(shape=[3], dtype=Float32, value= [ 4.00000000e+00,  5.00000000e+00,  6.00000000e+00])
>>>
>>> # case 2 : the data type is the one with higher precision or higher digits among the two
↳ inputs.
```

(下页继续)

(续上页)

```
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.int64)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float64)
>>> mindspore.ops.maximum(input, other)
Tensor(shape=[3], dtype=Float64, value= [ 4.00000000e+00,  5.00000000e+00,  6.00000000e+00])
```

mindspore.ops.minimum

`mindspore.ops.minimum(input, other)`
逐元素计算两个输入 `tensor` 的最小值。

$$output_i = \min(input_i, other_i)$$

说明:

- 输入 `input` 和 `other` 遵循隐式类型转换规则, 使数据类型保持一致。
- 当输入是两个 `Tensor` 时, 它们的数据类型不能同时是 `bool`。
- 当输入是一个 `Tensor` 和一个 `Scalar` 时, `Scalar` 只能是一个常数。
- 支持广播操作。
- 当一个元素与 `NaN` 比较时, 将返回 `NaN`。

参数:

- **input** (`Union[Tensor, Number, bool]`) - 第一个输入。
- **other** (`Union[Tensor, Number, bool]`) - 第二个输入。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1 : same data type
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.float32)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float32)
>>> mindspore.ops.minimum(input, other)
Tensor(shape=[3], dtype=Float32, value= [ 1.00000000e+00,  2.00000000e+00,  3.00000000e+00])
>>>
>>> # case 2 : the data type is the one with higher precision or higher digits among the two
↳ inputs.
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.int64)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float64)
>>> mindspore.ops.minimum(input, other)
Tensor(shape=[3], dtype=Float64, value= [ 1.00000000e+00,  2.00000000e+00,  3.00000000e+00])
```

mindspore.ops.msort

`mindspore.ops.msort(input)`

返回沿第一个维度对输入 `tensor` 进行升序排序后的 `tensor`。

`ops.msort(input)` 等价于 `ops.sort(axis=0)(input)[0]`。更多信息请参考 `mindspore.ops.sort()`。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[8, 2, 1],
...                             [5, 9, 3],
...                             [4, 6, 7]])
>>> mindspore.ops.msort(input)
Tensor(shape=[3, 3], dtype=Int64, value=
[[4, 2, 1],
 [5, 6, 3],
 [8, 9, 7]])
>>> # is equivalent to `ops.sort(axis=0)(input)[0]`
>>> mindspore.ops.sort(input, axis=0)[0]
Tensor(shape=[3, 3], dtype=Int64, value=
[[4, 2, 1],
 [5, 6, 3],
 [8, 9, 7]])
```

mindspore.ops.ne

`mindspore.ops.ne(input, other)`

逐元素计算两个输入是否不相等。

$$out_i = \begin{cases} \text{True, if } input_i \neq other_i \\ \text{False, if } input_i = other_i \end{cases}$$

说明：

- 支持隐式类型转换。
- 当输入是两个 Tensor 时，它们的 shape 可以广播。
- 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
- 支持广播。

参数：

- **input** (Union[Tensor, Number, bool]) - 第一个输入。

- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.ops.ne(input, 2.0)
>>> print(output)
[True False  True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.ops.ne(input, other)
>>> print(output)
[ False False  True]
```

mindspore.ops.not_equal

`mindspore.ops.not_equal(input, other)`

`mindspore.ops.ne()` 的别名。

支持平台：

Ascend GPU CPU

mindspore.ops.searchsorted

`mindspore.ops.searchsorted(sorted_sequence, values, *, out_int32=False, right=False, side=None, sorter=None)`

返回元素能够插入输入 tensor 的位置索引，以维持原 tensor 的递增顺序。

参数：

- **sorted_sequence** (Tensor) - 输入 tensor。如果未提供 *sorter*，最内层的维度上须为递增的序列。
- **values** (Tensor) - 要插入元素的值。

关键字参数：

- **out_int32** (bool, 可选) - 输出数据类型是否为 `mindspore.int32`。如果为 `False`，则输出数据类型将为 `mindspore.int64`。默认 `False`。
- **right** (bool, 可选) - 搜索策略。如果为 `True`，则返回找到的最后一个合适的索引；如果为 `False`，则返回第一个合适的索引。默认 `False`。
- **side** (str, 可选) - 跟参数 *right* 功能一致，如果参数值为 `left`，相当于 *right* 为 `False`。如果参数值为 `right`，相当于 *right* 为 `True`。如果值为 `left` 但是 *right* 为 `True` 则报错。默认 `None`。
- **sorter** (Tensor, 可选) - 按 *sorted_sequence* 最内层维度升序排序的索引序列，与未排序的 *sorted_sequence* 共同使用。CPU 和 GPU 只支持 `None`。默认 `None`。

返回：

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> sorted_sequence = mindspore.tensor([1, 2, 2, 3, 4, 5, 5])
>>> values = mindspore.tensor([2])
>>> mindspore.ops.searchsorted(sorted_sequence, values)
Tensor(shape=[1], dtype=Int64, value= [1])
```

`mindspore.ops.topk`

`mindspore.ops.topk(input, k, dim=None, largest=True, sorted=True)`

按指定维度返回前 k 个最大或最小元素及其对应索引。

警告:

- 如果 `sorted` 设置为 `False`，它将使用 `aicpu` 运算符，性能可能会降低。此外，由于在不同平台上存在内存排布以及遍历方式不同等问题，`sorted` 设置为 `False` 时计算结果的显示顺序可能会出现不一致的情况。

参数:

- input** (Tensor) - 输入 tensor。
- k** (int) - 返回的元素数量。
- dim** (int, 可选) - 指定维度。如果为 `None`，则会按最后一个维度排序。默认 `None`。
- largest** (bool, 可选) - 如果为 `True`，返回最大元素。如果为 `False`，返回最小元素。默认 `True`。
- sorted** (bool, 可选) - 如果为 `True`，则返回的元素按降序排序。如果为 `False`，则不对获取的元素进行排序。默认 `True`。

返回:

两个 tensor 组成的 tuple(values, indices)

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[8, 2, 1],
...                             [5, 9, 3],
...                             [4, 6, 7]])
>>> # case 1: If dim is not given, the last dimension of the input is chosen.
>>> mindspore.ops.topk(input, 2)
(Tensor(shape=[3, 2], dtype=Int64, value=
  [[8, 2],
   [9, 5],
   [7, 6]]),
 Tensor(shape=[3, 2], dtype=Int32, value=
```

(下页继续)

(续上页)

```
[[0, 1],
 [1, 0],
 [2, 1]])
>>> # case 2: when dim is 0:
>>> mindspore.ops.topk(input, 2, dim=0)
(Tensor(shape=[2, 3], dtype=Int64, value=
[[8, 9, 7],
 [5, 6, 3]]),
Tensor(shape=[2, 3], dtype=Int32, value=
[[0, 1, 2],
 [1, 2, 1]]))
>>> # case 3: when largest is False, return smallest values.
>>> mindspore.ops.topk(input, 2, dim=0, largest=False)
(Tensor(shape=[2, 3], dtype=Int64, value=
[[4, 2, 1],
 [5, 6, 3]]),
Tensor(shape=[2, 3], dtype=Int32, value=
[[2, 0, 0],
 [1, 2, 1]]))
```

2.2.4 线性代数函数

接口名	概述	支持平台
<code>mindspore.ops.addbmm</code>	对 <i>batch1</i> 和 <i>batch2</i> 中的矩阵相乘，按照第一维度累计相加，再与 <i>input</i> 相加。	Ascend GPU CPU
<code>mindspore.ops.addmm</code>	对 <i>mat1</i> 和 <i>mat2</i> 矩阵乘，再将结果与 <i>input</i> 相加。	Ascend GPU CPU
<code>mindspore.ops.addr</code>	计算 <i>vec1</i> 和 <i>vec2</i> 的外积，并加到 <i>x</i> 中。	Ascend GPU CPU
<code>mindspore.ops.adjoint</code>	逐元素计算输入 <i>tensor</i> 的共轭，并转置最后两个维度。	Ascend GPU CPU
<code>mindspore.ops.baddbmm</code>	对 <i>batch1</i> 和 <i>batch2</i> 中的矩阵相乘，并与 <i>input</i> 相加。	Ascend GPU CPU
<code>mindspore.ops.batch_dot</code>	计算 <i>x1</i> 和 <i>x2</i> 中的向量点积。	Ascend GPU CPU
<code>mindspore.ops.bmm</code>	对输入的两个 <i>tensor</i> 执行批量矩阵乘积。	Ascend GPU CPU
<code>mindspore.ops.cholesky</code>	计算给定对称正定矩阵的 Cholesky 分解。	GPU CPU
<code>mindspore.ops.cholesky_solve</code>	根据 Cholesky 分解因子 <i>input2</i> 计算一组具有正定矩阵的线性方程组的解。	Ascend GPU CPU
<code>mindspore.ops.cond</code>	返回给定 <i>Tensor</i> 的矩阵范数或向量范数。	GPU CPU
<code>mindspore.ops.dot</code>	计算两个输入 <i>tensor</i> 的点积。	Ascend GPU CPU
<code>mindspore.ops.eigvals</code>	计算输入方阵的特征值。	Ascend CPU
<code>mindspore.ops.geqrf</code>	将输入 <i>tensor</i> 进行 QR 分解 $A = QR$ 。	Ascend GPU CPU
<code>mindspore.ops.ger</code>	计算输入一维 <i>tensor input</i> 与 <i>vec2</i> 的外积。	Ascend GPU CPU
<code>mindspore.ops.inner</code>	计算两个一维 <i>tensor</i> 的点积。	Ascend GPU CPU
<code>mindspore.ops.inverse</code>	计算输入矩阵的逆。	GPU CPU
<code>mindspore.ops.kron</code>	计算 <i>input</i> 和 <i>other</i> 的 Kronecker 积。	Ascend GPU CPU
<code>mindspore.ops.logdet</code>	计算输入方块矩阵的对数行列式。	CPU
<code>mindspore.ops.lu_solve</code>	计算线性方程组 $Ay = b$ 的 LU 解。	Ascend GPU CPU
<code>mindspore.ops.lu_unpack</code>	将 <code>mindspore.scipy.linalg.lu_factor()</code> 返回的 LU 分解结果解包为 P、L、U 矩阵。	GPU CPU

下页继续

表 9 - 续上页

<code>mindspore.ops.matmul</code>	计算两个输入的矩阵乘积。	Ascend GPU CPU
<code>mindspore.ops.matrix_band_part</code>	返回一个 <code>tensor</code> , 保留指定对角线的值, 其余设为 0。	Ascend GPU CPU
<code>mindspore.ops.matrix_solve</code>	求解线性方程组。	Ascend CPU
<code>mindspore.ops.matrix_diag</code>	使用 x 中的元素值作为输出 <code>tensor</code> 的对角线, 其余用 <code>padding_value</code> 填充。	Ascend GPU CPU
<code>mindspore.ops.matrix_diag_part</code>	返回一个 <code>tensor</code> , 包含输入 <code>tensor x</code> 的第 $k[0]$ 到 $k[1]$ 个对角线中的值, 缺失部分使用 <code>padding_value</code> 填充。	Ascend GPU CPU
<code>mindspore.ops.matrix_set_diag</code>	返回一个 <code>tensor</code> , 使用输入 <code>diagonal</code> 中的元素值替换 x 矩阵的第 $k[0]$ 条到第 $k[1]$ 条对角线上的元素值。	Ascend GPU CPU
<code>mindspore.ops.mm</code>	计算两个输入的矩阵乘积。	Ascend GPU CPU
<code>mindspore.ops.mv</code>	实现矩阵 <code>mat</code> 和向量 <code>vec</code> 相乘。	Ascend GPU CPU
<code>mindspore.ops.outer</code>	计算两个 <code>tensor</code> 的外积。	Ascend GPU CPU
<code>mindspore.ops.orgqr</code>	计算 Householder 矩阵的前 N 列。	Ascend GPU CPU
<code>mindspore.ops.ormqr</code>	计算一个普通矩阵 <code>other</code> 与一个矩阵 Q (由 Householder 反射向量 <code>input</code> 和反射系数 <code>tau</code> 生成) 的乘积。	GPU
<code>mindspore.ops.pinv</code>	计算矩阵的伪逆 (Moore-Penrose)。	CPU
<code>mindspore.ops.svd</code>	计算单个或多个矩阵的奇异值分解。	Ascend GPU CPU
<code>mindspore.ops.slogdet</code>	对一个或多个方阵行列式的绝对值取对数, 返回其符号和值。	Ascend GPU CPU
<code>mindspore.ops.trace</code>	返回 <code>tensor</code> 对角线的元素总和。	Ascend GPU CPU
<code>mindspore.ops.tensor_dot</code>	沿指定轴计算两个张量的点乘。	Ascend GPU CPU
<code>mindspore.ops.vander</code>	生成一个范德蒙矩阵。	Ascend GPU CPU
<code>mindspore.ops.vecdot</code>	按指定轴, 计算两批向量的点积。	Ascend GPU CPU

mindspore.ops.addbmm

`mindspore.ops.addbmm(input, batch1, batch2, *, beta=1, alpha=1)`

对 `batch1` 和 `batch2` 中的矩阵相乘, 按照第一维度累计相加, 再与 `input` 相加。

说明:

- `batch1` 和 `batch2` 必须是三维的 `tensor`, 且包含相同数量的矩阵。
- 如果 `batch1` 是大小为 (C, W, T) 的 `tensor`, `batch2` 是大小为 (C, T, H) 的 `tensor`, 则 `input` 必须能够与大小为 (W, H) 的 `tensor` 进行广播, 且输出将是大小为 (W, H) 的 `tensor`。
- 若 `beta` 为 0, 那么 `input` 将会被忽略。

$$output = \beta input + \alpha \left(\sum_{i=0}^{b-1} batch1_i @ batch2_i \right)$$

参数:

- **input** (Tensor) - 输入 `tensor`。
- **batch1** (Tensor) - 第一个将被乘 `batch` 矩阵。
- **batch2** (Tensor) - 第二个将被乘 `batch` 矩阵。

关键字参数:

- **beta** (Union[int, float], 可选) - *input* 的尺度因子。默认 1。
- **alpha** (Union[int, float], 可选) - (*batch1 @ batch2*) 的尺度因子。默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> m = mindspore.ops.ones((3, 3))
>>> arr1 = mindspore.tensor([[8., 7., 6.], [5., 4., 3.], [2., 1., 0.]])
>>> arr2 = mindspore.tensor([[5., 4., 3.], [2., 1., 0.], [8., 7., 6.]])
>>> output = mindspore.ops.addbmm(m, arr1, arr2)
>>> print(output)
[[172. 136. 100.]
 [172. 136. 100.]
 [172. 136. 100.]
```

mindspore.ops.addmm

`mindspore.ops.addmm(input, mat1, mat2, *, beta=1, alpha=1)`

对 *mat1* 和 *mat2* 矩阵乘，再将结果与 *input* 相加。

说明：

- 若 *beta* 为 0，那么 *input* 将会被忽略。

$$output = \beta input + \alpha(mat1 @ mat2)$$

参数：

- **input** (Tensor) - 输入 tensor。
- **mat1** (Tensor) - 第一个矩阵。
- **mat2** (Tensor) - 第二个矩阵。

关键字参数：

- **beta** (Union[int, float], 可选) - *input* 的尺度因子。默认 1。
- **alpha** (Union[int, float], 可选) - (*mat1 @ mat2*) 的尺度因子。默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> m = mindspore.ops.ones((3, 3))
>>> arr1 = mindspore.tensor([[8., 7., 6.], [5., 4., 3.], [2., 1., 0.]])
>>> arr2 = mindspore.tensor([[5., 4., 3.], [2., 1., 0.], [8., 7., 6.]])
>>> output = mindspore.ops.addmm(m, arr1, arr2)
>>> print(output)
[[103.  82.  61.]
 [ 58.  46.  34.]
 [ 13.  10.   7.]]
```

mindspore.ops.addr

`mindspore.ops.addr(x, vec1, vec2, *, beta=1, alpha=1)`

计算 *vec1* 和 *vec2* 的外积，并加到 *x* 中。

说明:

- 如果 *vec1* 是一个大小为 *N* 的向量，*vec2* 是一个大小为 *M* 的向量，则 *x* 的大小可广播为。则 *x* 必须能够与大小为 (*N*, *M*) 的矩阵进行广播，且输出将是大小为 (*N*, *M*) 的矩阵。
- 若 *beta* 为 0，那么 *input* 将会被忽略。

$$output = x + (vec1vec2)$$

参数:

- **x** (Tensor) - 输入 tensor。
- **vec1** (Tensor) - 将被乘的向量。
- **vec2** (Tensor) - 将被乘的向量。

关键字参数:

- **beta** (scalar[int, float, bool], 可选) - *x* 的尺度因子。默认 1。
- **alpha** (scalar[int, float, bool], 可选) - (*vec1* $\otimes$ *vec2*) 的尺度因子。默认 1。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[2., 2.], [3., 2.], [3., 4.]])
>>> vec1 = mindspore.tensor([2., 3., 2.])
>>> vec2 = mindspore.tensor([3., 4.])
>>> output = mindspore.ops.addr(x, vec1, vec2)
>>> print(output)
[[ 8. 10.]
 [12. 14.]
 [ 9. 12.]]
```

mindspore.ops.adjoint

`mindspore.ops.adjoint(x)`

逐元素计算输入 `tensor` 的共轭，并转置最后两个维度。

参数：

- **x** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[[0. + 0.j, 1. + 1.j], [2. + 2.j, 3. + 3.j]]], mindspore.
↳complex128)
>>> mindspore.ops.adjoint(x)
Tensor(shape=[2, 2], dtype=Complex128, value=
[[0-0j, 2-2j],
 [1-1j, 3-3j]])
```

mindspore.ops.baddbmm

`mindspore.ops.baddbmm(input, batch1, batch2, beta=1, alpha=1)`

对 `batch1` 和 `batch2` 中的矩阵相乘，并与 `input` 相加。

说明：

- `batch1` 和 `batch2` 必须是三维的 `tensor`，且包含相同数量的矩阵。
- 如果 `batch1` 是大小为 (C, W, T) 的 `tensor`，`batch2` 是大小为 (C, T, H) 的 `tensor`，则 `input` 必须能够与大小为 (C, W, H) 的 `tensor` 进行广播，且输出将是大小为 (C, W, H) 的 `tensor`。
- 若 `beta` 为 0，那么 `input` 将会被忽略。
- 当输入的类型不是 `FloatTensor` 时，参数 `beta` 和 `alpha` 必须是整数。

$$\text{out}_i = \beta \text{input}_i + \alpha (\text{batch1}_i @ \text{batch2}_i)$$

参数：

- **input** (Tensor) - 输入 `tensor`。
- **batch1** (Tensor) - 第一个 `batch` 矩阵。
- **batch2** (Tensor) - 第二个 `batch` 矩阵。
- **beta** (Union[float, int], 可选) - `input` 的尺度因子。默认 1。
- **alpha** (Union[float, int], 可选) - $(\text{batch1} @ \text{batch2})$ 的尺度因子，默认 1。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.ops.ones((3, 3))
>>> batch1 = mindspore.tensor([[8., 7., 6.], [5., 4., 3.], [2., 1., 0.]])
>>> batch2 = mindspore.tensor([[5., 4., 3.], [2., 1., 0.], [8., 7., 6.]])
>>> output = mindspore.ops.baddbmm(input, batch1, batch2)
>>> print(output)
[[103.  82.  61.]
 [ 58.  46.  34.]
 [ 13.  10.   7.]
```

mindspore.ops.batch_dot

mindspore.ops.batch_dot (*x1*, *x2*, *axes=None*)

计算 *x1* 和 *x2* 中的向量点积。

说明:

- *x1* 和 *x2* 的第零维表示 batch 数量, *x1* 和 *x2* 数据类型为 float32 且秩大于或等于 2。

$$output = x1[batch, :] \otimes x2[batch, :]$$

参数:

- **x1** (Tensor) - 第一个 batch 向量。
- **x2** (Tensor) - 第二个 batch 向量。
- **axes** (Union[int, tuple(int), list(int)]) - 指定计算轴。默认 None。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: axes is a tuple(axes of `x1`, axes of `x2`)
>>> x1 = mindspore.ops.ones([2, 2, 3])
>>> x2 = mindspore.ops.ones([2, 3, 2])
>>> axes = (-1, -2)
>>> output = mindspore.ops.batch_dot(x1, x2, axes)
>>> print(output)
[[[3. 3.]
 [3. 3.]]
 [[3. 3.]
 [3. 3.]]]
>>> print(output.shape)
(2, 2, 2)
>>> x1 = mindspore.ops.ones([2, 2], mindspore.float32)
>>> x2 = mindspore.ops.ones([2, 3, 2], mindspore.float32)
```

(下页继续)

(续上页)

```

>>> axes = (1, 2)
>>> output = mindspore.ops.batch_dot(x1, x2, axes)
>>> print(output)
[[2. 2. 2.]
 [2. 2. 2.]]
>>> print(output.shape)
(2, 3)
>>>
>>> # case 2: axes is None
>>> x1 = mindspore.ops.ones([6, 2, 3, 4], mindspore.float32)
>>> x2 = mindspore.ops.ones([6, 5, 4, 8], mindspore.float32)
>>> output = mindspore.ops.batch_dot(x1, x2)
>>> print(output.shape)
(6, 2, 3, 5, 8)
>>>
>>> # case 3: axes is a int data.
>>> x1 = mindspore.ops.ones([2, 2, 4])
>>> x2 = mindspore.ops.ones([2, 5, 4, 5])
>>> output = mindspore.ops.batch_dot(x1, x2, 2)
>>> print(output.shape)
(2, 2, 5, 5)

```

mindspore.ops.bmm

mindspore.ops.bmm(input_x, mat2)

对输入的两个 tensor 执行批量矩阵乘积。

$$\text{output}[\dots, :, :] = \text{matrix}(\text{input}_x[\dots, :, :]) * \text{matrix}(\text{mat2}[\dots, :, :])$$

input_x 的维度不能小于 3，mat2 的维度不能小于 2。

参数：

- **input_x** (Tensor) - 输入的第一个 Tensor。
- **mat2** (Tensor) - 输入的第二个 Tensor。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> input_x = mindspore.ops.arange(24, dtype=mindspore.float32).reshape(2, 4, 1, 3)
>>> mat2 = mindspore.ops.arange(72, dtype=mindspore.float32).reshape(2, 4, 3, 3)
>>> out = mindspore.ops.bmm(input_x, mat2)
>>> print(out)
[[[ [ 15,   18,   21]],
  [ [ 150,  162,  174]],
  [ [ 447,  468,  489]],
  [ [ 906,  936,  966]]],
 [[ [1527, 1566, 1605]],

```

(下页继续)

(续上页)

```
[[2310, 2358, 2406]],
[[3255, 3312, 3369]],
[[4362, 4428, 4494]]]
```

mindspore.ops.cholesky

`mindspore.ops.cholesky(input_x, upper=False)`

计算给定对称正定矩阵的 Cholesky 分解。

如果 `upper` 为 `True`, 则返回的矩阵 U 为上三角矩阵, 分解形式为:

$$A = U^T U$$

如果 `upper` 为 `False`, 则返回的矩阵 L 为下三角矩阵, 分解形式为:

$$A = LL^T$$

参数:

- **input_x** (Tensor) - shape 大小为 $(*, N, N)$ 的输入 tensor, 公式中的 A 。
- **upper** (bool, 可选) - 是否返回上三角矩阵, 默认 `False`。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.tensor([[1.0, 1.0], [1.0, 2.0]])
>>> output = mindspore.ops.cholesky(input_x, upper=False)
>>> print(output)
[[1. 0.]
 [1. 1.]]
```

mindspore.ops.cholesky_solve

`mindspore.ops.cholesky_solve(input, input2, upper=False)`

根据 Cholesky 分解因子 `input2` 计算一组具有正定矩阵的线性方程组的解。

如果 `upper` 为 `True`, `input2` 是上三角矩阵, 输出的结果:

$$output = (input2^T * input2)^{-1} input$$

如果 `upper` 为 `False`, `input2` 是下三角矩阵, 输出的结果:

$$output = (input2 * input2^T)^{-1} input$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - shape 为 $(*, N, M)$ 的输入 tensor。
- **input2** (Tensor) - shape 为 $(*, N, N)$ 的 tensor，Cholesky 因子。
- **upper** (bool, 可选) - 是否视为上三角矩阵。默认 False。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input1 = mindspore.tensor([[1., 0., 0.], [0., 1., 0.], [0., 0., 1.]])
>>> input2 = mindspore.tensor([[2., 0., 0.], [4., 1., 0.], [-1., 1., 2.]])
>>> out = mindspore.ops.cholesky_solve(input1, input2, upper=False)
>>> print(out)
[[ 5.8125 -2.625  0.625 ]
 [-2.625  1.25  -0.25 ]
 [ 0.625 -0.25  0.25 ]]
```

mindspore.ops.cond

mindspore.ops.cond(A, p=None)

返回给定 Tensor 的矩阵范数或向量范数。

p 为 norm 的计算模式。支持下列 norm 模式。

p	矩阵范数	向量范数
None (默认值)	2-norm (参考最下方公式)	2-norm (参考最下方公式)
'fro'	Frobenius norm	不支持
'nuc'	nuclear norm	不支持
inf	$\max(\text{sum}(\text{abs}(x), \text{dim} = 1))$	$\max(\text{abs}(x))$
-inf	$\min(\text{sum}(\text{abs}(x), \text{dim} = 1))$	$\min(\text{abs}(x))$
0	不支持	$\text{sum}(x \neq 0)$
1	$\max(\text{sum}(\text{abs}(x), \text{dim} = 0))$	参考最下方公式
-1	$\min(\text{sum}(\text{abs}(x), \text{dim} = 0))$	参考最下方公式
2	最大奇异值	参考最下方公式
-2	最小奇异值	参考最下方公式
其余 int 或 float 值	不支持	$\text{sum}(\text{abs}(x)^p)^{(1/p)}$

说明：当前暂不支持复数。

参数：

- **A** (Tensor) - 是零个或多个 batch 维。

- **p** (Union[int, float, inf, -inf, 'fro', 'nuc'], 可选) - norm 的模式, 参考上表, 默认 None。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[1.0, 0.0, -1.0], [0.0, 1.0, 0.0], [1.0, 0.0, 1.0]])
>>> print(mindspore.ops.cond(x))
1.4142
>>> print(mindspore.ops.cond(x, 'fro'))
3.1622777
```

mindspore.ops.dot

mindspore.ops.dot (*input, other*)

计算两个输入 tensor 的点积。

说明:

- 输入为 float16 或 float32, 且秩必须大于或等于 2。

参数:

- **input** (Tensor) - 第一个输入的 tensor。
- **other** (Tensor) - 第二个输入的 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.ops.ones([2, 3])
>>> other = mindspore.ops.ones([1, 3, 2])
>>> output = mindspore.ops.dot(input, other)
>>> print(output)
[[[3. 3.]]
 [[3. 3.]]]
>>> print(output.shape)
(2, 1, 2)
>>> input = mindspore.ops.ones([1, 2, 3])
>>> other = mindspore.ops.ones([1, 3, 2])
>>> output = mindspore.ops.dot(input, other)
>>> print(output)
[[[[3. 3.]]
```

(下页继续)

(续上页)

```

[[[3. 3.]]]]
>>> print(output.shape)
(1, 2, 1, 2)
>>> input = mindspore.ops.ones([1, 2, 3])
>>> other = mindspore.ops.ones([2, 3, 2])
>>> output = mindspore.ops.dot(input, other)
>>> print(output)
[[[3. 3.]
  [3. 3.]]
 [[3. 3.]
  [3. 3.]]]]
>>> print(output.shape)
(1, 2, 2, 2)
>>> input = mindspore.ops.ones([3, 2, 3])
>>> other = mindspore.ops.ones([2, 1, 3, 2])
>>> output = mindspore.ops.dot(input, other)
>>> print(output)
[[[[3. 3.]]
  [[3. 3.]]]
 [[3. 3.]]
  [[3. 3.]]]]
[[[[3. 3.]]
  [[3. 3.]]]
 [[3. 3.]]
  [[3. 3.]]]]
[[[[3. 3.]]
  [[3. 3.]]]
 [[3. 3.]]
  [[3. 3.]]]]]
>>> print(output.shape)
(3, 2, 2, 1, 2)

```

mindspore.ops.eigvals

`mindspore.ops.eigvals(A)`

计算输入方阵的特征值。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **A** (Tensor) - shape 为 $(*, N, N)$ 的输入 tensor。

返回：

Tensor

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> mindspore.ops.eigvals(mindspore.tensor([[1.0, 0.0], [0.0, 2.0]]))
Tensor(shape=[2], dtype=Complex64, value= [1+0j, 2+0j])
```

mindspore.ops.geqrf

`mindspore.ops.geqrf(input)`

将输入 `tensor` 进行 QR 分解 $A = QR$ 。

分解为正交矩阵 Q 和上三角矩阵 R 的乘积。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

两个 `tensor` 组成的 `tuple(y, tau)`。

- **y** (Tensor) - 隐式存储 Q 和 R 矩阵。 Q (Householder 反射向量) 存储在对角线下方， R 的元素存储在对角线及上方。
- **tau** (Tensor) - 存储每个 Householder 变换的缩放因子 (Householder 反射系数)。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[-2.0, -1.0], [1.0, 2.0]])
>>> y, tau = mindspore.ops.geqrf(input)
>>> print(y)
[[ 2.236068    1.7888544]
 [-0.236068    1.3416407]]
>>> print(tau)
[1.8944271 0.          ]
```

mindspore.ops.ger

`mindspore.ops.ger(input, vec2)`

计算输入一维 `tensor input` 与 `vec2` 的外积。

说明：Ascend 平台，不支持 float64 类型。

参数：

- **input** (Tensor) - 一维输入 `tensor`。
- **vec2** (Tensor) - 一维输入 `tensor`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1., 2., 3., 4.])
>>> vec2 = mindspore.tensor([1., 2., 3.])
>>> output = mindspore.ops.ger(input, vec2)
>>> print(output)
[[ 1.  2.  3.]
 [ 2.  4.  6.]
 [ 3.  6.  9.]
 [ 4.  8. 12.]]
```

mindspore.ops.inner

mindspore.ops.**inner** (*input*, *other*)

计算两个一维 tensor 的点积。

对于更高的维度，返回最后一个轴上逐元素乘积后的和。

说明: 如果 *input* 或 *other* 之一是标量，那么 `mindspore.ops.inner()` 相当于 `mindspore.ops.mul()`。

参数:

- **input** (Tensor) - 第一个输入的 tensor。
- **other** (Tensor) - 第二个输入的 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> # case 1: Two 1D tensors
>>> input = mindspore.tensor([1., 2., 3.])
>>> y = mindspore.tensor([4., 5., 6.])
>>> mindspore.ops.inner(input, y)
Tensor(shape=[], dtype=Float32, value= 32)
>>> # case2: Tensor scalar and tensor
>>> input = mindspore.tensor([[[[1., 2., 3.], [3., 2., 1.]], [[4., 5., 6.], [4., 5., 6.]]]])
>>> y = mindspore.tensor(2.)
>>> output = mindspore.ops.inner(input, y)
>>> print(output)
[[[ 2.  4.  6.]
```

(下页继续)

(续上页)

```

[ 6.  4.  2.]]
[[ 8. 10. 12.]
 [ 8. 10. 12.]]]
>>> # case3: Two tensors
>>> input = mindspore.tensor([[[1., 2., 3.], [3., 2., 1.]], [[4., 5., 6.], [4., 5., 6.]])
>>> y = mindspore.tensor([[2., 3., 4.], [4., 3., 2.]])
>>> output = mindspore.ops.inner(input, y)
>>> print(output)
[[[20. 16.]
  [16. 20.]]
 [[47. 43.]
  [47. 43.]]]

```

mindspore.ops.inverse

`mindspore.ops.inverse(input)`

计算输入矩阵的逆。

说明: *input* 至少是两维的, 最后两个维度大小相同, 并且矩阵需要可逆。不支持复数类型的输入。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

GPU CPU

样例:

```

>>> import mindspore
>>> print(mindspore.ops.inverse(mindspore.tensor([[1., 2.], [3., 4.]])
[[-2.   1.]
 [ 1.5 -0.5]]

```

mindspore.ops.kron

`mindspore.ops.kron(input, other)`

计算 *input* 和 *other* 的 Kronecker 积。

如果 *input* 的 shape 为 $(r_0, r_1, \dots, r_N)$, *other* 的 shape 为 $(s_0, s_1, \dots, s_N)$, 则计算结果的 shape 为 $(r_0*s_0, r_1*s_1, \dots, r_N*s_N)$ 。

$$(inputy)_{k_0, k_1, \dots, k_n} = input_{i_0, i_1, \dots, i_n} * other_{j_0, j_1, \dots, j_n},$$

其中, 对于所有的 $0 \leq t \leq n$, 都有 $k_t = i_t * b_t + j_t$ 。

说明: 支持实数和复数类型的输入。

参数:

- **input** (Tensor) - 输入 tensor。
- **other** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[0., 1., 2.], [3., 4., 5.]])
>>> other = mindspore.tensor([[-1., -2., -3.], [-4., -6., -8.]])
>>> output = mindspore.ops.kron(input, other)
>>> print(output)
[[ 0.  0.  0. -1. -2. -3. -2. -4. -6.]
 [ 0.  0.  0. -4. -6. -8. -8. -12. -16.]
 [-3. -6. -9. -4. -8. -12. -5. -10. -15.]
 [-12. -18. -24. -16. -24. -32. -20. -30. -40.]]
```

mindspore.ops.logdet

mindspore.ops.logdet (*input*)

计算输入方块矩阵的对数行列式。

参数:

- **input** (Tensor) - shape 为 $(*, N, N)$ 的输入 tensor, * 表示 batch 数量。

返回:

Tensor

支持平台:

CPU

样例:

```
>>> import mindspore
>>> a = mindspore.tensor([[[8., 9.], [1., 2.]], [[5., 6.], [3., 4.]]])
>>> mindspore.ops.logdet(a)
Tensor(shape=[2], dtype=Float32, value= [ 1.94591010e+00,  6.93146825e-01])
```


mindspore.ops.lu_solve

`mindspore.ops.lu_solve(b, LU_data, LU_pivots)`

计算线性方程组 $Ay = b$ 的 LU 解。

说明:

- b 的 shape 为 $(*, m, k)$, LU_data 的 shape 为 $(*, m, m)$, LU_pivots 的 shape 为 $(*, m)$, $*$ 表示 batch 数量。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **b** (Tensor) - 上面公式的列向量 b 。
- **LU_data** (Tensor) - LU 分解的结果, 上面公式中的 A 。
- **LU_pivots** (Tensor) - LU 分解的主元, 主元可以被转为变换矩阵 P 。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> b = mindspore.tensor([[1.], [3.], [3.]])
>>> LU_data = mindspore.tensor([[2., 1., 1.], [0.5, 1., 1.5], [0.5, 0., 2.5]])
>>> LU_pivots = mindspore.tensor([2, 2, 3], mindspore.int32)
>>> y = mindspore.ops.lu_solve(b, LU_data, LU_pivots)
>>> print(y)
[[ 1.9000001]
 [-1.4000001]
 [ 0.6      ]]
```

mindspore.ops.lu_unpack

`mindspore.ops.lu_unpack(LU_data, LU_pivots, unpack_data=True, unpack_pivots=True)`

将 `mindspore.scipy.linalg.lu_factor()` 返回的 LU 分解结果解包为 P 、 L 、 U 矩阵。

说明:

- LU_data shape 为 $(*, M, N)$, LU_pivots shape 为 $(*, \min(M, N))$, $*$ 表示 batch 数量。

参数:

- **LU_data** (Tensor) - 打包的 LU 分解数据, 秩大于等于 2。
- **LU_pivots** (Tensor) - 打包的 LU 分解枢轴。
- **unpack_data** (bool, 可选) - 是否解压缩 LU_data 。如果为 `False`, 则返回的 L 和 U 为 `None`。默认 `True`。

- **unpack_pivots** (bool, 可选) - 是否将 *LU_pivots* 解压缩为置换矩阵 **P**。如果为 `False`，则返回的 **P** 为 `None`。默认 `True`。

返回：

由 `tensor` 组成的 `tuple`。分别为：LU 分解的置换矩阵、LU 分解的 **L** 矩阵、LU 分解的 **U** 矩阵。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> LU_data = mindspore.tensor([[-0.3806, -0.4872, 0.5536],
...                             [-0.1287, 0.6508, -0.2396],
...                             [ 0.2583, 0.5239, 0.6902]],
...                             [[ 0.6706, -1.1782, 0.4574],
...                             [-0.6401, -0.4779, 0.6701],
...                             [ 0.1015, -0.5363, 0.6165]])
>>> LU_pivots = mindspore.tensor([[[1, 3, 3], [2, 3, 3]]], mindspore.int32)
>>> pivots, L, U = mindspore.ops.lu_unpack(LU_data, LU_pivots)
>>> print(pivots)
[[[1. 0. 0.]
  [0. 0. 1.]
  [0. 1. 0.]]
 [[0. 0. 1.]
  [1. 0. 0.]
  [0. 1. 0.]]]
>>> print(L)
[[[ 1.         0.         0.]
  [-0.1287     1.         0.]
  [ 0.2583     0.5239     1.]]
 [[ 1.0000     0.         0.]
  [-0.6401     1.         0.]
  [ 0.1015    -0.5363     1.]]]
>>> print(U)
[[[-0.3806 -0.4872 0.5536]
  [ 0.        0.6508 -0.2396]
  [ 0.        0.        0.6902]]
 [[ 0.6706 -1.1782 0.4574]
  [ 0.      -0.4779 0.6701]
  [ 0.        0.        0.6165]]]
```

`mindspore.ops.matmul`

`mindspore.ops.matmul` (*input*, *other*)

计算两个输入的矩阵乘积。

说明：

- *input* 和 *other* 的数据类型必须一致，不支持 `Scalar`，两者须支持广播。
- Ascend 平台，*input* 和 *other* 的秩必须在 1 到 6 之间。

参数：

- **input** (Tensor) - 第一个输入 tensor。
- **other** (Tensor) - 第二个输入 tensor。

返回：

Tensor 或 Scalar

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> # case 1 : Reasonable application of broadcast mechanism.
>>> input = mindspore.ops.arange(24, dtype=mindspore.float32).reshape(2, 3, 4)
>>> other = mindspore.ops.arange(20, dtype=mindspore.float32).reshape(4, 5)
>>> output = mindspore.ops.matmul(input, other)
>>> print(output)
[[ [ 70,   76,   82,   88,   94],
   [ 190,  212,  234,  256,  278],
   [ 310,  348,  386,  424,  462]],
  [[ 430,  484,  538,  592,  646],
   [ 550,  620,  690,  760,  830],
   [ 670,  756,  842,  928, 1014]]]
>>>
>>> # case 2 : The rank of `input` is 1.
>>> input = mindspore.ops.ones([1, 2])
>>> other = mindspore.ops.ones([2])
>>> mindspore.ops.matmul(input, other)
Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00])
```

mindspore.ops.matrix_band_part

mindspore.ops.matrix_band_part (*x*, *lower*, *upper*)

返回一个 tensor，保留指定对角线的值，其余设为 0。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **x** (Tensor) - 输入 tensor。
- **lower** (Union[int, Tensor]) - 要保留的次对角线数。如果为负数，则保留对角线下方所有元素。
- **upper** (Union[int, Tensor]) - 要保留的超对角线数。如果为负数，则保留对角线上方所有元素。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.ops.ones([2, 4, 4])
>>> output = mindspore.ops.matrix_band_part(x, 2, 1)
>>> print(output)
[[[1. 1. 0. 0.]
  [1. 1. 1. 0.]
  [1. 1. 1. 1.]
  [0. 1. 1. 1.]]
 [[1. 1. 0. 0.]
  [1. 1. 1. 0.]
  [1. 1. 1. 1.]
  [0. 1. 1. 1.]]]
```

mindspore.ops.matrix_solve

`mindspore.ops.matrix_solve(matrix, rhs, adjoint=False)`

求解线性方程组。

$$\begin{aligned} matrix[..., M, M] * x[..., M, K] &= rhs[..., M, K] \\ adjoint(matrix[..., M, M]) * x[..., M, K] &= rhs[..., M, K] \end{aligned}$$

警告:

- 当平台为 GPU 时, 如果 *matrix* 中的矩阵不可逆, 将产生错误或者返回一个未知结果。

参数:

- **matrix** (Tensor) - 第一个输入 tensor。
- **rhs** (Tensor) - 第二个输入 tensor。
- **adjoint** (bool, 可选) - 表示是否需要在求解前对输入矩阵 *matrix* 做共轭转置, 默认 `False`。

返回:

Tensor

支持平台:

Ascend CPU

样例:

```
>>> import mindspore
>>> matrix = mindspore.tensor([[5., 4.], [3., 1.]])
>>> rhs = mindspore.tensor([[7.], [2.]])
>>> result = mindspore.ops.matrix_solve(matrix, rhs)
>>> print(result)
[[0.14285707]
 [1.5714287 ]]
```

mindspore.ops.matrix_diag

`mindspore.ops.matrix_diag(x, k=0, num_rows=-1, num_cols=-1, padding_value=0, align='RIGHT_LEFT')`
 使用 x 中的元素值作为输出 tensor 的对角线，其余用 `padding_value` 填充。

`num_rows` 和 `num_cols` 为 int32 类型的单值 tensor，若为 -1，则表示输出 tensor 的最内层矩阵是一个方阵。

参数：

- **x** (Tensor) - 输入 tensor。
- **k** (Union[int, Tensor], 可选) - 对角线偏移。正值表示超对角线，负值表示次对角线。当 **k** 是 2 个整数，表示子对角线的上界和下界。默认 0。
- **num_rows** (Union[int, Tensor], 可选) - 输出 tensor 的行数。默认 -1。
- **num_cols** (Union[int, Tensor], 可选) - 输出 tensor 的列数。默认 -1。
- **padding_value** (Union[int, float, Tensor], 可选) - 填充对角线带外区域的数值。默认 0。
- **align** (str, 可选) - 指定超对角线和次对角线的对齐方式。可选 "RIGHT_LEFT"、"LEFT_RIGHT"、"LEFT_LEFT"、"RIGHT_RIGHT"。默认 "RIGHT_LEFT"。
 - "RIGHT_LEFT" 表示将超对角线与右侧对齐（左侧填充行），将次对角线与左侧对齐（右侧填充行）。
 - "LEFT_RIGHT" 表示将超对角线与左侧对齐（右侧填充行），将次对角线与右侧对齐（左侧填充行）。
 - "LEFT_LEFT" 表示将超对角线和次对角线均与左侧对齐（右侧填充行）。
 - "RIGHT_RIGHT" 表示将超对角线和次对角线均与右侧对齐（左侧填充行）。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[8., 9., 0.],
...                       [1., 2., 3.],
...                       [0., 4., 5.]])
>>> k = mindspore.tensor([-1, 1], mindspore.int32)
>>> padding_value = mindspore.tensor(11.)
>>> num_rows = mindspore.tensor(3, mindspore.int32)
>>> num_cols = mindspore.tensor(3, mindspore.int32)
>>> output = mindspore.ops.matrix_diag(x, k, num_rows, num_cols, padding_value, align='LEFT_
↳RIGHT')
>>> print(output)
[[ 1.  8. 11.]
 [ 4.  2.  9.]
 [11.  5.  3.]]
>>> print(output.shape)
(3, 3)
```

mindspore.ops.matrix_diag_part

`mindspore.ops.matrix_diag_part(x, k, padding_value, align='RIGHT_LEFT')`

返回一个 `tensor`，包含输入 `tensor x` 的第 `k[0]` 到 `k[1]` 个对角线中的值，缺失部分使用 `padding_value` 填充。

在 `graph mode` 中，输入 `k` 和 `padding_value` 必须为常量 `tensor`。

参数：

- **x** (`Tensor`) - 输入 `tensor`，其秩不小于 2。
- **k** (`Union[int, Tensor]`, 可选) - 对角线偏移。正值表示超对角线，负值表示次对角线。当 **k** 是 2 个整数，表示子对角线的上界和下界。
- **padding_value** (`Tensor`) - 填充对角线带外区域的数值。
- **align** (`str`, 可选) - 指定超对角线和次对角线的对齐方式。可选 `"RIGHT_LEFT"`、`"LEFT_RIGHT"`、`"LEFT_LEFT"`、`"RIGHT_RIGHT"`。默认 `"RIGHT_LEFT"`。
 - `"RIGHT_LEFT"` 表示将超对角线与右侧对齐（左侧填充行），将次对角线与左侧对齐（右侧填充行）。
 - `"LEFT_RIGHT"` 表示将超对角线与左侧对齐（右侧填充行），将次对角线与右侧对齐（左侧填充行）。
 - `"LEFT_LEFT"` 表示将超对角线和次对角线均与左侧对齐（右侧填充行）。
 - `"RIGHT_RIGHT"` 表示将超对角线和次对角线均与右侧对齐（左侧填充行）。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1., 2., 3., 4.],
...                       [5., 6., 7., 8.],
...                       [9., 8., 7., 6.]])
>>> k = mindspore.tensor([1, 3], mindspore.int32)
>>> output = mindspore.ops.matrix_diag_part(x, k, mindspore.tensor(9.), align='RIGHT_LEFT')
>>> print(output)
[[9. 9. 4.]
 [9. 3. 8.]
 [2. 7. 6.]]
>>> print(output.shape)
(3, 3)
```

mindspore.ops.matrix_set_diag

`mindspore.ops.matrix_set_diag(x, diagonal, k=0, align='RIGHT_LEFT')`

返回一个 `tensor`，使用输入 `diagonal` 中的元素值替换 `x` 矩阵的第 `k[0]` 条到第 `k[1]` 条对角线上的元素值。

参数：

- **x** (`Tensor`) - 输入 `tensor`，其秩不小于 2。
- **diagonal** (`Tensor`) - 输入对角线 `tensor`。
- **k** (`Union[int, Tensor]`, 可选) - 对角线偏移。正值表示超对角线，负值表示次对角线。当 **k** 是 2 个整数，表示子对角线的上界和下界。默认 0。

- **align** (str, 可选) - 可选字符串, 指定超对角线和次对角线的对齐方式。可选 "RIGHT_LEFT"、"LEFT_RIGHT"、"LEFT_LEFT"、"RIGHT_RIGHT"。默认 "RIGHT_LEFT"。
 - "RIGHT_LEFT" 表示将超对角线与右侧对齐 (左侧填充行), 将次对角线与左侧对齐 (右侧填充行)。
 - "LEFT_RIGHT" 表示将超对角线与左侧对齐 (右侧填充行), 将次对角线与右侧对齐 (左侧填充行)。
 - "LEFT_LEFT" 表示将超对角线和次对角线均与左侧对齐 (右侧填充行)。
 - "RIGHT_RIGHT" 表示将超对角线和次对角线均与右侧对齐 (左侧填充行)。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[7., 7., 7., 7.],
...                        [7., 7., 7., 7.],
...                        [7., 7., 7., 7.]])
>>> diagonal = mindspore.tensor([[0., 9., 1.],
...                               [6., 5., 8.],
...                               [1., 2., 3.],
...                               [4., 5., 0.]])
>>> k = mindspore.tensor([-1, 2], mindspore.int32)
>>> align = 'RIGHT_LEFT'
>>> output = ops.matrix_set_diag(x, diagonal, k, align)
>>> print(output)
[[1. 6. 9. 7.]
 [4. 2. 5. 1.]
 [7. 5. 3. 8.]]
>>> print(output.shape)
(3, 4)
```

mindspore.ops.mm

`mindspore.ops.mm(input, mat2)`

计算两个输入的矩阵乘积。

若 *input* 的 shape 为 $(n \times m)$, *mat2* 的 shape 为 $(m \times p)$, *out* 的 shape 为 $(n \times p)$ 。

说明:

- 此函数不能支持广播。若需要可广播的方法, 请参考 `mindspore.ops.matmul()`。
- Ascend 平台, 不支持 float64 类型。

参数:

- **input** (Tensor) - 第一个输入 tensor。
- **mat2** (Tensor) - 第二个输入 tensor。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> out = mindspore.ops.mm(mindspore.ops.ones((2, 3)), mindspore.ops.ones((3, 4)))
>>> print(out.shape)
(2, 4)
```

mindspore.ops.mv

`mindspore.ops.mv(mat, vec)`

实现矩阵 *mat* 和向量 *vec* 相乘。

如果 *mat* 的 shape 为 (N, M) ，*vec* 的 shape 为 M ，则输出为 N 的一维 tensor。

参数:

- **mat** (Tensor) - 输入矩阵。
- **vec** (Tensor) - 输入向量。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.mv(mindspore.tensor([[3., 4.], [1., 6.], [1., 3.]]), mindspore.
↳tensor([1., 2.]))
>>> print(output)
[11. 13. 7.]
```

mindspore.ops.outer

`mindspore.ops.outer(input, vec2)`

计算两个 tensor 的外积。

如果向量 *input* 长度为 n ，*vec2* 长度为 m ，则输出矩阵 shape 为 (n, m) 。

说明: 该函数不支持广播。

参数:

- **input** (Tensor) - 一维输入 tensor。
- **vec2** (Tensor) - 一维输入 tensor。

返回:

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 3])
>>> vec2 = mindspore.tensor([4, 5, 6])
>>> mindspore.ops.outer(input, vec2)
Tensor(shape=[3, 3], dtype=Int64, value=
[[ 4,  5,  6],
 [ 8, 10, 12],
 [12, 15, 18]])
```

mindspore.ops.orgqr

`mindspore.ops.orgqr(input, input2)`

计算 Householder 矩阵的前 N 列。

通常用于计算 `mindspore.ops.geqrf` 返回的正交矩阵 Q 的显式表示。

下面以输入无 batch 维的情况为例：假设输入 *input* 的 shape 经过 Householder 转换之后为 (M, N) 。当 *input* 的对角线被置为 1，*input* 中下三角形的每一列都表示为： w_j ，其中 j 在 $j = 1, \dots, M$ 范围内，此函数返回 Householder 矩阵乘积的前 N 列：

$$H_1 H_2 \dots H_k \quad \text{with} \quad H_j = I_M - \tau_j w_j w_j^H$$

其中： I_M 是 M 维单位矩阵。当 w 是复数的时候， w^H 是共轭转置，否则是一般转置。输出的 shape 与输入 shape 相同。 τ 即输入 *input2*。

参数：

- **input** (Tensor) - 二维或三维输入 tensor，表示 Householder 反射向量, shape $(*, M, N)$ 。
- **input2** (Tensor) - 一维或二维输入 tensor，表示 Householder 反射系数，shape 为 $(*, K)$ ，其中 K 小于等于 N 。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[ -2.0,  -1.0], [ 1.0,  2.0]])
>>> y, tau = mindspore.ops.geqrf(input)
>>> mindspore.ops.orgqr(y, tau)
Tensor(shape=[2, 2], dtype=Float32, value=
[[ -8.94427061e-01,  4.47213590e-01],
 [ 4.47213590e-01,  8.94427180e-01]])
```

mindspore.ops.ormqr

`mindspore.ops.ormqr(input, tau, other, left=True, transpose=False)`

计算一个普通矩阵 *other* 与一个矩阵 Q（由 Householder 反射向量 *input* 和反射系数 *tau* 生成）的乘积。

如果 *left* 为 `True`，计算顺序为 $Q * other$ ，否则，计算顺序为 $other * Q$ 。

参数：

- **input** (Tensor) - 输入 tensor，shape $(*, mn, k)$ ，当 *left* 为 `True` 时，*mn* 的值等于 *m*，否则 *mn* 的值等于 *n*。
- **tau** (Tensor) - 输入 tensor，shape $(*, \min(mn, k))$ 。
- **other** (Tensor) - 输入 tensor，shape $(*, m, n)$ 。
- **left** (bool, 可选) - 计算顺序。默认 `True`。
- **transpose** (bool, 可选) - 是否对矩阵 Q 进行共轭转置变换。默认 `False`。

返回：

Tensor

支持平台：

GPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[-114.6, 10.9, 1.1], [-0.304, 38.07, 69.38], [-0.45, -0.17, 62]]),
...                               mindspore.float32)
>>> tau = mindspore.tensor([1.55, 1.94, 3.0]), mindspore.float32)
>>> other = mindspore.tensor([[-114.6, 10.9, 1.1],
...                             [-0.304, 38.07, 69.38],
...                             [-0.45, -0.17, 62]]), mindspore.float32)
>>> output = mindspore.ops.ormqr(input, tau, other)
>>> print(output)
[[ 63.82713   -13.823125 -116.28614 ]
 [ -53.659264  -28.157839  -70.42702 ]
 [ -79.54292    24.00183   -41.34253 ]]
```

mindspore.ops.pinv

`mindspore.ops.pinv(x, *, atol=None, rtol=None, hermitian=False)`

计算矩阵的伪逆（Moore-Penrose）。

本函数通过 SVD 计算。如果 $x = U * S * V^T$ ，则 *x* 的伪逆为 $x^+ = V * S^+ * U^T$ ， S^+ 为对 *S* 的对角线上的每个非零元素取倒数，零保留在原位。

支持批量矩阵。若 *x* 是批量矩阵，当 *atol* 或 *rtol* 为 `float` 时，则输出具有相同的批量维度。若 *atol* 或 *rtol* 为 `tensor`，则其 *shape* 必须可广播到 *x.svd()* 返回的奇异值的 *shape*。若 *x.shape* 为 (B, M, N) ，*atol* 或 *rtol* 的 *shape* 为 (K, B) ，则输出 *shape* 为 (K, B, N, M) 。

当 *hermitian* 为 `True` 时，暂时仅支持实数域，默认输入 *x* 为实对称矩阵，因此不会在内部检查 *x*，并且在计算中仅使用下三角部分。当 *x* 的奇异值（或特征值范数，*hermitian=True*）小于阈值（ $\max(atol, \sigma \cdot rtol)$ ， σ 为最大奇异值或特征值）时，将其置为零，且在计算中不使用。

若 *rtol* 和 *atol* 均未指定并且 *x* 的 *shape*(*M*, *N*)，则 *rtol* 设置为 $rtol = \max(M, N) * \epsilon$ ， ϵ 为 *x.dtype* 的 *eps* 值。若 *rtol* 未指定且 *atol* 指定值大于零，则 *rtol* 设置为零。

说明：该函数在内部使用 `svd`（或 `eigh`，`hermitian=True`），所以和这些函数具有相同问题，详细信息请参阅 `svd()` 和 `eigh()` 中的警告。

参数：

- **x** (Tensor) - 输入 tensor。shape 为 $(*, M, N)$ ，其中 $*$ 为零或多个批量维度。`hermitian` 为 `True` 时，暂不支持多个批量维度。

关键字参数：

- **atol** (float, Tensor) - 绝对容差值。默认 `None`。
- **rtol** (float, Tensor) - 相对容差值。默认 `None`。
- **hermitian** (bool) - 是否求解 x 为实对称的矩阵。默认 `False`。

输出：

一个 shape 为 $(*, N, M)$ 的 tensor。

支持平台：

CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[4., 0.], [0., 5.]], mindspore.float32)
>>> output = mindspore.ops.pinv(x)
>>> print(output)
[[0.25 0. ]
 [0.   0.2 ]]
```

`mindspore.ops.svd`

`mindspore.ops.svd(input, full_matrices=False, compute_uv=True)`

计算单个或多个矩阵的奇异值分解。

设矩阵 A ，`svd` 返回奇异值 S 、左奇异向量 U 和右奇异向量 V 。满足以下公式：

$$A = U * \text{diag}(S) * V^T$$

参数：

- **input** (Tensor) - 输入 tensor，shape 为 $(*, M, N)$ 。
- **full_matrices** (bool, 可选) - 如果为 `True`，则计算完整的 U 和 V 。否则仅计算前 P 个奇异向量， P 为 M 和 N 中的较小值。默认 `False`。
- **compute_uv** (bool, 可选) - 如果为 `True`，则计算 U 和 V ，否则只计算 S 。默认 `True`。

返回：

如果 `compute_uv` 为 `True`，返回三个 tensor 组成的 tuple(s, u, v)。否则，返回单个 tensor -> s 。

- s 是奇异值 tensor。shape 为 $(*, P)$ 。
- u 是左奇异 tensor。如果 `compute_uv` 为 `False`，该值不会返回。shape 为 $(*, M, P)$ 。如果 `full_matrices` 为 `True`，则 shape 为 $(*, M, M)$ 。

- v 是右奇异 tensor。如果 `compute_uv` 为 `False`，该值不会返回。shape 为 $(*, N, P)$ 。如果 `full_matrices` 为 `True`，则 shape 为 $(*, N, N)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2], [-4, -5], [2, 1]], mindspore.float32)
>>> s, u, v = mindspore.ops.svd(input, full_matrices=True, compute_uv=True)
>>> print(s)
[7.0652843 1.040081 ]
>>> print(u)
[[ 0.30821905 -0.48819482  0.81649697]
 [-0.90613353  0.11070572  0.40824813]
 [ 0.2896955  0.8656849  0.4082479 ]]
>>> print(v)
[[ 0.63863593  0.769509 ]
 [ 0.769509 -0.63863593]]
```

`mindspore.ops.slogdet`

`mindspore.ops.slogdet(input)`

对一个或多个方阵行列式的绝对值取对数，返回其符号和值。

说明：输出的类型是实数，即使 `input` 是复数。

参数：

- `input` (Tensor) - 输入 tensor，shape 为 $(*, M, M)$ 。

返回：

两个 tensor 组成的 tuple，分别为符号和绝对值的对数值。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[1., 2], [3, 4]])
>>> sign, value = mindspore.ops.slogdet(input)
>>> print(sign)
-1.0
>>> print(value)
0.6931472
>>> input = mindspore.tensor([[-4.5, -1.5], [7.0, 6.0]], [[2.5, 0.5], [3.0, 9.0]])
>>> sign, value = mindspore.ops.slogdet(input)
>>> print(sign)
[-1.  1.]
>>> print(value)
[2.8033605 3.0445223]
```

mindspore.ops.trace

`mindspore.ops.trace(input)`

返回 `tensor` 对角线的元素总和。

说明：输入必须是 `Tensor`，复数类型暂不支持。

参数：

- **input** (`Tensor`) - 输入 `tensor`。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[0, 1, 2],
...                           [3, 4, 5],
...                           [6, 7, 8]])
>>> mindspore.ops.trace(input)
Tensor(shape=[], dtype=Int64, value= 12)
```

mindspore.ops.tensor_dot

`mindspore.ops.tensor_dot(x1, x2, axes)`

沿指定轴计算两个张量的点乘。

参数：

- **x1** (`Tensor`) - 输入 `tensor`。
- **x2** (`Tensor`) - 输入 `tensor`。
- **axes** (`Union[int, tuple(int), tuple(tuple(int)), list(list(int))]`) - 指定轴。如果是整数 `k`，则对 `x1` 的后 `k` 个轴和 `x2` 的前 `k` 个轴求和。如果提供的是列表或元组，则 `axes[0]` 指定 `x1` 的轴，`axes[1]` 指定 `x2` 的轴。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> import mindspore
>>> import numpy as np
>>> input_x1 = Tensor(np.ones(shape=[1, 2, 3]), mindspore.float32)
>>> input_x2 = Tensor(np.ones(shape=[3, 1, 2]), mindspore.float32)
>>> output = ops.tensor_dot(input_x1, input_x2, ((0,1), (1,2)))
```

(下页继续)

(续上页)

```
>>> print(output)
[[2. 2. 2]
 [2. 2. 2]
 [2. 2. 2]]
```

mindspore.ops.vander

`mindspore.ops.vander(x, N=None)`

生成一个范德蒙矩阵。返回矩阵的各列是输入各元素的幂。第 i 个输出列是输入向量元素的幂，其幂为 $N - i - 1$ 。

参数：

- **x** (Tensor) - 一维输入 tensor。
- **N** (int, 可选) - 返回矩阵的列数。默认 None，为 $\text{len}(x)$ 。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1., 2., 3., 5.])
>>> output = mindspore.ops.vander(input)
>>> print(output)
[[ 1.  1.  1.  1.]
 [ 8.  4.  2.  1.]
 [27.  9.  3.  1.]
 [125. 25.  5.  1.]]
>>> output = mindspore.ops.vander(input, N=3)
>>> print(output)
[[ 1.  1.  1.]
 [ 4.  2.  1.]
 [ 9.  3.  1.]
 [25.  5.  1.]]
```

mindspore.ops.vecdot

`mindspore.ops.vecdot(x, y, *, axis=-1)`

按指定轴，计算两批向量的点积。

支持广播。

计算公式如下，如果 x 是复数向量， $\bar{x}_i$ 表示向量中元素的共轭；如果 x 是实数向量， $\bar{x}_i$ 表示向量中元素本身。

$$\sum_{i=1}^n \bar{x}_i y_i$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **x** (Tensor) - 第一批 tensor。
- **y** (Tensor) - 第二批 tensor。

关键字参数：

- **axis** (int) - 指定轴。默认 -1 。

返回：

Tensor

说明：当前在 GPU 上不支持复数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 3], [5, 7], [9, 8]])
>>> y = mindspore.tensor([[4, 5], [6, 7], [3, 2]])
>>> mindspore.ops.vecdot(x, y)
Tensor(shape=[3], dtype=Int64, value= [19, 79, 43])
>>> mindspore.ops.vecdot(x, y, axis=0)
Tensor(shape=[2], dtype=Int64, value= [61, 80])
```

2.2.5 谱函数

接口名	概述	支持平台
<code>mindspore.ops.bartlett_window</code>	巴特利特窗口函数。	Ascend GPU CPU
<code>mindspore.ops.blackman_window</code>	布莱克曼窗口函数。	Ascend GPU CPU
<code>mindspore.ops.hamming_window</code>	汉明窗口函数。	Ascend GPU CPU
<code>mindspore.ops.hann_window</code>	汉宁窗口函数。	Ascend GPU CPU
<code>mindspore.ops.kaiser_window</code>	凯瑟窗口函数。	Ascend GPU CPU

`mindspore.ops.bartlett_window`

`mindspore.ops.bartlett_window(window_length, periodic=True, *, dtype=None)`

巴特利特窗口函数。

是一种三角形状的加权函数，通常用于平滑处理或频域分析信号。

$$w[n] = 1 - \left| \frac{2n}{N-1} - 1 \right| = \begin{cases} \frac{2n}{N-1} & \text{if } 0 \leq n \leq \frac{N-1}{2} \\ 2 - \frac{2n}{N-1} & \text{if } \frac{N-1}{2} < n < N \end{cases}$$

其中，*N* 是总的窗口长度 `window_length`，*n* 为小于 *N* 的自然数 [0, 1, ..., N-1]。

参数：

- **window_length** (Tensor) - 窗口的大小。

- **periodic** (bool, 可选) - 如果为 True , 表示返回周期窗口。如果为 False , 表示返回对称窗口。默认 True 。

关键字参数:

- **dtype** (mindspore.dtype, 可选) - 指定数据类型。默认 None 。

返回:

一维 tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> window_length = mindspore.tensor(5)
>>> output = mindspore.ops.bartlett_window(window_length)
>>> print(output)
[0.  0.4 0.8 0.8 0.4]
>>> output = mindspore.ops.bartlett_window(window_length, periodic=False)
>>> print(output)
[0.  0.5 1.  0.5 0. ]
```

mindspore.ops.blackman_window

mindspore.ops.blackman_window (*window_length*, *periodic=True*, *, *dtype=None*)

布莱克曼窗口函数。

常用来为 FFT 截取有限长的信号片段。

$$w[n] = 0.42 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right)$$

其中, N 是总的窗口长度 *window_length* , n 为小于 N 的自然数 $[0, 1, \dots, N-1]$ 。

参数:

- **window_length** (Tensor) - 窗口的大小。
- **periodic** (bool, 可选) - 如果为 True , 表示返回周期窗口。如果为 False , 表示返回对称窗口。默认 True 。

关键字参数:

- **dtype** (mindspore.dtype, 可选) - 指定数据类型。默认 None 。

返回:

一维 tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> window_length = mindspore.tensor(10)
>>> output = mindspore.ops.blackman_window(window_length)
>>> print(output)
```

(下页继续)

(续上页)

```
[-2.9802322e-08  4.0212840e-02  2.0077014e-01  5.0978714e-01
 8.4922993e-01  1.0000000e+00  8.4922981e-01  5.0978690e-01
 2.0077008e-01  4.0212870e-02]
```

`mindspore.ops.hamming_window`

`mindspore.ops.hamming_window(window_length, periodic=True, alpha=0.54, beta=0.46, *, dtype=None)`

汉明窗口函数。

$$w[n] = \alpha\beta \cos\left(\frac{2\pi n}{N-1}\right),$$

其中, N 是总的窗口长度 `window_length`, n 为小于 N 的自然数 $[0, 1, \dots, N-1]$ 。

参数:

- **window_length** (int) - 窗口的大小。
- **periodic** (bool, 可选) - 如果为 `True`, 表示返回周期窗口。如果为 `False`, 表示返回对称窗口。默认 `True`。
- **alpha** (float, 可选) - 系数 α 。默认 `0.54`。
- **beta** (float, 可选) - 系数 β 。默认 `0.46`。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 指定数据类型。默认 `None`。

返回:

一维 `tensor`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> output = mindspore.ops.hamming_window(5)
>>> print(output)
[0.08000001 0.3978522 0.9121478 0.9121478 0.3978522 ]
>>> output = mindspore.ops.hamming_window(5, periodic=False)
>>> print(output)
[0.08000001 0.54          1.          0.54          0.08000001]
```

`mindspore.ops.hann_window`

`mindspore.ops.hann_window(window_length, periodic=True, *, dtype=None)`

汉宁窗口函数。

$$w(n) = \frac{1}{2} - \frac{1}{2} \cos\left(\frac{2\pi n}{M-1}\right), \quad 0 \leq n \leq M-1$$

参数:

- **window_length** (int) - 窗口的大小。
- **periodic** (bool, 可选) - 如果为 `True`, 表示返回周期窗口。如果为 `False`, 表示返回对称窗口。默认 `True`。

关键字参数：

- **dtype** (mindspore.dtype, 可选) - 指定数据类型。默认 None。

返回：

一维 **tensor**。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> output = mindspore.ops.hann_window(5)
>>> print(output)
[0.      0.3454915  0.9045085  0.9045085  0.3454915]
>>> output = mindspore.ops.hann_window(5, periodic=False)
>>> print(output)
[0.  0.5 1.  0.5 0.]
```

mindspore.ops.kaiser_window

mindspore.ops.kaiser_window (*window_length*, *periodic=True*, *beta=12.0*, *, *dtype=None*)

凯瑟窗口函数。

$$w(n) = \frac{I_0\left(\beta\sqrt{1 - \frac{4n^2}{(M-1)^2}}\right)}{I_0(\beta)}$$

n 的范围为

$$-\frac{M-1}{2} \leq n \leq \frac{M-1}{2}$$

其中 I_0 为零阶修正 Bessel 函数， M 代表了 *window_length*。

参数：

- **window_length** (int) - 窗口的大小。
- **periodic** (bool, 可选) - 如果为 True，表示返回周期窗口。如果为 False，表示返回对称窗口。默认 True。
- **beta** (float, 可选) - shape 参数，当 *beta* 变大时，窗口就会变窄。默认 12.0。

关键字参数：

- **dtype** (mindspore.dtype, 可选) - 指定数据类型。默认 None。

返回：

一维 **tensor**。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> output = mindspore.ops.kaiser_window(5)
>>> print(output)
[5.27734413e-05 1.01719688e-01 7.92939834e-01 7.92939834e-01
 1.01719688e-01]
>>> output = mindspore.ops.kaiser_window(5, periodic=False)
>>> print(output)
[5.27734413e-05 2.15672745e-01 1.00000000e+00 2.15672745e-01
 5.27734413e-05]
```

2.3 神经网络层函数

2.3.1 神经网络

接口名	概述	支持平台
<code>mindspore.ops.adaptive_avg_pool1d</code>	对可以看作是由一系列 1D 平面组成的输入 Tensor，应用一维自适应平均池化操作。	Ascend GPU CPU
<code>mindspore.ops.adaptive_avg_pool2d</code>	对一个多平面输入信号执行二维自适应平均池化。	Ascend GPU CPU
<code>mindspore.ops.adaptive_avg_pool3d</code>	对一个多平面输入信号执行三维自适应平均池化。	Ascend GPU CPU
<code>mindspore.ops.adaptive_max_pool1d</code>	对可以看作是由一系列 1D 平面组成的输入 Tensor，应用一维自适应最大池化操作。	Ascend GPU CPU
<code>mindspore.ops.adaptive_max_pool2d</code>	对输入 Tensor，提供二维自适应最大池化操作。	Ascend GPU CPU
<code>mindspore.ops.avg_pool1d</code>	在输入 Tensor 上应用 1D 平均池化，输入 Tensor 可以看作是由一系列 1D 平面组成。	Ascend GPU CPU
<code>mindspore.ops.avg_pool2d</code>	在输入 Tensor 上应用 2D 平均池化，输入 Tensor 可以看作是由一系列 2D 平面组成的。	Ascend GPU CPU
<code>mindspore.ops.avg_pool3d</code>	在输入 Tensor 上应用 3D 平均池化，输入 Tensor 可以看作是由一系列 3D 平面组成的。	Ascend GPU CPU
<code>mindspore.ops.batch_norm</code>	对输入数据进行批量归一化和参数更新。	Ascend GPU CPU
<code>mindspore.ops.bias_add</code>	返回输入 Tensor <i>input_x</i> 与偏置 Tensor <i>bias</i> 之和。	Ascend GPU CPU
<code>mindspore.ops.bidense</code>	对输入 <i>input1</i> 和 <i>input2</i> 应用双线性全连接操作。	Ascend GPU CPU
<code>mindspore.ops.ctc_greedy_decoder</code>	对输入中给定的 logits 执行贪婪解码。	Ascend GPU CPU
<code>mindspore.ops.conv1d</code>	对输入 Tensor 计算一维卷积。	Ascend GPU
<code>mindspore.ops.conv2d</code>	对输入 Tensor 计算二维卷积。	Ascend GPU
<code>mindspore.ops.conv3d</code>	对输入 Tensor 计算三维卷积。	Ascend GPU
<code>mindspore.ops.deformable_conv2d</code>	给定 4D 的 Tensor 输入 <i>x</i> ， <i>weight</i> 和 <i>offsets</i> ，计算一个 2D 的可变形卷积。	Ascend GPU CPU
<code>mindspore.ops.dense</code>	对输入 <i>input</i> 应用全连接操作。	Ascend GPU CPU
<code>mindspore.ops.dropout</code>	在训练期间，以服从伯努利分布的概率 <i>p</i> 随机将输入 Tensor 的某些值归零，起到减少神经元相关性的作用，避免过拟合。	Ascend GPU CPU

下页继续

表 11 – 续上页

<code>mindspore.ops.dropout1d</code>	在训练期间, 以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零 (对于形状为 NCL 的三维 Tensor , 其通道特征图指的是后一维 L 的一维特征图)。	Ascend GPU CPU
<code>mindspore.ops.dropout2d</code>	在训练期间, 以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零 (对于形状为 $NCHW$ 的四维 Tensor , 其通道特征图指的是后两维 HW 形状的二维特征图)。	Ascend GPU CPU
<code>mindspore.ops.dropout3d</code>	在训练期间, 以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零 (对于形状为 $NCDHW$ 的五维 Tensor , 其通道特征图指的是后三维 DHW 形状的三维特征图)。	Ascend GPU CPU
<code>mindspore.ops.embedding</code>	以 <code>input</code> 中的值作为索引, 从 <code>weight</code> 中查询对应的 embedding 向量。	Ascend
<code>mindspore.ops.flatten</code>	沿着从 <code>start_dim</code> 到 <code>end_dim</code> 的维度, 对输入 Tensor 进行展平。	Ascend GPU CPU
<code>mindspore.ops.fold</code>	将提取出的滑动局部区域块还原成更大的输出 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.fractional_max_pool3d</code>	在输入 <code>input</code> 上应用三维分数最大池化。	GPU CPU
<code>mindspore.ops.fused_infer_attention_score</code>	这是一个适配增量和全量推理场景的 FlashAttention 函数, 既可以支持全量计算场景 (PromptFlashAttention), 也可支持增量计算场景 (IncreFlashAttention)。	Ascend
<code>mindspore.ops.speed_fusion_attention</code>	本接口用于实现 self-attention 的融合计算。	Ascend
<code>mindspore.ops.group_norm</code>	在 mini-batch 输入上进行组归一化。	Ascend GPU CPU
<code>mindspore.ops.layer_norm</code>	在 mini-batch 输入上应用层归一化 (Layer Normalization)。	Ascend
<code>mindspore.ops.lp_pool1d</code>	在输入 Tensor 上应用 1D LP 池化运算, 可被视为组成一个 1D 输入平面。	Ascend GPU CPU
<code>mindspore.ops.lp_pool2d</code>	在输入 Tensor 上应用 2D LP 池化运算, 可被视为组成一个 2D 输入平面。	Ascend GPU CPU
<code>mindspore.ops.lrn</code>	局部响应归一化操作 LRN (Local Response Normalization)。	GPU CPU
<code>mindspore.ops.max_pool2d</code>	二维最大值池化。	Ascend GPU CPU
<code>mindspore.ops.max_pool3d</code>	三维最大值池化。	Ascend GPU CPU
<code>mindspore.ops.max_unpool1d</code>	<code>max_pool1d</code> 的逆过程。	Ascend GPU CPU
<code>mindspore.ops.max_unpool2d</code>	<code>max_pool2d</code> 的逆过程。	Ascend GPU CPU
<code>mindspore.ops.max_unpool3d</code>	<code>mindspore.ops.max_pool3d()</code> 的逆过程。	Ascend GPU CPU
<code>mindspore.ops.moe_token_permute</code>	根据 <code>indices</code> 对 <code>tokens</code> 进行排列。	Ascend
<code>mindspore.ops.moe_token_unpermute</code>	根据排序的索引对已排列的标记进行反排列, 并可选择将标记与其对应的概率合并。	Ascend
<code>mindspore.ops.incre_flash_attention</code>	增量推理场景接口。	Ascend
<code>mindspore.ops.prompt_flash_attention</code>	全量推理场景接口。	Ascend
<code>mindspore.ops.flash_attention_score</code>	训练场景下, 使用 FlashAttention 算法实现 self-attention 的计算。	Ascend

下页继续

表 11 – 续上页

<code>mindspore.ops.rms_norm</code>	RmsNorm(Root Mean Square Layer Normalization) 即均方根标准化。	Ascend
<code>mindspore.ops.unfold</code>	从一个 batch 的输入 Tensor 中提取滑动局部块。	Ascend GPU CPU

mindspore.ops.adaptive_avg_pool1d

`mindspore.ops.adaptive_avg_pool1d(input, output_size)`

对可以看作是由一系列 1D 平面组成的输入 Tensor，应用一维自适应平均池化操作。

通常，输入的 shape 为 (N, C, L_{in}) ，`adaptive_avg_pool1d` 输出区域平均值在 L_{in} 区间。输出的 shape 为 (N, C, L_{out}) ，其中 L_{out} 由 `output_size` 定义。

说明： L_{in} 必须能被 `output_size` 整除。

参数：

- **input** (Tensor) - 输入 shape 为 (N, C, L_{in}) ，数据类型为 float16、float32。
- **output_size** (int) - 大小为 L_{out} 。

返回：

Tensor，数据类型与 `input` 相同。输出的 shape 为 (N, C, L_{out}) 。

异常：

- **TypeError** - 如果 `output_size` 不是 int 类型。
- **TypeError** - 如果 `input` 的数据类型不是 float16 或者 float32。
- **ValueError** - 如果 `output_size` 小于 1。
- **ValueError** - 如果 `input` 的维度不等于 3。
- **ValueError** - 如果 `input` 的最后一维尺寸小于 `output_size`。
- **ValueError** - 如果 `input` 的最后一维尺寸不能被 `output_size` 整除。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.random.randint(0, 10, [1, 3, 6]), mindspore.float32)
>>> output = ops.adaptive_avg_pool1d(input, output_size=2)
>>> print(output.shape)
(1, 3, 2)
```

`mindspore.ops.adaptive_avg_pool2d`

`mindspore.ops.adaptive_avg_pool2d(input, output_size)`

对一个多平面输入信号执行二维自适应平均池化。即对于任意输入尺寸，指定输出的尺寸都为 $H * W$ 。但是输入和输出特征的数目保持不变。

输入和输出数据格式可以是“NCHW”和“CHW”。N 是批处理大小，C 是通道数，H 是特征高度，W 是特征宽度。

对于二维的自适应平均池化操作，有如下公式：

$$\begin{aligned}
 h_{start} &= \text{floor}(i * H_{in}/H_{out}) \\
 h_{end} &= \text{ceil}((i + 1) * H_{in}/H_{out}) \\
 w_{start} &= \text{floor}(j * W_{in}/W_{out}) \\
 w_{end} &= \text{ceil}((j + 1) * W_{in}/W_{out}) \\
 \text{Output}(i, j) &= \frac{\sum \text{Input}[h_{start}:h_{end}, w_{start}:w_{end}]}{(h_{end}-h_{start}) * (w_{end}-w_{start})}
 \end{aligned}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - `adaptive_avg_pool2d` 的输入，为三维或四维的 Tensor，数据类型为 float16、float32 或者 float64。
- **output_size** (Union[int, tuple]) - 输出特征图的 size。`output_size` 可以为二元 tuple，表示 (H, W) ，或单个 int，表示 (H, H) 。 H 和 W 可以是 int 或 None，如果是 None，则意味着输出的 size 与输入相同。

返回：

Tensor，数据类型与 `input` 相同。

输出的 shape 为 `input_shape[:len(input_shape) - len(out_shape)] + out_shape`。

$$\text{out\_shape} = \begin{cases} \text{input\_shape}[-2] + \text{output\_size}[1], & \text{if } \text{output\_size} \text{ is } (\text{None}, w); \\ \text{output\_size}[0] + \text{input\_shape}[-1], & \text{if } \text{output\_size} \text{ is } (h, \text{None}); \\ \text{input\_shape}[-2:], & \text{if } \text{output\_size} \text{ is } (\text{None}, \text{None}); \\ (h, h), & \text{if } \text{output\_size} \text{ is } h; \\ (h, w), & \text{if } \text{output\_size} \text{ is } (h, w) \end{cases}$$

异常：

- **ValueError** - 如果 `output_size` 是 tuple，并且 `output_size` 的长度不是 2。
- **ValueError** - 如果 `input` 的维度小于或等于 `output_size` 的维度。
- **TypeError** - 如果 `input` 不是 Tensor。
- **TypeError** - 如果 `input` 的数据类型不是 float16、float32 或者 float64。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1: output_size=(None, 2)

```

(下页继续)

(续上页)

```

>>> input = Tensor(np.array([[[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                             [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                             [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]]),
↳mindspore.float32)
>>> output = ops.adaptive_avg_pool2d(input, (None, 2))
>>> print(output)
[[[1.5 2.5]
  [4.5 5.5]
  [7.5 8.5]]
 [[1.5 2.5]
  [4.5 5.5]
  [7.5 8.5]]
 [[1.5 2.5]
  [4.5 5.5]
  [7.5 8.5]]]
>>> # case 2: output_size=2
>>> output = ops.adaptive_avg_pool2d(input, 2)
>>> print(output)
[[[3. 4.]
  [6. 7.]]
 [[3. 4.]
  [6. 7.]]
 [[3. 4.]
  [6. 7.]]]
>>> # case 3: output_size=(1, 2)
>>> output = ops.adaptive_avg_pool2d(input, (1, 2))
>>> print(output)
[[[4.5 5.5]]
 [[4.5 5.5]]
 [[4.5 5.5]]]

```

mindspore.ops.adaptive_avg_pool3d

`mindspore.ops.adaptive_avg_pool3d(input, output_size)`

对一个多平面输入信号执行三维自适应平均池化。对于任何输入尺寸，指定输出的尺寸都为 (D, H, W) ，但是输入和输出特征的数目不会变化。

假设输入 *input* 最后三维大小分别为 (D_{in}, H_{in}, W_{in}) ，则输出的最后三维大小分别为 $(D_{out}, H_{out}, W_{out})$ ，运算如下：

$$\begin{aligned}
 &\forall \quad od \in [0, D_{out} - 1], oh \in [0, H_{out} - 1], ow \in [0, W_{out} - 1] \\
 &output[od, oh, ow] = \\
 &\quad mean(x[D_{istart} : D_{iend} + 1, H_{istart} : H_{iend} + 1, W_{istart} : W_{iend} + 1]) \\
 &\text{where,} \\
 &\quad D_{istart} = \left\lceil \frac{od * D_{in}}{D_{out}} \right\rceil \\
 &\quad D_{iend} = \left\lfloor \frac{(od+1) * D_{in}}{D_{out}} \right\rfloor \\
 &\quad H_{istart} = \left\lceil \frac{oh * H_{in}}{H_{out}} \right\rceil \\
 &\quad H_{iend} = \left\lfloor \frac{(oh+1) * H_{in}}{H_{out}} \right\rfloor \\
 &\quad W_{istart} = \left\lceil \frac{ow * W_{in}}{W_{out}} \right\rceil \\
 &\quad W_{iend} = \left\lfloor \frac{(ow+1) * W_{in}}{W_{out}} \right\rfloor
 \end{aligned}$$

参数：

- **input** (Tensor) - `adaptive_avg_pool3d` 的输入，是 4D 或者 5D 的 Tensor。
- **output_size** (Union[int, tuple]) - 指定输出特征图的尺寸，可以是 tuple (D, H, W)，也可以是一个 int 值 D 来表示输出尺寸为 (D, D, D)。 D ， H 和 W 可以是 int 值或者 None，其中 None 表示输出大小与对应的输入的大小相同。

返回：

Tensor，与输入 *input* 的数据类型相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的数据类型不是 float16、float32 或者 float64。
- **ValueError** - *input* 维度不是 4D 或者 5D。
- **ValueError** - *output_size* 的值不是正数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1: output_size=(3, 3, 4)
>>> output_size=(3, 3, 4)
>>> input_val = np.random.randn(4, 3, 5, 6, 7)
>>> input = Tensor(input_val, mindspore.float32)
>>> output = ops.adaptive_avg_pool3d(input, output_size)
>>> print(output.shape)
(4, 3, 3, 3, 4)
>>> # case 2: output_size=4
>>> output_size=5
>>> input_val = np.random.randn(2, 3, 8, 6, 12)
>>> input = Tensor(input_val, mindspore.float32)
>>> output = ops.adaptive_avg_pool3d(input, output_size)
>>> print(output.shape)
(2, 3, 5, 5, 5)
>>> # case 3: output_size=(None, 4, 5)
>>> output_size=(None, 4, 5)
>>> input_val = np.random.randn(4, 1, 9, 10, 8)
>>> input = Tensor(input_val, mindspore.float32)
>>> output = ops.adaptive_avg_pool3d(input, output_size)
>>> print(output.shape)
(4, 1, 9, 4, 5)
```


`mindspore.ops.adaptive_max_pool1d`

`mindspore.ops.adaptive_max_pool1d(input, output_size)`

对可以看作是由一系列 1D 平面组成的输入 Tensor，应用一维自适应最大池化操作。

通常，输入的 shape 为 (N, C, L_{in}) ，`adaptive_max_pool1d` 输出区域最大值在 L_{in} 区间。输出的 shape 为 (N, C, L_{out}) ，其中 L_{out} 由 `output_size` 定义。

说明：

- L_{in} 必须能被 `output_size` 整除。
- Ascend 平台 `input` 参数仅支持 float16 类型。

参数：

- **input** (Tensor) - 输入 shape 为 (N, C, L_{in}) ，数据类型为 float16、float32。
- **output_size** (int) - 大小为 L_{out} 。

返回：

Tensor，数据类型与 `input` 相同。输出的 shape 为 (N, C, L_{out}) 。

异常：

- **TypeError** - 如果 `output_size` 不是 int 类型。
- **TypeError** - 如果 `input` 的数据类型不是 float16 或者 float32。
- **ValueError** - 如果 `output_size` 小于 1。
- **ValueError** - 如果 `input` 的维度不等于 3。
- **ValueError** - 如果 `input` 的最后一维尺寸小于 `output_size`。
- **ValueError** - 如果 `input` 的最后一维尺寸不能被 `output_size` 整除。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.random.randint(0, 10, [1, 3, 6]), mindspore.float32)
>>> output = ops.adaptive_max_pool1d(input, output_size=2)
>>> print(output.shape)
(1, 3, 2)
```

mindspore.ops.adaptive_max_pool2d

`mindspore.ops.adaptive_max_pool2d(input, output_size, return_indices=False)`

对输入 **Tensor**，提供二维自适应最大池化操作。即对于输入任何尺寸，指定输出的尺寸都为 $H * W$ 。但是输入和输出特征的数目不会变化。

输入和输出数据格式可以是“NCHW”和“CHW”。N 是批处理大小，C 是通道数，H 是特征高度，W 是特征宽度。运算如下：

$$\begin{aligned}
 h_{start} &= \text{floor}(i * H_{in} / H_{out}) \\
 h_{end} &= \text{ceil}((i + 1) * H_{in} / H_{out}) \\
 w_{start} &= \text{floor}(j * W_{in} / W_{out}) \\
 w_{end} &= \text{ceil}((j + 1) * W_{in} / W_{out}) \\
 \text{Output}(i, j) &= \max \text{Input}[h_{start} : h_{end}, w_{start} : w_{end}]
 \end{aligned}$$

说明：KBK 模式下，`output_size` 不支持 mutable 类型。

参数：

- **input** (Tensor) - adaptive_max_pool2d 的输入，为三维或四维的 Tensor，数据类型为 float16、float32 或者 float64。
- **output_size** (Union[int, tuple]) - 输出特征图的 size。`output_size` 可以为二元 tuple 表示 (H, W) 。或者是单个 int 表示 (H, H) 。 H 和 W 可以是 int 或 None，如果是 None，则意味着输出的 size 与输入相同。
- **return_indices** (bool, 可选) - 如果为 True，输出最大值的索引，默认值为 False。

返回：

Tensor，数据类型和维度与 `input` 相同。

异常：

- **TypeError** - `input` 不是 Tensor。
- **TypeError** - `input` 中的数据不是 float16、float32、float64。
- **TypeError** - `output_size` 不是 int 或者 tuple。
- **TypeError** - `return_indices` 不是 bool。
- **ValueError** - `output_size` 是 tuple，但大小不是 2。
- **ValueError** - `input` 的格式不是“CHW”或者“NCHW”。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1: output_size=(None, 2)
>>> input = Tensor(np.array([[[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                           [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                           [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]]]),
mindspore.float32)
>>> output = ops.adaptive_max_pool2d(input, (None, 2))
>>> print(output)

```

(下页继续)

(续上页)

```

[[[2. 3.]
  [5. 6.]
  [8. 9.]]
 [[2. 3.]
  [5. 6.]
  [8. 9.]]
 [[2. 3.]
  [5. 6.]
  [8. 9.]]]
>>> # case 2: output_size=2
>>> output = ops.adaptive_max_pool2d(input, 2)
>>> print(output)
[[[5. 6.]
  [8. 9.]]
 [[5. 6.]
  [8. 9.]]
 [[5. 6.]
  [8. 9.]]]
>>> # case 3: output_size=(1, 2)
>>> output = ops.adaptive_max_pool2d(input, (1, 2))
>>> print(output)
[[[8. 9.]]
 [[8. 9.]]
 [[8. 9.]]]

```

mindspore.ops.avg_pool1d

`mindspore.ops.avg_pool1d(input_x, kernel_size=1, stride=1, padding=0, ceil_mode=False, count_include_pad=True)`

在输入 Tensor 上应用 1D 平均池化，输入 Tensor 可以看作是由一系列 1D 平面组成。

一般地，输入的 shape 为 (N_{in}, C_{in}, L_{in}) ，输出 (L_{in}) 维度的区域平均值。给定 `kernel_size` 为 $ks = l_{ker}$ 和 `stride` 为 $s = s_0$ ，运算如下：

$$\text{output}(N_i, C_j, l) = \frac{1}{l_{ker}} \sum_{n=0}^{l_{ker}-1} \text{input}(N_i, C_j, s_0 \times l + n)$$

警告： `kernel_size` 取值为 [1, 255] 范围内的正整数，`stride` 的取值为 [1, 63] 范围内的正整数。

参数：

- **input_x** (Tensor) - 输入 shape 为 (N, C_{in}, L_{in}) 的 Tensor。
- **kernel_size** (int, 可选) - 指定池化核尺寸大小。默认 1。
- **stride** (Union(int, tuple[int]), 可选) - 池化操作的移动步长。可以是一个整数或者包含一个整数的 tuple。默认 1。
- **padding** (Union(int, tuple[int]), 可选) - 池化填充长度。可以是一个整数或者包含一个整数的 tuple。默认 0。
- **ceil_mode** (bool, 可选) - 如果为 True，用 ceil 代替 floor 来计算输出的 shape。默认 False。
- **count_include_pad** (bool, 可选) - 如果为 True，平均计算将包括零填充。默认 True。

返回：

Tensor, shape 为 (N, C_{out}, L_{out}) 。

异常：

- **TypeError** - *input_x* 不是一个 Tensor。
- **TypeError** - *kernel_size* 或 *stride* 不是 int。
- **TypeError** - *ceil_mode* 或 *count_include_pad* 不是 bool。
- **ValueError** - *input_x* 的 shape 长度不等于 3。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *padding* 不是 int 或者 tuple 的长度不等于 2。
- **ValueError** - *padding* 的值小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.random.randint(0, 10, [1, 3, 6]), mindspore.float32)
>>> output = ops.avg_pool1d(input_x, kernel_size=6, stride=1)
>>> print(output.shape)
(1, 3, 1)
```

mindspore.ops.avg_pool2d

`mindspore.ops.avg_pool2d(input_x, kernel_size=1, stride=1, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=0)`

在输入 Tensor 上应用 2D 平均池化，输入 Tensor 可以看作是由一系列 2D 平面组成的。

一般地，输入的 shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$ ，输出 (H_{in}, W_{in}) 维度的区域平均值。给定 *kernel_size* 为 (k_h, k_w) 和 *stride*，运算如下：

$$\text{output}(N_i, C_j, h, w) = \frac{1}{k_h * k_w} \sum_{m=0}^{k_h-1} \sum_{n=0}^{k_w-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

警告：

- *kernel_size* 取值为 [1, 255] 范围内的正整数，*stride* 的取值为 [1, 63] 范围内的正整数。

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **input_x** (Tensor) - 输入 shape 为 $(N, C_{in}, H_{in}, W_{in})$ 的 Tensor。
- **kernel_size** (Union[int, tuple[int]]) - 指定池化核尺寸大小，可以是单个整数表示池化核高度和宽度，或者整数 tuple 分别表示池化核高度和宽度。默认 1。

- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，可以是单个整数表示高度和宽度方向的移动步长，或者整数 tuple 分别表示高度和宽度方向的移动步长。默认 1。
- **padding** (Union(int, tuple[int])) - 池化填充长度。可以是一个整数表示在上下左右方向的填充长度，或者包含四个整数的 tuple，分别表示在上下左右方向的填充长度。默认 0。
- **ceil_mode** (bool) - 如果为 True，用 ceil 代替 floor 来计算输出的 shape。默认 False。
- **count_include_pad** (bool) - 如果为 True，平均计算将包括零填充。默认 True。
- **divisor_override** (int) - 如果指定了该值，它将在平均计算中用作除数，否则将使用 *kernel_size* 作为除数。默认 0，表示未指定该值。

返回：

Tensor, shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。

异常：

- **TypeError** - *input_x* 不是一个 Tensor。
- **TypeError** - *kernel_size* 或 *stride* 既不是 int 也不是 tuple。
- **TypeError** - *ceil_mode* 或 *count_include_pad* 不是 bool。
- **TypeError** - *divisor_override* 不是 int。
- **ValueError** - *input_x* 的 shape 长度不等于 4。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *kernel_size* 或 *stride* 是长度不等于 2 的 tuple。
- **ValueError** - *padding* 不是 int 或者 tuple 的长度不等于 4。
- **ValueError** - *padding* 的值小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.arange(1 * 3 * 3 * 4).reshape(1, 3, 3, 4), mindspore.float32)
>>> output = ops.avg_pool2d(x, kernel_size=2, stride=1)
>>> print(output)
[[[ [ 2.5   3.5   4.5]
     [ 6.5   7.5   8.5]]
  [[14.5  15.5  16.5]
   [18.5  19.5  20.5]]
  [[26.5  27.5  28.5]
   [30.5  31.5  32.5]]]]]
```

mindspore.ops.avg_pool3d

`mindspore.ops.avg_pool3d(input_x, kernel_size=1, stride=1, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=0)`

在输入 Tensor 上应用 3D 平均池化，输入 Tensor 可以看作是由一系列 3D 平面组成的。

一般地，输入的 shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ ，输出 (D_{in}, H_{in}, W_{in}) 维度的区域平均值。给定 $kernel\_size$ 为 $ks = (d_{ker}, h_{ker}, w_{ker})$ ， $stride$ 为 $s = (s_0, s_1, s_2)$ ，运算如下：

$$\text{output}(N_i, C_j, d, h, w) = \frac{1}{d_{ker} * h_{ker} * w_{ker}} \sum_{l=0}^{d_{ker}-1} \sum_{m=0}^{h_{ker}-1} \sum_{n=0}^{w_{ker}-1} \text{input}(N_i, C_j, s_0 \times d + l, s_1 \times h + m, s_2 \times w + n)$$

警告：

- $kernel\_size$ 取值为 $[1, 255]$ 范围内的正整数， $stride$ 的取值为 $[1, 63]$ 范围内的正整数。

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **input_x** (Tensor) - 输入 shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 的 Tensor，数据类型为 float16 和 float32。
- **kernel_size** (Union[int, tuple[int]], 可选) - 指定池化核尺寸大小，可以是单个整数表示池化核深度、高度、宽度，或者整数 tuple 分别表示池化核深度、高度、宽度。默认 1。
- **stride** (Union[int, tuple[int]], 可选) - 池化操作的移动步长，可以是单个整数表示深度、高度、宽度方向的移动步长，或者整数 tuple 分别表示深度、高度、深度方向的移动步长。默认 1。
- **padding** (Union(int, tuple[int]), 可选) - 池化填充长度。可以是一个整数表示在头尾上下左右方向的填充长度，或者包含六个整数的 tuple，分别表示在头尾上下左右方向的填充长度。默认 0。
- **ceil_mode** (bool, 可选) - 如果为 True，用 ceil 代替 floor 来计算输出的 shape。默认 False。
- **count_include_pad** (bool, 可选) - 如果为 True，平均计算将包括零填充。默认 True。
- **divisor_override** (int, 可选) - 如果指定了该值，它将在平均计算中用作除数，否则将使用 $kernel\_size$ 作为除数。默认 0，表示未指定该值。

返回：

Tensor, shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ ，数据类型与 $input\_x$ 一致。

异常：

- **TypeError** - $input\_x$ 不是一个 Tensor。
- **TypeError** - $kernel\_size$ 或 $stride$ 或 $padding$ 既不是 int 也不是 tuple。
- **TypeError** - $ceil\_mode$ 或 $count\_include\_pad$ 不是 bool。
- **TypeError** - $divisor\_override$ 不是 int。
- **ValueError** - $input\_x$ 的 shape 长度不等于 5。
- **ValueError** - $kernel\_size$ 或 $stride$ 的值不是正整数。
- **ValueError** - $kernel\_size$ 或 $stride$ 是长度不等于 3 的 tuple。

- **ValueError** - *padding* 是 tuple 时, 其长度不等于 6。
- **ValueError** - *padding* 的值小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.arange(1 * 2 * 2 * 2 * 3).reshape((1, 2, 2, 2, 3)), mindspore.
    ↳float16)
>>> output = ops.avg_pool3d(input_x, kernel_size=2, stride=1)
>>> print(output)
[[[[[ 5.  6.]]]
  [[17. 18.]]]]
```

mindspore.ops.batch_norm

`mindspore.ops.batch_norm(input_x, running_mean, running_var, weight, bias, training=False, momentum=0.1, eps=1e-5)`
对输入数据进行批量归一化和参数更新。

批量归一化广泛应用于卷积神经网络中。此运算对输入应用归一化, 避免内部协变量偏移, 详见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。使用 mini-batch 数据和学习参数进行训练, 学习的参数见如下公式中,

$$y = \frac{x - \text{mean}}{\sqrt{\text{variance} + \epsilon}} * \gamma + \beta$$

其中, γ 为 *weight*, β 为 *bias*, ϵ 为 *eps*, *mean* 为 *x* 的均值, *variance* 为 *x* 的方差。

警告:

- 对于 Atlas 200/300/500 推理产品, 由于平方根指令, 结果精度未能达到 1‰。

说明:

- 如果 *training* 为 False, 则 *running_mean*、*running_var*、*weight* 和 *bias* 是 Tensor。
- 如果 *training* 为 True, 则 *running_mean*、*running_var*、*weight* 和 *bias* 是 Parameter。

参数:

- **input_x** (Tensor) - 数据输入, shape 为 (N, C) 的 Tensor, 数据类型为 float16 或 float32。
- **running_mean** (Union[Tensor, Parameter]) - shape 为 $(C,)$, 具有与 *weight* 相同的数据类型。
- **running_var** (Union[Tensor, Parameter]) - shape 为 $(C,)$, 具有与 *weight* 相同的数据类型。
- **weight** (Union[Tensor, Parameter]) - shape 为 $(C,)$, 数据类型为 float16 或 float32。
- **bias** (Union[Tensor, Parameter]) - shape 为 $(C,)$, 具有与 *weight* 相同的数据类型。
- **training** (bool, 可选) - 如果 *training* 为 True, *running_mean* 和 *running_var* 会在训练过程中进行计算。如果 *training* 为 False, 它们会在推理阶段从 checkpoint 中加载。默认 False。

- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。(例如 $new\_running\_mean = (1 - momentum) * running\_mean + momentum * current\_mean$)。动量值必须为 $[0, 1]$ 。默认 0.1 。
- **eps** (float, 可选) - 添加到分母上的值, 以确保数值稳定性。默认 $1e-5$, 取值范围为 $(0, 1]$ 。

返回:

Tensor, 数据类型与 `shape` 大小与 `input_x` 相同, 其中, `shape` 大小为 (N, C) 。

异常:

- **TypeError** - `training` 不是 bool。
- **TypeError** - `eps` 或 `momentum` 的数据类型不是 float。
- **TypeError** - `input_x`、`weight`、`bias`、`running_mean` 或 `running_var` 不是 Tensor。
- **TypeError** - `input_x` 和 `weight` 的数据类型既不是 float16, 也不是 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input_x = Tensor([[1.0, 2.0], [3.0, 4.0]], mindspore.float32)
>>> running_mean = Tensor([0.5, 1.5], mindspore.float32)
>>> running_var = Tensor([0.1, 0.2], mindspore.float32)
>>> weight = Tensor([2.0, 2.0], mindspore.float32)
>>> bias = Tensor([-1.0, -1.0], mindspore.float32)
>>> output = ops.batch_norm(input_x, running_mean, running_var, weight, bias)
>>> print(output)
[[ 2.1621194  1.2360122]
 [14.810596  10.180061 ]]
```

mindspore.ops.bias_add

`mindspore.ops.bias_add(input_x, bias)`

返回输入 Tensor `input_x` 与偏置 Tensor `bias` 之和。相加前会把偏置 Tensor 广播成与输入 Tensor 的 `shape` 一致。

参数:

- **input_x** (Tensor) - 输入 Tensor。shape 可以有 2~5 个维度。支持数据类型:
 - Ascend/CPU: all Number type。
 - GPU: float16、float32、int8。
- **bias** (Tensor) - 偏置 Tensor, shape 为 (C) 。C 必须与 `input_x` 的通道维度 C 相同。其数据类型与 `input_x` 一致。

返回:

Tensor, shape 和数据类型与 `input_x` 相同。

异常:

- **TypeError** - `input_x` 或 `bias` 不是 Tensor。
- **TypeError** - `input_x` 与 `bias` 的数据类型不一致。
- **TypeError** - `input_x` 的维度不在 $[2, 5]$ 范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.arange(6).reshape((2, 3)), mindspore.float32)
>>> bias = Tensor(np.random.random(3).reshape((3)), mindspore.float32)
>>> output = ops.bias_add(input_x, bias)
>>> print(output.shape)
(2, 3)
```

mindspore.ops.bidense

`mindspore.ops.bidense(input1, input2, weight, bias=None)`

对输入 *input1* 和 *input2* 应用双线性全连接操作。双线性全连接函数定义如下:

$$output = x_1^T A x_2 + b$$

其中, x_1 代表 *input1*, x_2 代表 *input2*, A 代表 *weight*, b 代表 *bias*。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input1** (Tensor) - 输入 Tensor, shape 是 $(*, in1\_channels)$, 其中 $*$ 表示任意的附加维度, 除最后一维外的维度与 *input2* 保持一致。
- **input2** (Tensor) - 输入 Tensor, shape 是 $(*, in2\_channels)$, 其中 $*$ 表示任意的附加维度, 除最后一维外的维度与 *input1* 保持一致。
- **weight** (Tensor) - 输入 Tensor 的权重, shape 是 $(out\_channels, in1\_channels, in2\_channels)$ 。
- **bias** (Tensor, 可选) - 添加在输出结果的偏差, shape 是 $(out\_channels)$ 或 $()$ 。默认 `None`, 偏差为 0。

返回:

Tensor, shape 是 $(*, out\_channels)$, 其中 $*$ 表示任意的附加维度。输出 Tensor 除最后一维外其他维度与所有输入 Tensor 保持一致。

异常:

- **TypeError** - *input1* 不是 Tensor。
- **TypeError** - *input2* 不是 Tensor。
- **TypeError** - *weight* 不是 Tensor。
- **TypeError** - *bias* 不是 Tensor。
- **ValueError** - 如果除了最后一维, *input1* 其他维度与 *input2* 有不同。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input1 = mindspore.Tensor([[[-1.1283, 1.2603],
...                             [0.0214, 0.7801],
...                             [-1.2086, 1.2849]], mindspore.float32)
>>> input2 = mindspore.Tensor([[[-0.4631, 0.3238, 0.4201],
...                             [0.6215, -1.0910, -0.5757],
...                             [-0.7788, -0.0706, -0.7942]], mindspore.float32)
>>> weight = mindspore.Tensor([[[-0.3132, 0.9271, 1.1010],
...                             [0.6555, -1.2162, -0.2987]],
...                             [[1.0458, 0.5886, 0.2523],
...                             [-1.3486, -0.8103, -0.2080]],
...                             [[1.1685, 0.5569, -0.3987],
...                             [-0.4265, -2.6295, 0.8535]],
...                             [[0.6948, -1.1288, -0.6978],
...                             [0.3511, 0.0609, -0.1122]]], mindspore.float32)
>>> output = ops.bidense(input1, input2, weight)
>>> print(output)
[[-2.0612743 0.5581219 0.22383511 0.8667302]
 [1.4476739 0.12626505 1.6552988 0.21297503]
 [0.6003161 2.912046 0.5590313 -0.35449564]]

```

mindspore.ops.ctc_greedy_decoder

`mindspore.ops.ctc_greedy_decoder(inputs, sequence_length, merge_repeated=True)`

对输入中给定的 logits 执行贪婪解码。

说明：在 Ascend 平台上，目前不支持配置 `merge_repeated=False`。

参数：

- **inputs** (Tensor) - shape: $(max\_time, batch\_size, num\_classes)$ ，数据类型必须是 float32 或者 float64。
 $num\_classes$ 为 $num\_labels + 1$ classes，其中 num_labels 表示实际标签的个数，空标签默认使用 $num\_classes - 1$ 。
- **sequence_length** (Tensor) - shape: $(batch\_size,)$ ，数据类型必须是 int32，并且 Tensor 中的数值必须小于等于 max_time 。
- **merge_repeated** (bool) - True 表示返回的结果中会合并重复的类。默认值为 True。

返回：

- **decoded_indices** (Tensor) - shape: $(total\_decoded\_outputs, 2)$ ，数据类型为 int64。
- **decoded_values** (Tensor) - shape: $(total\_decoded\_outputs,)$ ，数据类型为 int64。
- **decoded_shape** (Tensor) - shape: $(batch\_size, max\_decoded\_length)$ ，数据类型为 int64。
- **log_probability** (Tensor) - shape: $(batch\_size, 1)$ ，包含序列的对数概率，其数据类型与 *inputs* 保持一致。

异常：

- **TypeError** - *merge_repeated* 不是一个布尔值。
- **ValueError** - *inputs* 的 shape 长度不等于 3。
- **ValueError** - *sequence_length* 的 shape 长度不等于 1。
- **ValueError** - *sequence_length* 中的数值大于 *max_time*。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> inputs = Tensor(np.array([[[0.6, 0.4, 0.2], [0.8, 0.6, 0.3]],
...                           [[0.0, 0.6, 0.0], [0.5, 0.4, 0.5]]]), mindspore.float32)
>>> sequence_length = Tensor(np.array([2, 2]), mindspore.int32)
>>> decoded_indices, decoded_values, decoded_shape, log_probability = ops.ctc_greedy_
↳decoder(inputs,
...
↳sequence_length)
>>> print(decoded_indices)
[[0 0]
 [0 1]
 [1 0]]
>>> print(decoded_values)
[0 1 0]
>>> print(decoded_shape)
[2 2]
>>> print(log_probability)
[[-1.2]
 [-1.3]]
```

mindspore.ops.conv1d

`mindspore.ops.conv1d(input, weight, bias=None, stride=1, pad_mode='valid', padding=0, dilation=1, groups=1)`

对输入 `Tensor` 计算一维卷积。通常输入的 `shape` 为 (N, C_{in}, L_{in}) ，其中 N 为 batch size， C 为通道数， L 为输入序列的长度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中，`bias` 为输出偏置，`ccor` 为 `cross-correlation` 操作，`weight` 为卷积核的值，`X` 为输入的特征图。

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中，`bias( $C_{out_j}$ )` 为第 j 个输出通道的偏置，`weight( $C_{out_j}, k$ )` 表示第 j 个卷积核在第 k 个输入通道的卷积核切片，`X( $N_i, k$ )` 为特征图第 i 个 batch 第 k 个输入通道的切片。卷积核 `shape` 为 `(kernel_size)`，其中 `kernel_size` 是卷积核的宽度。若考虑到输入输出通道以及 `groups`，则完整卷积核的 `shape` 为 `( $C_{out}, C_{in}/groups, kernel\_size$ )`，其中 `groups` 是分组卷积时在通道上分割输入 `x` 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#) 和 [ConvNets](#)。

说明：在 Ascend 平台上，目前只支持深度卷积场景下的分组卷积运算。也就是说，当 `groups>1` 的场景下，必须要满足 `$C_{in} = C_{out} = groups$` 的约束条件。

参数：

- **input** (Tensor) - 输入 Tensor, shape 为 (N, C_{in}, L_{in}) 。
- **weight** (Tensor) - 卷积核的值, 其 shape 为 $(C_{out}, C_{in}/groups, kernel\_size)$ 。
- **bias** (Tensor, 可选) - 偏置 Tensor, shape 为 (C_{out}) 的 Tensor。如果 *bias* 是 None, 将不会添加偏置。默认 None。
- **stride** (Union(int, tuple[int]), 可选) - 卷积核移动的步长, 数据类型为 int 或 1 个 int 组成的 tuple。表示在宽度方向的移动步长。默认 1。
- **pad_mode** (str, 可选) - 指定填充模式。取值为 "same", "valid", 或 "pad"。默认 "valid"。
 - "same": 输出的宽度与输入整除 *stride* 后的值相同。填充将被均匀地添加到两侧, 剩余填充量将被添加到维度末端。若设置该模式, *padding* 的值必须为 0。
 - "valid": 在不填充的前提下返回有效计算所得的输出。不满足计算的多余像素会被丢弃。如果设置此模式, 则 *padding* 的值必须为 0。
 - "pad": 对输入 *input* 进行填充。在输入上填充 *padding* 大小的 0。如果设置此模式, *padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int]), 可选) - 当 *pad_mode* 为 "pad" 时, 指定在输入 *input* 的宽度方向上填充的数量。数据类型为 int 或包含 1 个 int 组成的 tuple。表示宽度方向的 *padding* 数量 (左右两边均为该值)。值必须大于等于 0, 默认 0。
- **dilation** (Union(int, tuple[int]), 可选) - 卷积核膨胀尺寸。可以为单个 int, 或者由一个 int 组成的 tuple。假设 $dilation = (d0,)$, 则卷积核在宽度方向间隔 $d0 - 1$ 个元素进行采样。取值范围为 $[1, L]$ 。默认 1。
- **groups** (int, 可选) - 将过滤器拆分为组。默认 1。

返回：

Tensor, 卷积后的值。shape 为 (N, C_{out}, L_{out}) 。要了解不同的填充模式如何影响输出 shape, 请参考 [mindspore.nn.Conv1d](#) 以获取更多详细信息。

异常：

- **TypeError** - *stride*、*padding* 或 *dilation* 既不是 int 也不是 tuple。
- **TypeError** - *groups* 不是 int。
- **TypeError** - *bias* 不是 Tensor。
- **ValueError** - *bias* 的 shape 不是 (C_{out}) 。
- **ValueError** - *stride* 或 *dilation* 小于 1。
- **ValueError** - *pad_mode* 不是 "same"、"valid" 或 "pad"。
- **ValueError** - *padding* 是一个长度不等于 1 的 tuple。
- **ValueError** - *pad_mode* 不等于 "pad" 时, *padding* 大于 0。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.arange(64).reshape((4, 4, 4)), mindspore.float32)
>>> weight = Tensor(np.arange(8).reshape((2, 2, 2)), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> bias = Tensor([-0.12345, 2.7683], mindspore.float32)
>>> output = ops.conv1d(x, weight, pad_mode='pad', padding=(1,), bias=bias, groups=2)
>>> print(output.shape)
(4, 2, 5)
```

mindspore.ops.conv2d

`mindspore.ops.conv2d(input, weight, bias=None, stride=1, pad_mode='valid', padding=0, dilation=1, groups=1)`

对输入 `Tensor` 计算二维卷积。通常输入的 `shape` 为 $(N, C_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C 为通道数， H 为特征图的高度， W 为特征图的宽度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中，`bias` 为输出偏置，`ccor` 为 `cross-correlation` 操作，`weight` 为卷积核的值， X 为输入的特征图。

i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。

j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。

k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中，`bias`(C_{out_j}) 为第 j 个输出通道的偏置，`weight`(C_{out_j}, k) 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。卷积核 `shape` 为 $(\text{kernel\_size}[0], \text{kernel\_size}[1])$ ，其中 `kernel_size[0]` 和 `kernel_size[1]` 是卷积核的高度和宽度。若考虑到输入输出通道以及 `groups`，则完整卷积核的 `shape` 为 $(C_{out}, C_{in}/\text{groups}, \text{kernel\_size}[0], \text{kernel\_size}[1])$ ，其中 `groups` 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#) 以及 [ConvNets](#)。

说明：在 Ascend 平台上，目前只支持深度卷积场景下的分组卷积运算。也就是说，当 `groups>1` 的场景下，必须要满足 $C_{in} = C_{out} = \text{groups}$ 的约束条件。

参数：

- **input** (Tensor) - `shape` 为 $(N, C_{in}, H_{in}, W_{in})$ 的 Tensor。
- **weight** (Tensor) - `shape` 为 $(C_{out}, C_{in}/\text{groups}, \text{kernel\_size}[0], \text{kernel\_size}[1])$ ，则卷积核的大小为 $(\text{kernel\_size}[0], \text{kernel\_size}[1])$ 。
- **bias** (Tensor, 可选) - 偏置 Tensor，`shape` 为 (C_{out}) 的 Tensor。如果 `bias` 是 `None`，将不会添加偏置。默认 `None`。
- **stride** (Union(int, tuple[int]), 可选) - 卷积核移动的步长，数据类型为 `int`，或者由两个或四个 `int` 组成的 `tuple`。一个 `int` 表示在高度和宽度方向的移动步长均为该值。两个 `int` 组成的 `tuple` 分别表示在高度和宽度方向的移动步长。默认 1。
- **pad_mode** (str, 可选) - 指定填充模式。取值为 `"same"`，`"valid"`，或 `"pad"`。默认 `"valid"`。
 - `"same"`：输出的高度和宽度分别与输入整除 `stride` 后的值相同。填充将被均匀地添加到高和宽的两侧，剩余填充量将被添加到维度末端。若设置该模式，`padding` 的值必须为 0。
 - `"valid"`：在不填充的前提下返回有效计算所得的输出。不满足计算的多余像素会被丢弃。如果设置此模式，则 `padding` 的值必须为 0。

- "pad": 对输入 *input* 进行填充。在输入的高度和宽度方向上填充 *padding* 大小的 0。如果设置此模式, *padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int]), 可选) - 输入 *input* 的高度和宽度方向上填充的数量。数据类型为 int 或包含 2 个 int 组成的 tuple。如果 *padding* 是一个 int, 那么上、下、左、右的填充都等于 *padding*。如果 *padding* 是一个有 2 个 int 组成的 tuple, 那么上、下的填充为 *padding[0]*, 左、右的填充为 *padding[1]*。值必须大于或等于 0, 默认 0。
- **dilation** (Union(int, tuple[int]), 可选) - 卷积核元素间的间隔。数据类型为 int 或由 2 个 int 组成的 tuple。若 $k > 1$, 则卷积核间隔 k 个元素进行采样。垂直和水平方向上的 k , 其取值范围分别为 $[1, H]$ 和 $[1, W]$ 。默认 1。
- **groups** (int, 可选) - 将过滤器拆分为组。默认 1。

返回:

Tensor, 卷积后的值。shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。要了解不同的填充模式如何影响输出 shape, 请参考 [mindspore.nn.Conv2d](#) 以获取更多详细信息。

异常:

- **TypeError** - *stride*、*padding* 或 *dilation* 既不是 int 也不是 tuple。
- **TypeError** - *groups* 不是 int。
- **TypeError** - *bias* 不是 Tensor。
- **ValueError** - *bias* 的 shape 不是 (C_{out}) 。
- **ValueError** - *stride* 或 *dilation* 小于 1。
- **ValueError** - *pad_mode* 不是 "same"、"valid" 或 "pad"。
- **ValueError** - *padding* 是一个长度不等于 2 的 tuple 或 list。
- **ValueError** - *pad_mode* 不等于 "pad" 时, *padding* 大于 0。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.ones([10, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> output = ops.conv2d(x, weight)
>>> print(output.shape)
(10, 32, 30, 30)
```

mindspore.ops.conv3d

`mindspore.ops.conv3d(input, weight, bias=None, stride=1, pad_mode='valid', padding=0, dilation=1, groups=1)`

对输入 Tensor 计算三维卷积。通常，输入 Tensor 的 shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C 为通道数， D, H, W 分别为特征图的深度、高度和宽度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{\text{out}_j}, k), X(N_i, k))$$

其中， bias 为输出偏置， ccor 为 [cross-correlation](#) 操作， weight 为卷积核的值， X 为输入的特征图。

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中， $\text{bias}(C_{\text{out}_j})$ 为第 j 个输出通道的偏置， $\text{weight}(C_{\text{out}_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。

卷积核 shape 为 $(\text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ ，其中 $\text{kernel\_size}[0]$ 、 $\text{kernel\_size}[1]$ 和 $\text{kernel\_size}[2]$ 分别是卷积核的深度、高度和宽度。若考虑到输入输出通道以及 groups ，则完整卷积核的 shape 为 $(C_{out}, C_{in}/\text{groups}, \text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ ，其中 groups 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

说明：

1. 在 Ascend 平台上，目前只支持 $\text{groups} = 1$ 。
2. 在 Ascend 平台上，目前只支持 $\text{dilation} = 1$ 。

参数：

- **input** (Tensor) - shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor。
- **weight** (Tensor) - shape 为 $(C_{out}, C_{in}/\text{groups}, \text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ ，则卷积核的大小为 $(\text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ 。
- **bias** (Tensor, 可选) - 偏置 Tensor，shape 为 (C_{out}) 的 Tensor。如果 bias 是 None，将不会添加偏置。默认 None。
- **stride** (Union[int, tuple[int]], 可选) - 卷积核移动的步长，可以为单个 int 或三个 int 组成的 tuple。一个 int 表示在深度、高度和宽度方向的移动步长均为该值。三个 int 组成的 tuple 分别表示在深度、高度和宽度方向的移动步长。默认 1。
- **pad_mode** (str, 可选) - 指定填充模式。取值为 "same"，"valid"，或 "pad"。默认 "valid"。
 - "same": 输出的高度和宽度分别与输入整除 stride 后的值相同。填充将被均匀地添加到高和宽的两侧，剩余填充量将被添加到维度末端。若设置该模式， padding 的值必须为 0。
 - "valid": 在不填充的前提下返回有效计算所得的输出。不满足计算的多余像素会被丢弃。如果设置此模式，则 padding 的值必须为 0。
 - "pad": 对输入 input 进行填充。在输入的高度和宽度方向上填充 padding 大小的 0。如果设置此模式， padding 必须大于或等于 0。

- **padding** (Union[int, tuple[int], list[int]], 可选) - 输入 *input* 的深度、高度和宽度方向上填充的数量。数据类型为 int 或包含 3 个 int 组成的 tuple。如果 *padding* 是一个 int, 那么前、后、上、下、左、右的填充都等于 *padding*。如果 *padding* 是一个有 3 个 int 组成的 tuple, 那么前、后的填充为 *padding[0]*, 上、下的填充为 *padding[1]*, 左、右的填充为 *padding[2]*。值必须大于等于 0, 默认 0。
- **dilation** (Union[int, tuple[int]], 可选) - 卷积核膨胀尺寸。数据类型为 int 或由 3 个 int 组成的 tuple: (*dilation_d*, *dilation_h*, *dilation_w*)。目前在 Ascend 后端, 只支持该值为 1。若 $k > 1$, 则卷积核间隔 k 个元素进行采样。前后、垂直和水平方向上, 其取值范围分别为 [1, D]、[1, H] 和 [1, W]。默认 1。
- **groups** (int, 可选) - 将过滤器拆分的组数。默认 1。

返回:

Tensor, 卷积后的值。shape 为 ($N, C_{out}, D_{out}, H_{out}, W_{out}$)。

pad_mode 为 "same" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in}}{\text{stride}[0]} \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in}}{\text{stride}[1]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in}}{\text{stride}[2]} \right\rceil \end{aligned}$$

pad_mode 为 "valid" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in} - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1)}{\text{stride}[0]} + 1 \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in} - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1)}{\text{stride}[1]} + 1 \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} - \text{dilation}[2] \times (\text{kernel\_size}[2] - 1)}{\text{stride}[2]} + 1 \right\rceil \end{aligned}$$

pad_mode 为 "pad" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in} + \text{padding}[0] + \text{padding}[0] - (\text{dilation}[0] - 1) \times \text{kernel\_size}[0] - 1}{\text{stride}[0]} + 1 \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in} + \text{padding}[1] + \text{padding}[1] - (\text{dilation}[1] - 1) \times \text{kernel\_size}[1] - 1}{\text{stride}[1]} + 1 \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} + \text{padding}[2] + \text{padding}[2] - (\text{dilation}[2] - 1) \times \text{kernel\_size}[2] - 1}{\text{stride}[2]} + 1 \right\rceil \end{aligned}$$

异常:

- **TypeError** - *stride*、*padding* 或 *dilation* 既不是 int 也不是 tuple。
- **TypeError** - *groups* 不是 int。
- **TypeError** - *bias* 不是 Tensor。
- **ValueError** - *bias* 的 shape 不是 (C_{out})。
- **ValueError** - *stride* 或 *dilation* 小于 1。
- **ValueError** - *pad_mode* 不是 "same"、"valid" 或 "pad"。
- **ValueError** - *padding* 是一个长度不等于 3 的 tuple。
- **ValueError** - *pad_mode* 不等于 "pad" 时, *padding* 大于 0。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.ones([16, 3, 10, 32, 32]), mindspore.float16)
>>> weight = Tensor(np.ones([32, 3, 4, 3, 3]), mindspore.float16)
>>> output = ops.conv3d(x, weight, pad_mode="same", padding=0, stride=1, dilation=1,
↳groups=1)
>>> print(output.shape)
(16, 32, 10, 32, 32)
>>> output = ops.conv3d(x, weight, pad_mode="valid", padding=0, stride=1, dilation=1,
↳groups=1)
>>> print(output.shape)
(16, 32, 7, 30, 30)
>>> output = ops.conv3d(x, weight, pad_mode="pad", padding=(2, 1, 1), stride=1, dilation=1,
↳groups=1)
>>> print(output.shape)
(16, 32, 11, 32, 32)
```

mindspore.ops.deformable_conv2d

`mindspore.ops.deformable_conv2d(x, weight, offsets, kernel_size, strides, padding, bias=None, dilations=(1, 1, 1, 1), groups=1, deformable_groups=1, modulated=True)`

给定 4D 的 Tensor 输入 x , $weight$ 和 $offsets$, 计算一个 2D 的可变形卷积。可变形卷积运算可以表达如下:

可变形卷积 v1:

$$y(p) = \sum_{k=1}^K w_k \cdot x(p + p_k + \Delta p_k)$$

可变形卷积 v2:

$$y(p) = \sum_{k=1}^K w_k \cdot x(p + p_k + \Delta p_k) \cdot \Delta m_k$$

其中 Δp_k 和 Δm_k 分别为第 k 个位置的可学习偏移和调制标量。细节请参考论文 [Deformable ConvNets v2: More Deformable, Better Results](#) 和 [Deformable Convolutional Networks](#)。

参数:

- **x** (Tensor) - 一个四维 Tensor, 表示输入图像。数据格式为“NCHW”, shape 为 $(N, C_{in}, H_{in}, W_{in})$ 。Dtype 为 float16 或 float32。
- **weight** (Tensor) - 一个四维 Tensor, 表示可学习的滤波器。数据类型必须与 x 相同, shape 为 $(C_{out}, C_{in}/groups, H_f, W_f)$ 。
- **offsets** (Tensor) - 一个四维 Tensor, 存储 x 和 y 坐标的偏移, 以及可变形卷积的输入掩码 mask。数据格式为“NCHW”, shape 为 $(batch, 3 * deformable\_groups * H_f * W_f, H_{out}, W_{out})$, 注意其中 C 维度的存储顺序为 (offset_x, offset_y, mask)。数据类型必须与 x 相同。
- **kernel_size** (tuple[int]) - 一个包含两个整数的元组, 表示卷积核的大小。
- **strides** (tuple[int]) - 一个包含四个整数的元组, 表示对于输入的每个维度的滑动窗口步长。其维度顺序依据 x 的数据格式, 对应 N 和 C 维度的值必须设置成 1。
- **padding** (tuple[int]) - 一个包含四个整数的元组, 表示沿 (上, 下, 左, 右) 四个方向往输入填充的像素点个数。

- **bias** (Tensor, 可选) - 一个一维 Tensor, 表示加到卷积输出的偏置参数。shape 为 (C_{out}) 。默认值为 None。
- **dilations** (tuple[int], 可选) - 一个包含四个整数的元组, 表示对于输入的每个维度的膨胀系数。其维度顺序依据 x 的数据格式, 对应 N 和 C 维度的值必须设置成 1。默认值为 $(1, 1, 1, 1)$ 。
- **groups** (int, 可选) - 一个 int32 类型的整数, 表示从输入通道到输出通道的阻塞连接数。输入通道数和输出通道数必须都能被 *groups* 整除。默认值为 1。
- **deformable_groups** (int, 可选) - 一个 int32 类型的整数, 表示可变形卷积组数。输入通道数必须能被 *deformable_groups* 整除。默认值为 1。
- **modulated** (bool, 可选) - 指定可变形二维卷积的版本。True 表示 v2, False 表示 v1。当前只支持设置为 v2 版本。默认值为 True。

返回:

Tensor, 一个四维 Tensor, 表示输出特征图。数据类型与 x 相同, 数据格式为“NCHW”, shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。

$$H_{out} = \left\lfloor \frac{H_{in} + padding[0] + padding[1] - (H_f - 1) \times dilations[2] - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + padding[2] + padding[3] - (W_f - 1) \times dilations[3] - 1}{stride[1]} + 1 \right\rfloor$$

异常:

- **TypeError** - 如果 *strides*, *padding*, *kernel_size* 或者 *dilations* 不是一个整数元组。
- **TypeError** - 如果 *modulated* 不是一个布尔值。
- **ValueError** - 如果 *strides*, *padding*, *kernel_size* 或者 *dilations* 的元组不是期望的大小。
- **ValueError** - 如果 *strides* 或者 *dilations* 对应 N 和 C 维度的值不为 1。
- **ValueError** - 如果 *modulated* 的值不是 True。

警告: 这是一个实验性 API, 后续可能修改或删除。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.ones((4, 3, 10, 10)), mstype.float32)
>>> kh, kw = 3, 3
>>> weight = Tensor(np.ones((5, 3, kh, kw)), mstype.float32)
>>> offsets = Tensor(np.ones((4, 3 * kh * kw, 8, 8)), mstype.float32)
>>> output = ops.deformable_conv2d(x, weight, offsets, (kh, kw), (1, 1, 1, 1), (0, 0, 0, 0))
>>> print(output.shape)
(4, 5, 8, 8)
```

mindspore.ops.dense

`mindspore.ops.dense(input, weight, bias=None)`

对输入 *input* 应用全连接操作。全连接定义为：

$$output = input * weight^T + bias$$

警告：

- 这是一个实验性 API，后续可能修改或删除。
- 在 Ascend 硬件平台下，设置 PYNATIVE 或 KBK 模式时，如果 *bias* 不是 1D，*input* 不可以大于 6D。

参数：

- **input** (Tensor) - 输入 Tensor，shape 是 $(*, in\_channels)$ ，其中 $*$ 表示任意的附加维度。
- **weight** (Tensor) - 输入 Tensor 的权重，shape 是 $(out\_channels, in\_channels)$ 或 $(in\_channels)$ 。
- **bias** (Tensor, 可选) - 添加在输出结果的偏差，shape 是 $(out\_channels)$ 或 $()$ 。默认 None，偏差为 0。

返回：

输出结果，shape 由 *input* 和 *weight* 的 shape 决定。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *weight* 不是 Tensor。
- **TypeError** - *bias* 不是 Tensor。
- **RuntimeError** - 在 Ascend 硬件平台下，设置 PYNATIVE 或 KBK 模式时，*bias* 不是 1D 且 *input* 大于 6D。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input = Tensor([[-1., 1., 2.], [-3., -3., 1.]], mindspore.float32)
>>> weight = Tensor([[ -2., -2., -2.], [0., -1., 0.]], mindspore.float32)
>>> bias = Tensor([0., 1.], mindspore.float32)
>>> output = ops.dense(input, weight, bias)
>>> print(output)
[[-4.  0.]
 [10.  4.]]
```

mindspore.ops.dropout

`mindspore.ops.dropout(input, p=0.5, training=True, seed=None)`

在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些值归零，起到减少神经元相关性的作用，避免过拟合。并且训练过程中返回值会乘以 $\frac{1}{1-p}$ 。在推理过程中，此层返回与 *input* 相同的 Tensor。

警告：Ascend 后端不支持随机数重现功能，*seed* 参数不起作用。

参数：

- **input** (Tensor) - 输入 Tensor，shape 为 $(*, N)$ ，数据类型为 float16、float32 或 float64。
- **p** (float, 可选) - 输入神经元丢弃概率，数值范围在 0 到 1 之间。例如， $p=0.1$ ，删除 10% 的神经元。默认 0.5。
- **training** (bool, 可选) - 若为 True 则启用 dropout 功能。默认 True。
- **seed** (int, 可选) - 随机数生成器的种子，必须是非负数，默认 None，即 *seed* 的数量为 0。

返回：

- **output** (Tensor) - 归零后的 Tensor，shape 和数据类型与 *input* 相同。

异常：

- **TypeError** - *p* 不是 float。
- **TypeError** - *input* 的数据类型不是 float16、float32 或 float64。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input = Tensor(((20, 16), (50, 50)), mindspore.float32)
>>> output = ops.dropout(input, p=0.5)
>>> print(output.shape)
(2, 2)
```

mindspore.ops.dropout1d

`mindspore.ops.dropout1d(input, p=0.5, training=True)`

在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零（对于形状为 NCL 的三维 Tensor，其通道特征图指的是最后一维 L 的一维特征图）。例如，在批处理输入中 i_th 批， j_th 通道的 *input*[i, j] 1D Tensor 是一个待处理数据。每个通道将会独立依据伯努利分布概率 p 来确定是否被清零。

论文 [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#) 中提出了该技术，并证明其能有效地减少过度拟合，防止神经元共适应。更多详细信息，请参见 [Improving neural networks by preventing co-adaptation of feature detectors](#)。

dropout1d 可以提高通道特征映射之间的独立性。

参数：

- **input** (Tensor) - 一个 shape 为 (N, C, L) 或 (C, L) 的 3D 或 2D Tensor, 其中 N 是批处理大小, C 是通道数, L 是特征长度。其数据类型应为 int8、int16、int32、int64、float16、float32 或 float64。
- **p** (float, 可选) - 通道的丢弃概率, 介于 0 和 1 之间, 例如 $p = 0.8$, 意味着 80% 的清零概率。默认 0.5。
- **training** (bool, 可选) - 是否启用 dropout 功能。默认 True。

返回:

Tensor, 输出, 具有与输入 *input* 相同的 shape 和数据类型。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *p* 的数据类型不是 float。
- **ValueError** - *p* 值不在 $[0.0, 1.0]$ 之间。
- **ValueError** - *input* 的维度不是 2D 或 3D。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.random.randn(4, 3), mindspore.float32)
>>> output = ops.dropout1d(input_x, 0.5)
>>> print(output.shape)
(4, 3)
```

mindspore.ops.dropout2d

`mindspore.ops.dropout2d(input, p=0.5, training=True)`

在训练期间, 以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零 (对于形状为 $NCHW$ 的四维 Tensor, 其通道特征图指的是后两维 HW 形状的二维特征图)。例如, 在批处理输入中 i_th 批, j_th 通道的 $input[i, j]$ 2D Tensor 是一个待处理数据。每个通道将会独立依据伯努利分布概率 p 来确定是否被清零。论文 [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#) 中提出了该技术, 并证明其能有效地减少过度拟合, 防止神经元共适应。更多详细信息, 请参见 [Improving neural networks by preventing co-adaptation of feature detectors](#)。

dropout2d 可以提高通道特征映射之间的独立性。

参数:

- **input** (Tensor) - 一个形状为 (N, C, H, W) 的 4D Tensor, 其中 N 是批处理大小, C 是通道数, H 是特征高度, W 是特征宽度。其数据类型应为 int8、int16、int32、int64、float16、float32 或 float64。
- **p** (float) - 通道的丢弃概率, 取值范围为 $[0.0, 1.0]$ 。例如 $p = 0.8$ 表示 80% 的清零概率。默认 0.5。
- **training** (bool) - 如果为 True, 则执行对 *input* 的某些通道概率清零的操作, 否则不执行。默认 True。

返回:

- Tensor, 输出, 具有与输入 *input* 相同的形状和数据类型。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的数据类型不是 int8、int16、int32、int64、float16、float32 或 float64。

- **TypeError** - p 不是 float 类型。
- **ValueError** - p 不在 $[0.0, 1.0]$ 范围内。
- **ValueError** - $input$ 的维度不是 4。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.ones([2, 1, 2, 3]), mindspore.float32)
>>> output = ops.dropout2d(input, 0.5)
>>> print(output.shape)
(2, 1, 2, 3)
```

mindspore.ops.dropout3d

`mindspore.ops.dropout3d(input, p=0.5, training=True)`

在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零（对于形状为 $NCDHW$ 的五维 Tensor，其通道特征图指的是后三维 DHW 形状的三维特征图）。例如，在批处理输入中 i_th 批， j_th 通道的 $input[i, j]$ 3D Tensor 是一个待处理数据。每个通道将会独立依据伯努利分布概率 p 来确定是否被清零。

`dropout3d` 可以提高通道特征映射之间的独立性。

参数：

- **input** (Tensor) - 一个形状为 (N, C, D, H, W) 的 5D Tensor，其中 N 是批处理大小， C 是通道数， D 是特征深度， H 是特征高度， W 是特征宽度。其数据类型应为 int8、int16、int32、int64、float16、float32 或 float64。
- **p** (float) - 通道的丢弃概率，介于 0 和 1 之间，例如 $p = 0.8$ ，意味着 80% 的清零概率。默认 0.5。
- **training** (bool) - 如果 training 为 True，则执行对 $input$ 的某些通道概率清零的操作，否则，不执行。默认 True。

返回：

- Tensor，输出，具有与输入 $input$ 相同的形状和数据类型。

异常：

- **TypeError** - $input$ 不是 Tensor。
- **TypeError** - $input$ 的数据类型不是 int8、int16、int32、int64、float16、float32 或 float64。
- **TypeError** - p 的数据类型不是 float。
- **ValueError** - p 值不在 $[0.0, 1.0]$ 之间。
- **ValueError** - $input$ 的维度不等于 5。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.ones([2, 1, 2, 1, 2]), mindspore.float32)
>>> output = ops.dropout3d(input, 0.5)
>>> print(output.shape)
(2, 1, 2, 1, 2)
```

mindspore.ops.embedding

`mindspore.ops.embedding` (`input`, `weight`, `padding_idx=None`, `max_norm=None`, `norm_type=2.0`, `scale_grad_by_freq=False`)
以 `input` 中的值作为索引，从 `weight` 中查询对应的 `embedding` 向量。

警告：在 Ascend 后端，`input` 的值非法将导致不可预测的行为。

参数：

- **input** (Tensor) - 用于检索的索引输入。取值范围： $[0, \text{weight.shape}[0])$ 。
- **weight** (Union[Parameter, Tensor]) - 用于检索的数据。必须是 2D 输入。
- **padding_idx** (int, 可选) - 如果给定非 None 值，则 `weight` 中对应行在反向计算中不会更新。取值范围： $[-\text{weight.shape}[0], \text{weight.shape}[0])$ 。默认值 None。
- **max_norm** (float, 可选) - 如果给定非 None 值，则先求出 `input` 指定位置的 `weight` 的 p-范数结果 `result` (p 的值通过 `norm_type` 指定)，然后对 `result > max_norm` 位置的 `weight` 进行原地更新，更新公式： $\frac{\text{max\_norm}}{\text{result}+1e^{-7}}$ 。默认值 None。
- **norm_type** (float, 可选) - 指定 p-范数计算中的 p 值。默认值 2.0。
- **scale_grad_by_freq** (bool, 可选) - 如果值为 True，则反向梯度值会按照 `input` 中索引值重复的次数进行缩放。默认值 False。

返回：

Tensor，数据类型与 `weight` 保持一致，shape 为 $(*\text{input.shape}, \text{weight.shape}[1])$ 。

异常：

- **ValueError** - `padding_idx` 的取值不在有效范围。
- **ValueError** - `weight.shape` 非法。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, Parameter, ops
>>> input = Tensor([[1, 0, 1, 1], [0, 0, 1, 0]])
>>> weight = Parameter(np.random.randn(3, 3).astype(np.float32))
>>> output = ops.embedding(input, weight, max_norm=0.4)
>>> print(output)
[[[ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01],
```

(下页继续)

(续上页)

```
[ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01],
[ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01],
[ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01]],
[[ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01],
[ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01],
[ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01],
[ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01]]]
```

mindspore.ops.flatten

`mindspore.ops.flatten(input, order='C', *, start_dim=1, end_dim=-1)`

沿着从 *start_dim* 到 *end_dim* 的维度，对输入 Tensor 进行展平。

参数：

- **input** (Tensor) - 输入 Tensor。
- **order** (str, 可选) - 仅支持 'C' 和 'F'。'C' 表示按行优先顺序 (C 风格) 展平，'F' 表示按列优先顺序 (Fortran 风格) 展平。默认 'C'。

关键字参数：

- **start_dim** (int, 可选) - 要展平的第一个维度。默认 1。
- **end_dim** (int, 可选) - 要展平的最后一个维度。默认 -1。

返回：

Tensor。如果没有维度被展平，返回原始的输入 *input*，否则返回展平后的 Tensor。如果 *input* 是零维 Tensor，将会返回一个一维 Tensor。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *order* 不是 string 类型。
- **ValueError** - *order* 是 string 类型，但不是 'C' 或 'F'。
- **TypeError** - *start_dim* 或 *end_dim* 不是 int 类型。
- **ValueError** - 规范化后，*start_dim* 大于 *end_dim*。
- **ValueError** - *start_dim* 或 *end_dim* 不在 $[-input.dim, input.dim-1]$ 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.ones(shape=[1, 2, 3, 4]), mindspore.float32)
>>> output = ops.flatten(input_x)
>>> print(output.shape)
(1, 24)
```


mindspore.ops.fold

`mindspore.ops.fold(input, output_size, kernel_size, dilation=1, padding=0, stride=1)`

将提取出的滑动局部区域块还原成更大的输出 Tensor。

考虑一个 batch 的输入 Tensor，其 shape 为 $(N, C \times \prod(\text{kernel\_size}), L)$ ，其中 N 是批次维度， $C \times \prod(\text{kernel\_size})$ 为每个滑块内值的总数量（一个滑块有 $\prod(\text{kernel\_size})$ 个空间位置，每个位置都包含一个 C 通道的向量），共有 L 个这样的滑块：

$$L = \prod_d \left\lfloor \frac{\text{output\_size}[d] + 2 \times \text{padding}[d] - \text{dilations}[d] \times (\text{kernel\_size}[d] - 1) - 1}{\text{strides}[d]} + 1 \right\rfloor,$$

其中， d 遍历所有的空间维度。

因此，在最后一个维度（列维度）上 *output* 包含特定块内的所有值。

dilation，*padding* 和 *stride* 决定了滑块如何被取出。

警告：

- 输入是三维 Tensor，其 shape 为 $(N, C \times \prod(\text{kernel\_size}), L)$ 。
- 输出是四维 Tensor，其 shape 为 $(N, C, \text{output\_size}[0], \text{output\_size}[1], \dots)$ 。

参数：

- **input** (Tensor) - 三维 Tensor，支持的数据类型：float16、float32、float64、complex64 和 complex128。
- **output_size** (Tensor) - 包含两个 int 元素的一维 Tensor。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 滑动窗口的大小。如果类型为 tuple 或者 list，则存在两个 int 元素分别对应 kernel 的高度与宽度；如果类型为 int，则 kernel 的高度与宽度均为该值。
- **dilation** (Union[int, tuple[int], list[int]])，可选 - 窗口的空洞卷积的扩充率。如果类型为 tuple 或者 list，则存在两个 int 元素分别作用于滑窗的高度和宽度；如果类型为 int，则这个值作用于滑窗的高度和宽度。默认 1。
- **padding** (Union[int, tuple[int], list[int]])，可选 - 滑窗的隐式零填充量。如果类型为 tuple 或者 list，则存在的两个 int 元素分别为滑窗的高度和宽度方向的填充量；如果类型为 int，则高度和宽度方向的填充量均为这个 int 值。默认 0。
- **stride** (Union[int, tuple[int], list[int]])，可选 - 空间维度上滑动的步长。如果类型为 tuple 或者 list，则存在的两个 int 元素分别为滑窗的高度和宽度方向上的步长；如果类型为 int，则高度和宽度方向上的步长均为这个 int 值。默认 1。

返回：

Tensor，数据类型与 *input* 相同。

异常：

- **TypeError** - 如果 *kernel_size*、*stride*、*dilation*、*padding* 的数据类型不是 int、tuple 或者 list。
- **ValueError** - 如果 *kernel_size*、*dilation*、*stride* 包含元素的值小于等于 0 或者元素个数大于 2。
- **ValueError** - 如果 *padding* 包含元素的值小于零或者元素个数大于 2。
- **ValueError** - 如果 $\text{input.shape}[1] \neq \text{kernel\_size}[0] * \text{kernel\_size}[1]$ 。
- **ValueError** - 如果 $\text{input.shape}[2]$ 与计算出的滑块数量不匹配。

支持平台：

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(input_data=np.random.rand(16, 64, 25), dtype=mstype.float32)
>>> output_size = Tensor(input_data=[8, 8], dtype=mstype.int32)
>>> output = ops.fold(x, output_size, [2, 2], [2, 2], [2, 2], [2, 2])
>>> print(output.shape)
(16, 16, 8, 8)
```

mindspore.ops.fractional_max_pool3d

`mindspore.ops.fractional_max_pool3d(input, kernel_size, output_size=None, output_ratio=None, return_indices=False, _random_samples=None)`

在输入 *input* 上应用三维分数最大池化。输出 Tensor 的 *shape* 可以由 *output_size* 或 *output_ratio* 确定，步长由 *_random_samples* 随机生成。当同时设置 *output_size* 和 *output_ratio* 时，*output_size* 生效。*output_size* 和 *output_ratio* 不能同时为 *None*。

分数最大池化的详细描述在 [Fractional MaxPooling by Ben Graham](#)。

输入输出的数据格式可以是“NCDHW”。其中，N 是批次大小，C 是通道数，D 是特征深度，H 是特征高度，W 是特征宽度。

警告：这是一个实验性 API，后续可能修改或删除。

参数:

- **input** (Tensor) - 4D 或 5D 的 Tensor，支持的数据类型：float16、float32、double。支持 *shape* 为 $(N, C, D_{in}, H_{in}, W_{in})$ 或 $(C, D_{in}, H_{in}, W_{in})$ 。
- **kernel_size** (Union[int, tuple[int]]) - 指定池化核尺寸大小，如果为 int，则代表池化核的深度，高和宽。如果为 tuple，其值必须包含三个正 int 值分别表示池化核的深度，高和宽。取值必须为正 int。
- **output_size** (Union[int, tuple[int]])，可选 - 目标输出大小。如果是 int，则表示输出目标的深、高和宽。如果是 tuple，其值必须包含三个 int 值分别表示目标输出的深、高和宽。默认 *None*。
- **output_ratio** (Union[float, tuple[float]])，可选 - 目标输出 *shape* 与输入 *shape* 的比率。通过输入 *shape* 和 *output_ratio* 确定输出 *shape*。支持数据类型：float16、float32、double，数值范围 (0, 1)。默认 *None*。
- **return_indices** (bool，可选) - 是否返回最大值的索引值。默认 *False*。
- **_random_samples** (Tensor，可选) - 随机步长。支持的数据类型：float16、float32、double。*shape* 为 $(N, C, 3)$ 或 $(1, C, 3)$ 的 Tensor。数值范围 [0, 1)。默认 *None*，*_random_samples* 的值由区间 [0, 1) 上的均匀分布随机生成。

返回:

- **y** (Tensor) - 3D 分数最大池化的输出，是一个 Tensor。数据类型和 *input* 相同，*shape* 是 $(N, C, D_{out}, H_{out}, W_{out})$ 或 $(C, D_{out}, H_{out}, W_{out})$ 。其中， $(D_{out}, H_{out}, W_{out}) = output\_size$ 或 $(D_{out}, H_{out}, W_{out}) = output\_ratio * (D_{in}, H_{in}, W_{in})$ 。
- **argmax** (Tensor) - 仅当 *return_indices* 为 *True* 时，输出最大池化的索引值。*shape* 和输出 *y* 一致。

异常:

- **TypeError** - *input* 不是四维或五维 Tensor。
- **TypeError** - *_random_samples* 不是三维 Tensor。

- **TypeError** - *input* 数据类型不是 float16、float32、double。
- **TypeError** - *_random_samples* 数据类型不是 float16、float32、double。
- **TypeError** - *argmax* 数据类型不是 int32、int64。
- **TypeError** - *_random_samples* 和 *input* 数据类型不相同。
- **ValueError** - *output_shape* 不是长度为 3 的 tuple。
- **ValueError** - *kernel_size* 不是长度为 3 的 tuple。
- **ValueError** - *output_shape* 和 *kernel_size* 不是正数。
- **ValueError** - *output_size* 和 *output_ratio* 同时为 *None*。
- **ValueError** - *input* 和 *_random_samples* 的第一维度大小不相等。
- **ValueError** - *input* 和 *_random_samples* 第二维度大小不相等。
- **ValueError** - *_random_samples* 第三维度大小不是 3。

支持平台：

GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]))
...     .reshape([1, 1, 2, 2, 4]), mstype.float32)
>>> _random_samples = Tensor(np.array([0.7, 0.7, 0.7]).reshape([1, 1, 3]), mstype.float32)
>>> output, argmax = ops.fractional_max_pool3d(x, kernel_size=(1, 1, 1), output_size=(1, 1, 3),
...     _random_samples=_random_samples, return_
...     indices=True)
>>> print(output)
[[[[[13. 14. 16.]]]]]
>>> print(argmax)
[[[[[12 13 15]]]]]
>>> output, argmax = ops.fractional_max_pool3d(x, kernel_size=(1, 1, 1), output_ratio=(0.5, 0.5, 0.5),
...     _random_samples=_random_samples, return_
...     indices=True)
>>> print(output)
[[[[[13. 16.]]]]]
>>> print(argmax)
[[[[[12 15]]]]]
```

mindspore.ops.fused_infer_attention_score

```
mindspore.ops.fused_infer_attention_score(query, key, value, *, pse_shift=None, atten_mask=None,
      actual_seq_lengths=None, actual_seq_lengths_kv=None,
      dequant_scale1=None, quant_scale1=None, dequant_scale2=None,
      quant_scale2=None, quant_offset2=None, antiquant_scale=None,
      antiquant_offset=None, key_antiquant_scale=None,
      key_antiquant_offset=None, value_antiquant_scale=None,
      value_antiquant_offset=None, block_table=None,
      query_padding_size=None, kv_padding_size=None,
      key_shared_prefix=None, value_shared_prefix=None,
      actual_shared_prefix_len=None, num_heads=1, scale=1.0,
      pre_tokens=2147483647, next_tokens=2147483647,
      input_layout='BSH', num_key_value_heads=0, sparse_mode=0,
      inner_precise=1, block_size=0, antiquant_mode=0,
      key_antiquant_mode=0, value_antiquant_mode=0,
      softmax_lse_flag=False)
```

这是一个适配增量和全量推理场景的 FlashAttention 函数，既可以支持全量计算场景（PromptFlashAttention），也可支持增量计算场景（IncreFlashAttention）。当 Query 矩阵的 S（Q_S）为 1，进入 IncreFlashAttention 分支，其余场景进入 PromptFlashAttention 分支。

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d}})V$$

警告：

- 这是一个实验性 API，后续可能修改或删除。
- 对于 Ascend，当前仅支持 Atlas A2 训练系列产品/Atlas 800I A2 推理产品。

说明：

- query、key、value 数据排布格式支持从多种维度解读，如下所示：
 - B, Batch size。表示输入样本批量大小。
 - S, Sequence length。表示输入样本序列长度。S1 表示 query 的序列长度，S2 表示 key/value 的序列长度。
 - H, Head size。表示隐藏层的大小。
 - N, Head nums。表示多头数。
 - D, Head dims。表示隐藏层最小的单元尺寸，且满足 $D = H/N$ 。

参数：

- **query** (Tensor) - attention 结构的 query 输入，数据类型支持 float16、bfloat16 或 int8，shape 为 (B, S, H) ， (B, N, S, D) 或 (B, S, N, D) 。
- **key** (Union[Tensor, tuple[Tensor], list[Tensor]]) - attention 结构的 key 输入，数据类型支持 float16、bfloat16 或 int8，shape 为 (B, S, H) ， (B, N, S, D) 或 (B, S, N, D) 。
- **value** (Union[Tensor, tuple[Tensor], list[Tensor]]) - attention 结构的 value 输入，数据类型支持 float16、bfloat16 或 int8，shape 为 (B, S, H) ， (B, N, S, D) 或 (B, S, N, D) 。

关键字参数：

- **pse_shift** (Tensor, 可选) - 位置编码参数, 数据类型支持 float16、bfloat16, 默认值为 None。
 - 当 Q_S 不为 1 时, 如果 pse_shift 为 float16 类型, 则此时的 query 要求为 float16 或 int8 类型。如果 pse_shift 为 bfloat16 类型, 则此时的 query 要求为 bfloat16 类型。输入 shape 类型需为 (B, N, Q_S, KV_S) 或 $(1, N, Q\_S, KV\_S)$, 其中 Q_S 为 query 的 S 轴 shape 值, KV_S 为 key 和 value 的 S 轴 shape 值。对于 pse_shift 的 KV_S 为非 32 对齐的场景, 建议填充到 32 字节来提高性能, 多余部分的填充值不做要求。
 - 当 Q_S 为 1 时, 如果 pse_shift 为 float16 类型, 则此时的 query 要求为 float16 类型。如果 pse_shift 为 bfloat16 类型, 则此时的 query 要求为 bfloat16 类型。输入 shape 类型需为 $(B, N, 1, KV\_S)$ 或 $(1, N, 1, KV\_S)$, 其中 KV_S 为 key 和 value 的 S 轴 shape 值。对于 pse_shift 的 KV_S 为非 32 对齐的场景, 建议填充到 32 字节来提高性能, 多余部分的填充值不做要求。
- **atten_mask** (Tensor, 可选) - 对 query 乘 key 的结果进行掩码, 数据类型支持 int8、uint8 和 bool。对于每个元素, 0 表示保留, 1 表示屏蔽。默认值为 None。
 - Q_S 不为 1 时建议 shape 输入为 Q_S, KV_S; B, Q_S, KV_S; 1, Q_S, KV_S; B, 1, Q_S, KV_S; 1, 1, Q_S, KV_S。
 - Q_S 为 1 时建议 shape 输入为 B, KV_S; B, 1, KV_S; B, 1, 1, KV_S。
- **actual_seq_lengths** (Union[tuple[int], list[int], Tensor], 可选) - 描述入参 query 的实际序列长度, 数据类型支持 int64。如果该参数不指定, 可以传入 None, 表示和 query 的 S 轴 shape 值相同。限制: 该入参中每个 batch 的有效序列长度应该不大于 query 中对应 batch 的序列长度, Q_S 为 1 时该参数无效。默认值为 None。
- **actual_seq_lengths_kv** (Union[tuple[int], list[int], Tensor], 可选) - 描述入参 key/value 的实际序列长度, 数据类型支持 int64。如果该参数不指定, 可以传入 None, 表示和 key/value 的 S 轴 shape 值相同。限制: 该入参中每个 batch 的有效序列长度应该不大于 key/value 中对应 batch 的序列长度, Q_S 为 1 时该参数无效。默认值为 None。
- **dequant_scale1** (Tensor, 可选) - BMM1 后面反量化的量化因子, 数据类型支持 uint64。支持 per-tensor 模式。如不使用该功能时可传入 None。默认值为 None。
- **quant_scale1** (Tensor, 可选) - BMM2 前面量化的量化因子, 数据类型支持 float32。支持 per-tensor 模式。如不使用该功能时可传入 None。默认值为 None。
- **dequant_scale2** (Tensor, 可选) - BMM2 后面量化的量化因子, 数据类型支持 uint64。支持 per-tensor 模式。如不使用该功能时可传入 None。默认值为 None。
- **quant_scale2** (Tensor, 可选) - 输出量化的量化因子, 数据类型支持 float32、float16。支持 per-tensor、per-channel 模式。如不使用该功能时可传入 None。默认值为 None。
- **quant_offset2** (Tensor, 可选) - 输出量化的量化偏移, 数据类型支持 float32、float16。支持 per-tensor、per-channel 模式。如不使用该功能时可传入 None。默认值为 None。

输入为 int8, 输出为 int8 的场景: 入参 dequant_scale1、quant_scale1、dequant_scale2、quant_scale2 需要同时存在, quant_offset2 可选, 不传时默认为 0。

- 输出为 int8, quant_scale2 和 quant_offset2 为 per-channel 时, 暂不支持左 padding、Ring Attention 或者 D 非 32Byte 对齐的场景。
- 输出为 int8 时, 暂不支持 sparse 为 band 且 pre_tokens/next_tokens 为负数。
- 输出为 int8, 入参 quant_offset2 传入非空指针和非空 tensor 值, 并且 sparse_mode、pre_tokens 和 next_tokens 满足以下条件, 矩阵会存在某几行不参与计算的情况, 导致计算结果误差, 该场景会拦截 (解决方案: 如果希望该场景不被拦截, 需要在 FIA 接口外部做后量化操作, 不在 FIA 接口内部使用):
 - * sparse_mode = 0, atten_mask 如果非空指针, 每个 batch actual_seq_lengths - actual_seq_lengths_kv - pre_tokens > 0 或 next_tokens < 0 时, 满足拦截条件;
 - * sparse_mode = 1 或 2, 不会出现满足拦截条件的情况;
 - * sparse_mode = 3, 每个 batch actual_seq_lengths_kv - actual_seq_lengths < 0, 满足拦截条件;
 - * sparse_mode = 4, pre_tokens < 0 或每个 batch next_tokens + actual_seq_lengths_kv - actual_seq_lengths < 0 时, 满足拦截条件。

输入为 int8, 输出为 float16 的场景: 入参 dequant_scale1、quant_scale1、dequant_scale2 需要同时存在。

输入全为 float16 或 bfloat16, 输出为 int8 的场景: 入参 quant_scale2 需存在, quant_offset2 可选, 不传时默认为 0。

入参 quant_scale2 和 quant_offset2 支持 per-tensor/per-channel 两种格式和 float32/bfloat16 两种数据类型。若传入 quant_offset2, 需保证其类型和 shape 信息与 quant_scale2 一致。当输入为 bfloat16 时, 同时支持 float32 和 bfloat16, 否则仅支持 float32。per-channel 格式, 当输出 layout 为 BSH 时, 要求 quant_scale2 所有维度的乘积等于 H; 其他 layout 要求乘积等于 N*D。(建议输出 layout 为 BSH 时, quant_scale2 shape 传入 (1, 1, H) 或 (H); 输出为 BNSD 时, 建议传入 (1, N, 1, D) 或 (N, D); 输出为 BSND 时, 建议传入 (1, 1, N, D) 或 (N, D))。

- **antiquant_scale** (Tensor, 可选) - 表示反量化因子, 数据类型支持 float16、float32 或 bfloat16。Q_S 大于 1 时只支持 float16。支持 per-tensor、per-channel、per-token 模式。默认值为 None。
- **antiquant_offset** (Tensor, 可选) - 表示反量化偏移, 数据类型支持 float16、float32 或 bfloat16。Q_S 大于 1 时只支持 float16。支持 per-tensor、per-channel、per-token 模式。默认值为 None。

antiquant_scale 和 antiquant_offset 参数约束:

– 支持 per-channel、per-tensor 和 per-token 三种模式:

- * per-channel 模式: 两个参数 BNSD 场景下 shape 为 (2, N, 1, D), BSND 场景下 shape 为 (2, N, D), BSH 场景下 shape 为 (2, H), 其中, 2 分别对应到 key 和 value, N 为 num_key_value_heads。参数数据类型和 query 数据类型相同, antiquant_mode 置 0。
- * per-tensor 模式: 两个参数的 shape 均为 (2), 数据类型和 query 数据类型相同, antiquant_mode 置 0。
- * per-token 模式: 两个参数的 shape 均为 (2, B, S), 数据类型固定为 float32, antiquant_mode 置 1。

– 支持对称量化和非对称量化:

- * 非对称量化模式下, antiquant_scale 和 antiquant_offset 参数需同时存在。
- * 对称量化模式下, antiquant_offset 可以为空 (即 None); 当 antiquant_offset 参数为空时, 执行对称量化, 否则执行非对称量化。

- **key_antiquant_scale** (Tensor, 可选) - kv 伪量化参数分离时表示 key 的反量化因子, 数据类型支持 float16、float32 或 bfloat16。支持 per-tensor、per-channel、per-token 模式。默认值为 None。Q_S 大于 1 时该参数无效。
- **key_antiquant_offset** (Tensor, 可选) - kv 伪量化参数分离时表示 key 的反量化偏移, 数据类型支持 float16、float32 或 bfloat16。支持 per-tensor、per-channel、per-token 模式。默认值为 None。Q_S 大于 1 时该参数无效。
- **value_antiquant_scale** (Tensor, 可选) - kv 伪量化参数分离时表示 value 的反量化因子, 数据类型支持 float16、float32 或 bfloat16。支持 per-tensor、per-channel、per-token 模式。默认值为 None。Q_S 大于 1 时该参数无效。
- **value_antiquant_offset** (Tensor, 可选) - kv 伪量化参数分离时表示 value 的反量化偏移, 数据类型支持 float16、float32 或 bfloat16。支持 per-tensor、per-channel、per-token 模式。默认值为 None。Q_S 大于 1 时该参数无效。
- **block_table** (Tensor, 可选) - PageAttention 中 KV 存储使用的 block 映射表, 数据类型支持 int32。如不使用该功能时可传入 None。默认值为 None。Q_S 大于 1 时该参数无效。
- **query_padding_size** (Tensor, 可选) - query 填充尺寸, 数据类型支持 int64。表示 query 中每个 batch 的数据是否右对齐, 且右对齐的个数是多少。默认值为 None。仅支持 Q_S 大于 1, 其余场景该参数无效。
- **kv_padding_size** (Tensor, 可选) - key/value 填充尺寸, 数据类型支持 int64。表示 key/value 中每个 batch 的数据是否右对齐, 且右对齐的个数是多少。默认值为 None。仅支持 Q_S 大于 1, 其余场景该参数无效。
- **key_shared_prefix** (Tensor, 可选) - key 的公共前缀输入。预留参数, 暂未启用。默认值为 None。

- **value_shared_prefix** (Tensor, 可选) - value 的公共前缀输入。预留参数，暂未启用。默认值为 None。
- **actual_shared_prefix_len** (Union[tuple[int], list[int], Tensor], 可选) - 描述公共前缀的实际长度。预留参数，暂未启用。默认值为 None。
- **num_heads** (int, 可选) - query 的 head 个数，当 input_layout 为 BNSD 时和 query 的 N 轴 shape 值相同。默认值为 1。
- **scale** (double, 可选) - 缩放系数，作为计算流中 Muls 的 scalar 值。通常，其值为 $1.0/\sqrt{d}$ 。默认值为 1.0。
- **pre_tokens** (int, 可选) - 用于稀疏计算，表示 attention 需要和前多少个 token 计算关联。默认值为 2147483647。Q_S 为 1 时该参数无效。
- **next_tokens** (int, 可选) - 用于稀疏计算，表示 attention 需要和后多少个 token 计算关联。默认值为 2147483647。Q_S 为 1 时该参数无效。
- **input_layout** (str, 可选) - 指定输入 query、key、value 的数据排布格式。当前支持 BSH、BSND、BNSD、BNSD_BSND。当 input_layout 为 BNSD_BSND 时，表示输入格式为 BNSD，输出格式为 BSND，仅支持 Q_S 大于 1。默认值为 BSH。
- **num_key_value_heads** (int, 可选) - key/value 的 head 个数，用于支持 GQA (Grouped-Query Attention, 分组查询注意力) 场景。默认值为 0。0 意味着和 key/value 的 head 个数相等。num_heads 必须要能够整除 num_key_value_heads。num_heads 和 num_key_value_heads 的比值不能大于 64。当 input_layout 为 BNSD 时，该参数需和 key/value 的 N 轴 shape 值相同，否则执行异常。
- **sparse_mode** (int, 可选) - 指定稀疏模式，默认值为 0。Q_S 为 1 时该参数无效。
 - 0: 代表 defaultMask 模式。如果 atten_mask 未传入则不做 mask 操作，忽略 pre_tokens 和 next_tokens (内部赋值为 INT_MAX)；如果传入，则需要传入完整的 atten_mask 矩阵 ($S1 * S2$)，表示 pre_tokens 和 next_tokens 之间的部分需要计算。
 - 1: 代表 allMask，必须传入完整的 atten_mask 矩阵 ($S1 * S2$)。
 - 2: 代表 leftUpCausal 模式的 mask，需传入优化后的 atten_mask 矩阵 ($2048*2048$)。
 - 3: 代表 rightDownCausal 模式的 mask，对应以右顶点为划分的下三角场景，需传入优化后的 atten_mask 矩阵 ($2048*2048$)。
 - 4: 代表 band 模式的 mask，即计算 pre_tokens 和 next_tokens 之间的部分。需传入优化后的 atten_mask 矩阵 ($2048*2048$)。
 - 5: 代表 prefix 场景，暂不支持。
 - 6: 代表 global 场景，暂不支持。
 - 7: 代表 dilated 场景，暂不支持。
 - 8: 代表 block_local 场景，暂不支持。
- **inner_precise** (int, 可选) - 共 4 种模式：0、1、2、3，通过 2 个 bit 位表示：第 0 位 (bit0) 表示高精度或者高性能选择，第 1 位 (bit1) 表示是否做行无效修正。
 - 0: 代表开启高精度模式，且不做行无效修正。
 - 1: 代表高性能模式，且不做行无效修正。
 - 2: 代表开启高精度模式，且做行无效修正。
 - 3: 代表高性能模式，且做行无效修正。

当 $Q\_S > 1$ 时，如果 sparse_mode 为 0 或 1，并传入用户自定义 mask，则建议开启行无效；当 Q_S 为 1 时，该参数只能为 0 和 1。默认值为 1。

高精度和高性能只对 float16 生效，行无效修正对 float16、bfloat16 和 int8 均生效。当前 0、1 为保留配置值，当计算过程中“参与计算的 mask 部分”存在某整行全为 1 的情况时，精度可能会有损失。此时可以尝试将

该参数配置为 2 或 3 来使能行无效功能以提升精度，但是该配置会导致性能下降。如果函数可判断出存在无效行场景，会自动使能无效行计算，例如 `sparse_mode` 为 3，`Sq > Skv` 场景。

- **block_size** (int, 可选) - PageAttention 中 KV 存储每个 block 中最大的 token 个数。默认值为 0。Q_S 大于 1 时该参数无效。
- **antiquant_mode** (int, 可选) - 伪量化的方式，传入 0 时表示为 per-channel（per-channel 包含 per-tensor），传入 1 时表示为 per-token。per-channel 和 per-tensor 可以通过入参 shape 的维数区分，如果 shape 的维数为 1，则运行在 per-tensor 模式；否则运行在 per-channel 模式。默认值为 0。Q_S 大于 1 时该参数无效。
- **key_antiquant_mode** (int, 可选) - key 的伪量化的方式，传入 0 时表示为 per-channel（per-channel 包含 per-tensor），传入 1 时表示为 per-token。默认值为 0。Q_S 大于 1 时该参数无效。
- **value_antiquant_mode** (int, 可选) - value 的伪量化的方式，传入 0 时表示为 per-channel（per-channel 包含 per-tensor），传入 1 时表示为 per-token。默认值为 0。Q_S 大于 1 时该参数无效。
- **softmax_lse_flag** (bool, 可选) - 是否输出 softmax_lse。默认值为 False。

返回：

`attention_out` (Tensor)，注意力分值，数据类型为 float16、bfloat16 或 int8。当 `input_layout` 为 BNSD_BSND 时，输出的 shape 为 (B, S, N, D) 。其余情况输出的 shape 和 query 的 shape 一致。

`softmax_lse` (Tensor)，`softmax_lse` 值，数据类型为 float32，通过将 query 乘 key 的结果经过 lse (log、sum 和 exp) 计算后获得。具体地，ring attention 算法对 query 乘 key 的结果先取 max，得到 `softmax_max`；query 乘 key 的结果减去 `softmax_max`，再取 exp，最后取 sum，得到 `softmax_sum`；最后对 `softmax_sum` 取 log，再加上 `softmax_max`，得到 `softmax_lse`。`softmax_lse` 只在 `softmax_lse_flag` 为 True 时计算，输出的 shape 为 $(B, N, Q_S, 1)$ 。如果 `softmax_lse_flag` 为 False，则将返回一个 shape 为 (1) 的全 0 的 Tensor。在图模式且 JitConfig 为 O2 场景下请确保使能 `softmax_lse_flag` 后再使用 `softmax_lse`，否则将遇到异常。

约束：

- 全量推理场景 ($Q_S > 1$)：
 - query, key, value 输入，功能使用限制如下：
 - * 支持 B 轴小于等于 65535，输入类型包含 int8 时 D 轴非 32 对齐或输入类型为 float16 或 bfloat16 时 D 轴非 16 对齐时，B 轴仅支持到 128。
 - * 支持 N 轴小于等于 256，支持 D 轴小于等于 512。
 - * S 支持小于等于 20971520 (20M)。部分长序列场景下，如果计算量过大可能会导致 pfa 算子执行超时 (AICore error 类型报错，errorStr 为：timeout or trap error)，此场景下建议做 S 切分处理，注：这里计算量会受 B、S、N、D 等的影响，值越大计算量越大。典型的会超时的长序列（即 B、S、N、D 的乘积较大）场景包括但不限于：
 1. B=1, Q_N=20, Q_S=2097152, D=256, KV_N=1, KV_S=2097152;
 2. B=1, Q_N=2, Q_S=20971520, D=256, KV_N=2, KV_S=20971520;
 3. B=20, Q_N=1, Q_S=2097152, D=256, KV_N=1, KV_S=2097152;
 4. B=1, Q_N=10, Q_S=2097152, D=512, KV_N=1, KV_S=2097152。
 - * query、key、value 或 `attention_out` 类型包含 int8 时，D 轴需要 32 对齐；类型全为 float16、bfloat16 时，D 轴需 16 对齐。
 - 参数 `sparse_mode` 当前仅支持值为 0、1、2、3、4 的场景，取其它值时会报错：
 - * `sparse_mode = 0` 时，`atten_mask` 如果为 None，或者在左 padding 场景传入 `atten_mask`，则忽略入参 `pre_tokens`、`next_tokens`。
 - * `sparse_mode = 2、3、4` 时，`atten_mask` 的 shape 需要为 S,S 或 1,S,S 或 1,1,S,S，其中 S 的值需要固定为 2048，且需要用户保证传入的 `atten_mask` 为下三角矩阵，不传入 `atten_mask` 或者传入的 shape 不正确报错。

- * `sparse_mode = 1、2、3` 的场景忽略入参 `pre_tokens、next_tokens` 并按照相关规则赋值。
- kvCache 反量化仅支持 query 为 float16 时, 将 int8 类型的 key 和 value 反量化到 float16。入参 key/value 的 `datarange` 与入参 `antiquant_scale` 的 `datarange` 乘积范围在 $(-1, 1)$ 范围内, 高性能模式可以保证精度, 否则需要开启高精度模式来保证精度。
- query 左 padding 场景:
 - * query 左 padding 场景 query 的搬运起点计算公式为: $Q\_S - query\_padding\_size - actual\_seq\_lengths$ 。query 的搬运终点计算公式为: $Q\_S - query\_padding\_size$ 。其中 query 的搬运起点不能小于 0, 终点不能大于 Q_S , 否则结果将不符合预期。
 - * query 左 padding 场景 kv_padding_size 小于 0 时将被置为 0。
 - * query 左 padding 场景需要与 `actual_seq_lengths` 参数一起使能, 否则默认为 query 右 padding 场景。
 - * query 左 padding 场景不支持 PageAttention, 不能与 `block_table` 参数一起使能。
- kv 左 padding 场景:
 - * kv 左 padding 场景 key 和 value 的搬运起点计算公式为: $KV\_S - kv\_padding\_size - actual\_seq\_lengths\_kv$ 。key 和 value 的搬运终点计算公式为: $KV\_S - kv\_padding\_size$ 。其中 key 和 value 的搬运起点不能小于 0, 终点不能大于 KV_S , 否则结果将不符合预期。
 - * kv 左 padding 场景 kv_padding_size 小于 0 时将被置为 0。
 - * kv 左 padding 场景需要与 `actual_seq_lengths_kv` 参数一起使能, 否则默认为 kv 右 padding 场景。
 - * kv 左 padding 场景不支持 PageAttention, 不能与 `block_table` 参数一起使能。
- pse_shift 功能使用限制如下:
 - * 支持 query 数据类型为 float16 或 bfloat16 或 int8 场景下使用该功能。
 - * query 数据类型为 float16 且 pse_shift 存在时, 强制走高精度模式, 对应的限制继承自高精度模式的限制。
 - * Q_S 需大于等于 query 的 S 长度, KV_S 需大于等于 key 的 S 长度。
- kv 伪量化参数分离当前暂不支持。
- 增量推理场景 ($Q\_S = 1$):
 - query, key, value 输入, 功能使用限制如下:
 - * 支持 B 轴小于等于 65536, 支持 N 轴小于等于 256, 支持 D 轴小于等于 512。
 - * query、key、value 输入类型均为 int8 的场景暂不支持。
 - page attention 场景:
 - * page attention 的使能必要条件是 `block_table` 存在且有效, 同时 key、value 是按照 `block_table` 中的索引在一片连续内存中排布, 支持 key、value dtype 为 float16/bfloat16/int8, 在该场景下 key、value 的 `input_layout` 参数无效。
 - * `block_size` 是用户自定义的参数, 该参数的取值会影响 page attention 的性能, 在使能 page attention 场景下, `block_size` 需要传入非 0 值, 且 `block_size` 最大不超过 512。key、value 输入类型为 float16/bfloat16 时需要 16 对齐, key、value 输入类型为 int8 时需要 32 对齐, 推荐使用 128。通常情况下, page attention 可以提高吞吐量, 但会带来性能上的下降。
 - * page attention 使能场景下, 当输入 kv cache 排布格式为 (blocknum, block_size, H), 且 `num_key_value_heads` * D 超过 64k 时, 受硬件指令约束, 会被拦截报错。可通过使能 GQA (减小 `num_key_value_heads`) 或调整 kv cache 排布格式为 (blocknum, num_key_value_heads, block_size, D) 解决。
 - * page attention 场景的参数 key、value 各自对应 tensor 的 shape 所有维度相乘不能超过 int32 的表示范围。

- page attention 的使能场景下，以下场景输入 S 需要大于等于 $\text{max\_block\_num\_per\_seq} * \text{block\_size}$:
 - * 使能 Attention mask，如 mask shape 为 (B, 1, 1, S)。
 - * 使能 pse_shift，如 pse_shift shape 为 (B, N, 1, S)。
 - * 使能伪量化 per-token 模式：输入参数 `antiquant_scale` 和 `antiquant_offset` 的 shape 均为 (2, B, S)。
- kv 左 padding 场景：
 - * kv 左 padding 场景 kvCache 的搬运起点计算公式为： $\text{KV\_S} - \text{kv\_padding\_size} - \text{actual\_seq\_lengths}$ 。kv-Cache 的搬运终点计算公式为： $\text{KV\_S} - \text{kv\_padding\_size}$ 。其中 kvCache 的搬运起点或终点小于 0 时，返回数据结果为全 0。
 - * kv 左 padding 场景 kv_padding_size 小于 0 时将被置为 0。
 - * kv 左 padding 场景需要与 `actual_seq_lengths` 参数一起使能，否则默认为 kv 右 padding 场景。
 - * kv 左 padding 场景需要与 `atten_mask` 参数一起使能，且需要保证 `atten_mask` 含义正确，即能够正确的对无效数据进行隐藏。否则将引入精度问题。
- pse_shift 功能使用限制如下：
 - * `pse_shift` 数据类型需与 `query` 数据类型保持一致。
 - * 仅支持 D 轴对齐，即 D 轴可以被 16 整除。
- kv 伪量化参数分离：
 - * `key_antiquant_mode` 和 `value_antiquant_mode` 需要保持一致。
 - * `key_antiquant_scale` 和 `value_antiquant_scale` 要么都为空，要么都不为空。
 - * `key_antiquant_offset` 和 `value_antiquant_offset` 要么都为空，要么都不为空。
 - * `key_antiquant_scale` 和 `value_antiquant_scale` 都不为空时，其 shape 需要保持一致。
 - * `key_antiquant_offset` 和 `value_antiquant_offset` 都不为空时，其 shape 需要保持一致。

支持平台：

Ascend

样例：

```
>>> from mindspore import ops
>>> from mindspore import Tensor
>>> import numpy as np
>>> B, N, S, D = 1, 8, 1024, 128
>>> query = Tensor(np.random.rand(B, N, S, D).astype(np.float16))
>>> key = Tensor(np.random.rand(B, N, S, D).astype(np.float16))
>>> value = Tensor(np.random.rand(B, N, S, D).astype(np.float16))
>>> out = ops.fused_infer_attention_score(query, key, value, num_heads=N, input_layout='BNSD
↪')
>>> print(out[0].shape)
(1, 8, 1024, 128)
```

mindspore.ops.speed_fusion_attention

```
mindspore.ops.speed_fusion_attention(query, key, value, head_num, input_layout, *, pse=None,
                                     padding_mask=None, atten_mask=None, scale=1.0, keep_prob=1.0,
                                     pre_tokens=2147483647, next_tokens=2147483647, inner_precise=0,
                                     prefix=None, actual_seq_qlen=None, actual_seq_kvlen=None,
                                     sparse_mode=0, gen_mask_parallel=True, sync=False, pse_type=1,
                                     q_start_idx=None, kv_start_idx=None)
```

本接口用于实现 self-attention 的融合计算。当 *pse_type* 为 1 时，计算公式为：

$$attention\_out = Dropout(Softmax(Mask(scale * (pse + query * key^T), atten\_mask)), keep\_prob) * value$$

当 *pse_type* 为其他有效值时，计算公式为：

$$attention\_out = Dropout(Softmax(Mask(scale * (query * key^T) + pse, atten\_mask)), keep\_prob) * value$$

- B: batch 维，取值范围 1 至 2k。
- S1: *query* 的序列长度，取值范围 1 至 512k。
- S2: *key* 和 *value* 的序列长度，取值范围 1 至 512k。
- N1: *query* 的多头数量，取值范围 1 至 256。
- N2: *key* 和 *value* 的多头数量，且 N2 必须为 N1 的因子。
- D: 单头大小，取值范围为 16 的倍数，最大值为 512。
- H1: *query* 的隐藏层大小，等于 N1*D。
- H2: *key* 和 *value* 的隐藏层大小，等于 N2*D。

警告：

- 这是一个实验性 API，可能会发生变更或被删除。
- 只支持 Atlas A2 训练系列产品。

说明：不支持在 图模式 (*mode=mindspore.GRAPH_MODE*) 下执行。

参数：

- **query** (Tensor) - query Tensor, shape 支持: $(B, S1, H1)$ 、 $(B, N1, S1, D)$ 、 $(S1, B, H1)$ 、 $(B, S1, N1, D)$ 或 $(T1, N1, D)$ 。
- **key** (Tensor) - key Tensor, shape 支持: $(B, S2, H2)$ 、 $(B, N2, S2, D)$ 、 $(S2, B, H2)$ 、 $(B, S2, N2, D)$ 或 $(T2, N2, D)$ 。
- **value** (Tensor) - value Tensor, shape 支持: $(B, S2, H2)$ 、 $(B, N2, S2, D)$ 、 $(S2, B, H2)$ 、 $(B, S2, N2, D)$ 或 $(T2, N2, D)$ 。
。 *value* 的 shape 应与 *key* 一致。
- **head_num** (int) - *query* 的多头数量，等于 N1。
- **input_layout** (str) - 指定输入 *query*、*key* 和 *value* 的数据格式。值可以为 "BSH"、"BNSD"、"SBH"、"BSND" 或 "TND"。其中 "TND" 是实验性格式。

当 *input_layout* 为 "TND" 时，必须满足以下限制：假设有两个列表 *list_seq_q* 和 *list_seq_k* 表示输入序列的长度。每个列表的值表示 batch 中每个序列的长度。例如，*list_seq_q* = [4, 2, 6]，*list_seq_k* = [10, 3, 9]。T1 =

$\text{sum}(\text{list\_seq\_q}) = 12$, $T2 = \text{sum}(\text{list\_seq\_k}) = 22$ 。 $\text{max\_seqlen\_q} = \max(\text{list\_seq\_q})$, $\text{max\_seqlen\_k} = \max(\text{list\_seq\_k})$ 。
 $\text{qk\_pointer} = \text{sum}(\text{list\_seq\_q} * \text{list\_seq\_k})$, 表示元素乘积的总和。

- 两个列表的长度需要相同, 列表大小等于 batch 数量。batch 数量最大为 1024。
- 当 `input_layout` 为 "TND" 时, `actual_seq_qlen` 和 `actual_seq_kvlen` 必须不为 None。否则, 它们为 None。
- `actual_seq_qlen` 和 `actual_seq_kvlen` 是 `key/value` 序列的累计和, 因此必须是非递减的。
- 如果 `pse` 不为 None, 则 `list_seq_q` 和 `list_seq_k` 必须相同。`list_seq_q` 和 `list_seq_k` 的最大值必须大于 1024。`pse` 应为 $(B, N1, 1024, S2)$ 或 $(1, N1, 1024, S2)$, 其中 $S2$ 等于 `max_seqlen_k`。
- `atten_mask` 必须是下三角矩阵, 因此 `sparse_mode` 应为 2 或 3, 且 `atten_mask` 的 shape 应为 $(2048, 2048)$ 。
- `prefix` 为 None。
- `next_tokens` 为 0, `pre_tokens` 不小于 `max_seqlen_q`。
- 当 `sparse_mode` 为 3 时, 每个 batch 的 $S1$ 应小于或等于 $S2$ 。
- `list_seq_k` 中不应存在 0。

关键字参数:

- **pse** (Tensor, 可选) - 位置编码 Tensor, 数据类型与 `query` 一致。默认 None。如果 S 大于 1024 并且使用了下三角的掩码, 则仅使用下三角的逆向 1024 行以进行内存优化。支持的 shape 为: $(B, N1, S1, S2)$ 、 $(1, N1, S1, S2)$ 、 $(B, N1, 1024, S2)$ 或 $(1, N1, 1024, S2)$ 。
 - ALiBi 场景: `pse` 必须满足 ALiBi 规则, 并且在下三角场景下 `sparse_mode` 为 2 或 3。在此场景中, `pse` 的 shape 为 $(B, N1, 1024, S2)$ 或 $(1, N1, 1024, S2)$ 。
 - 非 ALiBi 场景: `pse` 的 shape 为 $(B, N1, S1, S2)$ 或 $(1, N1, S1, S2)$ 。
 - `input_layout` 为 "TND" 时: shape 应为 $(B, N1, 1024, S2)$ 或 $(1, N1, 1024, S2)$ 。
 - 如果 `pse_type` 为 2 或 3 时, 数据类型需为 float32, 支持的 shape 为: $(B, N1)$ 或 $(N1,)$ 。
- **padding_mask** (Tensor, 可选) - 保留参数, 尚未使能。默认值为: None。
- **atten_mask** (Tensor, 可选) - 注意力掩码 Tensor, 数据类型为 bool、uint8。对于每个元素, 0/False 表示保留, 1/True 表示丢弃。shape 可以是 $(B, N1, S1, S2)$ 、 $(B, 1, S1, S2)$ 、 $(S1, S2)$ 或 $(2048, 2048)$ 。默认值为: None。
 - 在压缩场景中, 当 `sparse_mode` 为 2、3 或 4 时, `atten_mask` 必须为 $(2048, 2048)$ 。
 - 当 `sparse_mode` 为 5 时, `atten_mask` 应该为 $(B, N1, S1, S2)$ 或 $(B, 1, S1, S2)$ 。
 - 当 `sparse_mode` 为 0 和 1 时, `atten_mask` 应该是 $(B, N1, S1, S2)$ 、 $(B, 1, S1, S2)$ 或 $(S1, S2)$ 。
- **scale** (float, 可选) - 缩放因子。通常, 该值为 $1.0 / (D ** 0.5)$ 。默认值为: 1.0。
- **keep_prob** (float, 可选) - Dropout 的保留概率, 取值范围为 $(0.0, 1.0]$ 。默认值为: 1.0。
- **pre_tokens** (int, 可选) - 用于稀疏计算, 表示向前计算的 token 数量。当 `sparse_mode` 设置为 1、2、3 或 5 时, 此参数无效。默认值为: 2147483647。
- **next_tokens** (int, 可选) - 稀疏计算的参数, 表示向后计算的 token 数量。当 `sparse_mode` 设置为 1、2、3 或 5 时, 此参数无效。默认值为: 2147483647。`pre_tokens` 的值对应于 $S1$, 而 `next_tokens` 的值对应于 $S2$ 。它们定义了 `atten_mask` 矩阵上的有效区域, 并确保带宽不为空。以下情况是非法的:
 - `pre_tokens < 0` 且 `next_tokens < 0`。
 - `(pre_tokens < 0 且 next_tokens >= 0)` 且 `(next_tokens < abs(pre_tokens) 或 abs(pre_tokens) >= S2)`。
 - `(pre_tokens >= 0 且 next_tokens < 0)` 且 `(abs(next_tokens) > pre_tokens 或 abs(next_tokens) >= S1)`。
- **inner_precision** (int, 可选) - 保留参数, 尚未使能。默认值为: 0。

- **prefix** (Union[tuple[int], list[int]], 可选) - 在前缀稀疏计算场景下, 每个 batch 的 N 值。输入为 Tensor 时的 shape 为 (B,), 其中 B 的最大值为 32。仅当 *sparse_mode* 为 5 时, 此参数不为 None。默认值为: None。如果 $S1 > S2$, N 的范围为 0 到 $S2$ 。如果 $S1 \leq S2$, N 的范围为 $S2 - S1$ 到 $S2$ 。
- **actual_seq_qlen** (Union[tuple[int], list[int]], 可选) - 每个 batch 中 query 序列的大小, 用一个递增值数组表示, 最后一个值等于 T1。默认值为: None。
- **actual_seq_kvlen** (Union[tuple[int], list[int]], 可选) - 每个 batch 中 key 和 value 序列的大小, 用一个递增值数组表示, 最后一个值等于 T2。默认值为: None。
- **sparse_mode** (int, 可选) - 指定稀疏模式。默认值为: 0。
 - 0: 表示 defaultMask 模式。如果未传入 *atten_mask*, 则不执行掩码操作, 并忽略 *pre_tokens* 和 *next_tokens* (内部分配为 INT_MAX)。如果传入, 需传递完整的 *atten_mask* 矩阵 ($S1 * S2$), 表示需计算的部分在 *pre_tokens* 和 *next_tokens* 之间。
 - 1: 表示 allMask 模式, 即传入完整的 *atten_mask* 矩阵。
 - 2: 表示 leftUpCausal 模式, 适用于以左顶点划分的下三角场景, 需优化的 *atten_mask* 矩阵为 (2048 * 2048)。
 - 3: 表示 rightDownCausal 模式, 适用于以右下顶点划分的下三角场景, 需优化的 *atten_mask* 矩阵为 (2048 * 2048)。
 - 4: 表示带状场景, 即计算 *pre_tokens* 和 *next_tokens* 之间的部分, 需优化的 *atten_mask* 矩阵为 (2048 * 2048)。
 - 5: 表示前缀场景, 即在 rightDownCausal 的基础上, 矩阵左侧添加长度为 $S1$ 、宽度为 N 的矩阵。N 的值由新输入 *prefix* 获得, 每个 batch 轴的 N 值不同, 目前尚未使能。
 - 6: 表示全局场景, 尚未使能。
 - 7: 表示膨胀场景, 尚未使能。
 - 8: 表示局部块场景, 尚未使能。
- **gen_mask_parallel** (bool, 可选) - Debug 参数, 控制 dropout_gen_mask 执行方式的开关, 如果为 True, dropout_gen_mask 并行执行。如果为 False, dropout_gen_mask 串行执行。尚未使能。默认值为: True。
- **sync** (bool, 可选) - Debug 参数, 控制 dropout_gen_mask 执行方式的开关, 如果为 True, dropout_gen_mask 同步执行。如果为 False, dropout_gen_mask 异步执行。尚未使能。默认值为: False。
- **pse_type** (int, 可选) - 指定 pse 的使用方式。默认值为: 1。
 - 0: pse 由外部传入, 计算流程是先 mul scale 再 add pse。
 - 1: pse 由外部传入, 计算流程是先 add pse 再 mul scale。
 - 2: pse 由内部生成, 生成标准 alibi 位置信息。内部生成的 alibi 矩阵 0 线与 $query * key^T$ 的左上角对齐。
 - 3: pse 由内部生成, 生成的 alibi 位置信息为标准的基础上再做 sqrt 开平方。内部生成的 alibi 矩阵 0 线与 $query * key^T$ 的左上角对齐。
- **q_start_idx** (Union[tuple[int], list[int]], 可选) - 长度为 1 的序列, 默认值为: None。pse_type 配置为 2 或 3 时, 表示内部生成 alibi 编码在 $S1$ 方向偏移的格数, 正数表示 0 对角线向上移动。
- **kv_start_idx** (Union[tuple[int], list[int]], 可选) - 长度为 1 的序列, 默认值为: None。pse_type 配置为 2 或 3 时, 表示内部生成 alibi 编码在 $S2$ 方向偏移的格数, 正数表示 0 对角线向上移动。

返回:

一个 Tensor 元组, 包含 *attention_out*、*softmax_max*、*softmax_sum*、*softmax_out*、*seed*、*offset* 和 *numels*。

- *attention_out* 是输出 Tensor, 其 shape 和 dtype 与 *query* 相同。
- *softmax_max* 是 Softmax 计算的 Max 中间结果, 用于反向计算。

- *softmax_sum* 是 Softmax 计算的 Sum 中间结果，用于反向计算。
- *softmax_out* 是保留参数。
- *seed* 是生成的 seed，用于 Dropout。
- *offset* 是生成的 offset，用于 Dropout。
- *numels* 是生成的 dropout_mask 的长度。

异常：

- **TypeError** - *query*、*key* 和 *value* 的数据类型不同。
- **TypeError** - *atten_mask* 的数据类型不是 bool 或 uint8。
- **TypeError** - *scale* 或 *keep_prob* 不是 float 类型。
- **TypeError** - *input_layout* 不是字符串。
- **TypeError** - *head_num* 不是整数。
- **TypeError** - *sparse_mode* 不是整数。
- **TypeError** - *pse* 不是 Tensor。
- **TypeError** - *padding_mask* 不是 Tensor。
- **TypeError** - *atten_mask* 不是 Tensor。
- **TypeError** - *pse_type* 不是整数。
- **ValueError** - *input_layout* 是字符串但不是合法值。
- **ValueError** - *sparse_mode* 被指定的值不合法。
- **ValueError** - *pse_type* 被指定的值不合法。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import mindspore.common.dtype as mstype
>>> import numpy as np
>>> from mindspore import ops, Tensor
>>> query = Tensor(np.ones([2, 4, 64]), dtype=mstype.float16)
>>> key = Tensor(np.ones([2, 4, 64]), dtype=mstype.float16)
>>> value = Tensor(np.ones([2, 4, 64]), dtype=mstype.float16)
>>> head_num = 4
>>> input_layout = "BSH"
>>> output = ops.speed_fusion_attention(query, key, value, head_num, input_layout)
>>> print(output[0].shape)
(2, 4, 64)
```

mindspore.ops.group_norm

`mindspore.ops.group_norm(input, num_groups, weight=None, bias=None, eps=1e-5)`

在 mini-batch 输入上进行组归一化。

Group Normalization 被广泛用于递归神经网络中。适用于单个训练用例的 mini-batch 输入归一化，详见论文 [Group Normalization](#)。

Group Normalization 把通道划分为组，然后计算每一组之内的均值和方差，以进行归一化。其中 γ 是通过训练学习出的 scale 值， β 是通过训练学习出的 shift 值。

公式如下，

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中， γ 为 *weight*， β 为 *bias*， ϵ 为 *eps*。

参数：

- **input** (Tensor) - shape 为 $(N, C, *)$ 的特征输入，其中 $*$ 表示任意的附加维度。
- **num_groups** (int) - 沿通道维度待划分的组数。
- **weight** (Tensor, 可选) - shape 为 $(C,)$ ，默认值为：None，具有与 *input* 相同的数据类型。
- **bias** (Tensor, 可选) - shape 为 $(C,)$ ，默认值为：None，具有与 *input* 相同的数据类型。
- **eps** (float, 可选) - 添加到分母中的值，以确保数值稳定。默认 $1e-5$ 。

返回：

Tensor，标准化和缩放的偏移 Tensor，具有与 *input* 相同的 shape 和数据类型。

异常：

- **TypeError** - *num_groups* 不是 int。
- **TypeError** - *eps* 不是 float。
- **ValueError** - *num_groups* 小于 1。
- **ValueError** - C (*input* 的第二维) 未被 *num_groups* 整除。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import ops
>>> x = ms.Tensor(np.ones([1, 2, 4, 4], np.float32))
>>> output = ops.group_norm(x, 2)
>>> print(output)
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]
 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]]]
```

mindspore.ops.layer_norm

`mindspore.ops.layer_norm(input, normalized_shape, weight=None, bias=None, eps=1e-5)`

在 mini-batch 输入上应用层归一化 (Layer Normalization)。

层归一化在递归神经网络中被广泛的应用。适用单个训练用例的 mini-batch 输入上应用归一化, 详见论文 [Layer Normalization](#)。

与批归一化 (Batch Normalization) 不同, 层归一化在训练和测试时执行完全相同的计算。应用于所有通道和像素, 即使 `batch_size=1` 也适用。公式如下:

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中 γ 是通过训练学习出的 `weight` 值, β 是通过训练学习出的 `bias` 值。

参数:

- **input** (Tensor) - `input` 的 shape 为 $(N, *)$, 其中 $*$ 表示任意的附加维度。
- **normalized_shape** (Union(int, tuple[int], list[int])) - 表示需要进行归一化的 shape, `normalized_shape` 等于 `input_shape[begin_norm_axis:]`, `begin_norm_axis` 代表归一化要开始的轴。
- **weight** (Tensor, 可选) - 可学习的权重值, shape 为 `normalized_shape`, 默认值: `None`。为 `None` 时, 初始化为 1。
- **bias** (Tensor, 可选) - 可学习的偏移值, shape 为 `normalized_shape`, 默认值: `None`。为 `None` 时, 初始化为 0。
- **eps** (float, 可选) - 添加到分母中的值 (ϵ), 以确保数值稳定。默认 `1e-5`。

返回:

Tensor, 归一化后的 Tensor, shape 和数据类型与 `input` 相同。

异常:

- **TypeError** - `input` 不是 Tensor。
- **TypeError** - `normalized_shape` 既不是 list 也不是 tuple。
- **TypeError** - `eps` 不是 float。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[1, 2, 3], [1, 2, 3]]), mindspore.float32)
>>> normalized_shape = (3,)
>>> gamma = Tensor(np.ones(normalized_shape), mindspore.float32)
>>> beta = Tensor(np.zeros(normalized_shape), mindspore.float32)
>>> eps = 1e-7
>>> output = ops.layer_norm(input_x, normalized_shape, gamma, beta, eps)
>>> print(output)
[[-1.2247448 0. 1.2247448]
 [-1.2247448 0. 1.2247448]]
```


mindspore.ops.lp_pool1d

`mindspore.ops.lp_pool1d(x, norm_type, kernel_size, stride=None, ceil_mode=False)`

在输入 Tensor 上应用 1D LP 池化运算，可被视为组成一个 1D 输入平面。

通常，输入的 shape 为 (N, C, L_{in}) 或 (C, L_{in}) ，输出的 shape 为 (N, C, L_{out}) 或 (C, L_{out}) 。

$$L_{out} = \left\lfloor \frac{L_{in} - \text{kernel\_size}}{\text{stride}} + 1 \right\rfloor$$

公式如下：

$$f(X) = \sqrt[p]{\sum_{x \in X} x^p}$$

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **x** (Tensor) - shape 为 (N, C, L_{in}) 或 (C, L_{in}) 的 Tensor。
- **norm_type** (Union[int, float]) - 标准化类型，代表公式里的 p ，不能为 0。
 - 如果 $p = 1$ ，得到的结果为池化核内元素之和（与平均池化成比例）。
 - 如果 $p = \infty$ ，得到的结果为最大池化的结果。
- **kernel_size** (int) - 池化核的尺寸大小。
- **stride** (int) - 池化操作的移动步长，数据类型为整型。默认 None，表示移动步长为 `kernel_size`。
- **ceil_mode** (bool) - 若为 True，使用 ceil 来计算输出 shape。若为 False，使用 floor 来计算输出 shape。默认 False。

返回：

- **output** - LPPool1d 的计算结果，shape 为 (N, C, L_{out}) 或 (C, L_{out}) 的 Tensor，与输入 x 的类型一致，其中：

$$L_{out} = \left\lfloor \frac{L_{in} - \text{kernel\_size}}{\text{stride}} + 1 \right\rfloor$$

异常：

- **TypeError** - x 不是 Tensor。
- **TypeError** - `kernel_size` 或 `stride` 不是 int。
- **TypeError** - `ceil_mode` 不是 bool。
- **TypeError** - `norm_type` 不是 float 也不是 int。
- **ValueError** - `norm_type` 等于 0。
- **ValueError** - `kernel_size` 或 `stride` 小于 1。
- **ValueError** - x 的 shape 长度不等于 2 或 3。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import ops
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.arange(2 * 3 * 4).reshape((2, 3, 4)), dtype=ms.float32)
>>> out = ops.lp_pool1d(x, norm_type=1, kernel_size=3, stride=1, ceil_mode=False)
>>> print(out)
[[[ 3.  6.]
  [15. 18.]
  [27. 30.]]
 [[39. 42.]
  [51. 54.]
  [63. 66.]]]
```

mindspore.ops.lp_pool2d

`mindspore.ops.lp_pool2d(x, norm_type, kernel_size, stride=None, ceil_mode=False)`

在输入 Tensor 上应用 2D LP 池化运算，可被视为组成一个 2D 输入平面。

通常，输入的 shape 为 (N, C, H_{in}, W_{in}) ，输出的 shape 为 (N, C, H_{in}, W_{in}) ，输出与输入的 shape 一致，公式如下：

$$f(X) = \sqrt[p]{\sum_{x \in X} x^p}$$

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **x** (Tensor) - shape 为 (N, C, H_{in}, W_{in}) 的 Tensor。
- **norm_type** (Union[int, float]) - 标准化类型，代表公式里的 p ，不能为 0，
 - 如果 $p = 1$ ，得到的结果为池化核内元素之和（与平均池化成比例）；
 - 如果 $p = \infty$ ，得到的结果为最大池化的结果。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。如果为整数，则代表池化核的高和宽。如果为 tuple，其值必须包含两个整数值分别表示池化核的高和宽。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，如果为整数，则代表 stride 的高和宽。如果为 tuple，其值必须包含两个整数值分别表示 stride 的高和宽。默认 None，表示移动步长为 `kernel_size`。
- **ceil_mode** (bool) - 若为 True，使用 ceil 模式来计算输出 shape。若为 False，使用 floor 模式来计算输出 shape。默认 False。

返回：

- **output** - LPPool2d 的计算结果，shape 为 (N, C, H_{in}, W_{in}) 的 Tensor，与输入 x 的类型一致，其中

$$H_{out} = \left\lfloor \frac{H_{in} - \text{kernel\_size}[0]}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} - \text{kernel\_size}[1]}{\text{stride}[1]} + 1 \right\rfloor$$

异常：

- **TypeError** - x 不是 Tensor。

- **TypeError** - *kernel_size* 或 *stride* 不是 int 也不是 tuple。
- **TypeError** - *ceil_mode* 不是 bool。
- **TypeError** - *norm_type* 不是 float 也不是 int。
- **ValueError** - *norm_type* 等于 0。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *kernel_size* 或 *stride* 是一个长度不为 2 的 tuple。
- **ValueError** - *x* 的 shape 长度不等于 4。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import ops
>>> from mindspore import Tensor
>>> import numpy as np
>>> x = Tensor(np.arange(2 * 3 * 4 * 5).reshape((2, 3, 4, 5)), dtype=ms.float32)
>>> out = ops.lp_pool2d(x, norm_type=1, kernel_size=3, stride=1, ceil_mode=False)
>>> print(out)
[[[ [ 54.   63.   72.]
      [ 99.  108.  117.]]
  [ [ 234.  243.  252.]
      [ 279.  288.  297.]]
  [ [ 414.  423.  432.]
      [ 459.  468.  477.]]]]
[[ [ 594.  603.  612.]
      [ 639.  648.  657.]]
  [ [ 774.  783.  792.]
      [ 819.  828.  837.]]
  [ [ 954.  963.  972.]
      [ 999. 1008. 1017.]]]]
```

mindspore.ops.lrn

`mindspore.ops.lrn(x, depth_radius=5, bias=1.0, alpha=1.0, beta=0.5, norm_region='ACROSS_CHANNELS')`
局部响应归一化操作 LRN(Local Response Normalization)。

警告： `lrn` 在 Ascend 平台已废弃，存在潜在精度问题。建议使用其他归一化方法，如 `mindspore.ops.batch_norm` 代替 LRN。

$$b_c = a_c \left(k + \frac{\alpha}{n} \sum_{c'=\max(0, c-n/2)}^{\min(N-1, c+n/2)} a_{c'}^2 \right)^{-\beta}$$

其中 a_c 表示特征图中 c 对应的具体像素值； $n/2$ 为参数 *depth_radius*； k 为参数 *bias*； α 为参数 *alpha*； β 为参数 *beta*。

参数：

- **x** (Tensor) - 数据类型为 float16 或 float32 的四维 Tensor。
- **depth_radius** (int) - 一维归一化窗口的半宽。默认 5。
- **bias** (float) - 偏移量（通常为正以避免除零问题）。默认 1.0。
- **alpha** (float) - 比例系数，通常为正。默认 1.0。
- **beta** (float) - 指数。默认 0.5。
- **norm_region** (str) - 指定归一化区域。可选值: "ACROSS_CHANNELS"。默认 "ACROSS_CHANNELS"。

返回：

Tensor，与 x 的 shape 和数据类型相同。

异常：

- **TypeError** - *depth_radius* 不是 int 类型。
- **TypeError** - *bias*、*alpha* 或 *beta* 不是 float 类型。
- **TypeError** - *norm_region* 不是 str。
- **TypeError** - x 不是 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[[[0.1], [0.2]],
...                             [[0.3], [0.4]]]]), mindspore.float32)
>>> output = ops.lrn(input_x)
>>> print(output)
[[[0.09534626]
  [0.1825742 ]]
 [0.2860388 ]
 [0.3651484 ]]]
```

mindspore.ops.max_pool2d

`mindspore.ops.max_pool2d(x, kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)`
二维最大值池化。

输入是 shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$ 的 Tensor，输出 (H_{in}, W_{in}) 维度中的最大值。给定 *kernel_size* $ks = (h_{ker}, w_{ker})$ 和 *stride* $s = (s_0, s_1)$ ，运算如下：

$$\text{output}(N_i, C_j, h, w) = \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} \text{input}(N_i, C_j, s_0 \times h + m, s_1 \times w + n)$$

参数：

- **x** (Tensor) - shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$ 的 Tensor。在 CPU 和 GPU 上，支持数据类型包括 int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32 和 float64。在 Ascend 上，数据类型仅支持 float16。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。可以是一个整数表示池化核的高度和宽度，或者包含两个整数的 tuple，分别表示池化核的高度和宽度。

- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长。可以是一个整数表示在高度和宽度方向的移动步长，或者包含两个整数的 tuple，分别表示在高度和宽度方向的移动步长。默认 None，表示移动步长为 *kernel_size*。
- **padding** (Union[int, tuple[int]]) - 池化填充长度。可以是一个整数表示在高度和宽度方向的填充长度，或者包含两个整数的 tuple，分别表示在高度和宽度方向的填充长度。默认为 0。
- **dilation** (Union[int, tuple[int]]) - 控制池化核内元素的间距。默认为 1。
- **return_indices** (bool) - 是否输出最大值的索引。默认为 False。
- **ceil_mode** (bool) - 是否使用 ceil 代替 floor 来计算输出的 shape。默认为 False。

返回：

当 *return_indices* 是 False 时，输出单个 *output* 张量，否则输出一个包含 *output* 张量和 *argmax* 张量的元组。

- **output** (Tensor) - 输出的池化后的最大值，shape 为 $(N_{out}, C_{out}, H_{out}, W_{out})$ 。其数据类型与 *x* 相同。

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 * padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

- **argmax** (Tensor) - 输出的最大值对应的索引，在 CPU 和 GPU 上，数据类型为 int64。在 Ascend 上，数据类型为 uint16。仅当 *return_indices* 为 True 的时候才返回该值。

异常：

- **TypeError** - *x* 不是 Tensor。
- **ValueError** - *x* 的维度不是 4D。
- **TypeError** - *kernel_size*、*stride*、*padding*、*dilation* 不是 int 或者 tuple。
- **ValueError** - *kernel_size*、*stride* 或者 *dilation* 的元素值小于 1。
- **ValueError** - *padding* 的元素值小于 0。
- **ValueError** - *padding* 的元素值大于 *kernel_size* 的一半。
- **TypeError** - *ceil_mode* 不是 bool 类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.arange(20 * 16 * 50 * 32).reshape((20, 16, 50, 32)), mindspore.float32)
>>> output_tensor, argmax = ops.max_pool2d(x, kernel_size=(3, 2), stride=(2, 1), return_
↪indices=True)
>>> print(output_tensor.shape)
(20, 16, 24, 31)
>>> print(argmax.shape)
(20, 16, 24, 31)
```

mindspore.ops.max_pool3d

`mindspore.ops.max_pool3d(x, kernel_size, stride=None, padding=0, dilation=1, ceil_mode=False, return_indices=False)`
 三维最大值池化。

输入是 shape 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor，输出 $(D_{out}, H_{out}, W_{out})$ 维度中的最大值。给定 $kernel\_size\ ks = (d_{ker}, h_{ker}, w_{ker})$ ，和 $stride\ s = (s_0, s_1, s_2)$ ，运算如下：

$$output(N_i, C_j, d, h, w) = \max_{l=0, \dots, d_{ker}-1} \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} input(N_i, C_j, s_0 \times d + l, s_1 \times h + m, s_2 \times w + n)$$

参数：

- **x** (Tensor) - shape 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor。支持数据类型包括 int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32 和 float64。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。可以是一个整数表示池化核的深度，高度和宽度，或者包含三个整数的 tuple，分别表示池化核的深度，高度和宽度。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长。可以是一个整数表示在深度，高度和宽度方向的移动步长，或者包含三个整数的 tuple，分别表示在深度，高度和宽度方向的移动步长。默认 None，表示移动步长为 $kernel\_size$ 。
- **padding** (Union[int, tuple[int]]) - 池化填充长度。可以是一个整数表示在深度，高度和宽度方向的填充长度，或者包含三个整数的 tuple，分别表示在深度，高度和宽度方向的填充长度。默认为 0。
- **dilation** (Union[int, tuple[int]]) - 控制池化核内元素的间距。默认为 1。
- **ceil_mode** (bool) - 是否是用 ceil 代替 floor 来计算输出的 shape。默认为 False。
- **return_indices** (bool) - 是否输出最大值的索引。默认为 False。

返回：

如果 `return_indices` 是 False，返回 Tensor `output`，否则返回 tuple (`output`, `argmax`)。

- **output** (Tensor) - 输出的池化后的最大值，shape 为 $(N_{out}, C_{out}, D_{out}, H_{out}, W_{out})$ 。其数据类型与 `x` 相同。

$$D_{out} = \left\lfloor \frac{D_{in} + 2 \times padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[2] - dilation[2] \times (kernel\_size[2] - 1) - 1}{stride[2]} + 1 \right\rfloor$$

- **argmax** (Tensor) - 输出的最大值对应的索引，数据类型为 int64。仅当 `return_indices` 为 True 的时候才返回该值。

异常：

- **TypeError** - `x` 不是 Tensor。
- **ValueError** - `x` 的维度不是 5D。
- **TypeError** - `kernel_size`、`stride`、`padding`、`dilation` 不是 int 或者 tuple。
- **ValueError** - `kernel_size` 或 `stride` 的元素值小于 1。
- **ValueError** - `padding` 的元素值小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.arange(2 * 1 * 2 * 2 * 2).reshape((2, 1, 2, 2, 2)), mindspore.float32)
>>> output_tensor, argmax = ops.max_pool3d(x, kernel_size=2, stride=1, padding=1, return_
    indices=True)
>>> print(output_tensor.shape)
(2, 1, 3, 3, 3)
>>> print(argmax.shape)
(2, 1, 3, 3, 3)
```

mindspore.ops.max_unpool1d

`mindspore.ops.max_unpool1d(x, indices, kernel_size, stride=None, padding=0, output_size=None)`
`max_pool1d` 的逆过程。

`max_unpool1d` 在计算过程中，保留最大值位置的元素，并将非最大值位置元素设置为 0。支持的输入数据格式为 (N, C, H_{in}) 或 (C, H_{in}) ，输出数据的格式为 (N, C, H_{out}) 或 (C, H_{out}) ，计算公式如下：

$$H_{out} = (H_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0]$$

参数：

- **x** (Tensor) - 待求逆的 Tensor。shape 为 (N, C, H_{in}) 或 (C, H_{in}) 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 x 相同。取值范围需满足 $[0, H_{in} - 1]$ 。数据类型必须是 int32 或 int64。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，若取值为 '0'，'(0)' 或 None，stride 值与 kernel_size 相同。默认 None，表示移动步长为 `kernel_size`。
- **padding** (Union[int, tuple[int]]) - 填充值。默认 0。
- **output_size** (tuple[int], 可选) - 输出 shape。默认 None。如果 output_size 为 ()，那么输出 shape 根据 `kernel_size`、`stride` 和 `padding` 计算得出。如果 output_size 不为 ()，那么 output_size 必须满足格式 (N, C, H) ， (C, H) 或 (H) ，取值范围需满足： $[(N, C, H_{out} - stride[0]), (N, C, H_{out} + stride[0])]$ 。

返回：

shape 为 (N, C, H_{out}) 或 (C, H_{out}) 的 Tensor，数据类型与输入 x 相同。

异常：

- **TypeError** - x 或 `indices` 的数据类型不支持。
- **TypeError** - `kernel_size`、`stride` 或 `padding` 既不是整数也不是 tuple。
- **ValueError** - `stride`、`padding` 或 `kernel_size` 的值不是非负的。
- **ValueError** - x 和 `indices` 的 shape 不一致。
- **ValueError** - x 的长度不是 2 或 3。
- **ValueError** - `output_size` 的类型不是 tuple。
- **ValueError** - `output_size` 的长度不为 0、2 或 3。
- **ValueError** - `output_size` 的取值与根据 `kernel_size`、`stride`、`padding` 计算得到的结果差距太大。

支持平台：

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[2, 4, 6, 8]]).astype(np.float32))
>>> indices = Tensor(np.array([[1, 3, 5, 7]]).astype(np.int64))
>>> output = ops.max_unpool1d(x, indices, kernel_size=2, stride=2, padding=0)
>>> print(output.asnumpy())
[[0. 2. 0. 4. 0. 6. 0. 8.]]
```

mindspore.ops.max_unpool2d

`mindspore.ops.max_unpool2d(x, indices, kernel_size, stride=None, padding=0, output_size=None)`
`max_pool2d` 的逆过程。

`max_unpool2d` 在计算过程中, 保留最大值位置的元素, 并将非最大值位置元素设置为 0。支持的输入数据格式为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) , 输出数据的格式为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) , 计算公式如下:

$$H_{out} = (H_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0]$$

$$W_{out} = (W_{in} - 1) \times stride[1] - 2 \times padding[1] + kernel\_size[1]$$

参数:

- **x** (Tensor) - 待求逆的 Tensor。shape 为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 x 相同。取值范围需满足 $[0, H_{in} \times W_{in} - 1]$ 。数据类型必须是 int32 或 int64。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。int 类型表示池化核的长宽相同。tuple 类型中的两个值分别代表池化核的长和宽。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长, int 类型表示长宽方向的移动步长相同。tuple 中的两个值分别代表长宽方向移动的步长。默认 None, 表示移动步长为 `kernel_size`。
- **padding** (Union[int, tuple[int]]) - 填充值。默认 0。若为 int 类型, 则长宽方向的填充大小相同, 均为 `padding`。若为 tuple 类型, 则 tuple 中的两个值分别代表长宽方向填充的大小。
- **output_size** (tuple[int], 可选) - 输出 shape。默认 None。如果 `output_size` 为 (), 那么输出 shape 根据 `kernel_size`、`stride` 和 `padding` 计算得出。如果 `output_size` 不为 (), 那么 `output_size` 必须满足格式 (N, C, H, W) , (C, H, W) 或 (H, W) , 取值范围需满足: $[(N, C, H_{out} - stride[0], W_{out} - stride[1]), (N, C, H_{out} + stride[0], W_{out} + stride[1])]$ 。

返回:

shape 为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) 的 Tensor, 数据类型与输入 x 相同。

异常:

- **TypeError** - x 或 `indices` 的数据类型不支持。
- **TypeError** - `kernel_size`、`stride` 或 `padding` 既不是整数也不是 tuple。
- **ValueError** - `stride`、`padding` 或 `kernel_size` 的值不是非负的。
- **ValueError** - x 和 `indices` 的 shape 不一致。
- **ValueError** - `kernel_size`、`stride` 或 `padding` 为 tuple 时长度不等于 2。
- **ValueError** - x 的长度不为 3 或 4。

- **ValueError** - *output_size* 的类型不是 tuple。
- **ValueError** - *output_size* 的长度不为 0、3 或 4。
- **ValueError** - *output_size* 的取值与根据 *kernel_size*、*stride*、*padding* 计算得到的结果差距太大。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> indices = Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> output = ops.max_unpool2d(x, indices, kernel_size=1, stride=1, padding=0)
>>> print(output.asnumpy())
[[[0. 1.]
  [8. 9.]]]]
```

mindspore.ops.max_unpool3d

`mindspore.ops.max_unpool3d(x, indices, kernel_size, stride=None, padding=0, output_size=None)`
`mindspore.ops.max_pool3d()` 的逆过程。

max_unpool3d 在计算过程中，保留最大值位置的元素，并将非最大值位置元素设置为 0。支持的输入数据格式为 $(N, C, D_{in}, H_{in}, W_{in})$ 或 $(C, D_{in}, H_{in}, W_{in})$ ，输出数据的格式为 $(N, C, D_{out}, H_{out}, W_{out})$ 或 $(C, D_{out}, H_{out}, W_{out})$ ，计算公式如下：

$$\begin{aligned} D_{out} &= (D_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0] \\ H_{out} &= (H_{in} - 1) \times stride[1] - 2 \times padding[1] + kernel\_size[1] \\ W_{out} &= (W_{in} - 1) \times stride[2] - 2 \times padding[2] + kernel\_size[2] \end{aligned}$$

参数：

- **x** (Tensor) - 待求逆的 Tensor。shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 或 $(C, D_{in}, H_{in}, W_{in})$ 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 *x* 相同。取值范围需满足 $[0, D_{in} \times H_{in} \times W_{in} - 1]$ 。数据类型必须是 int32 或 int64。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。int 类型表示池化核的深度、长和宽相同。tuple 类型中的三个值分别代表池化核的深度、长和宽。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，int 类型表示深度、长和宽方向的移动步长相同。tuple 中的三个值分别代表深度、长和宽方向移动的步长。默认 None，表示移动步长为 *kernel_size*。
- **padding** (Union[int, tuple[int]]) - 填充值。默认 0。若为 int 类型，则深度、长和宽方向的填充大小相同，均为 *padding*。若为 tuple 类型，则 tuple 中的三个值分别代表深度、长和宽方向填充的大小。
- **output_size** (tuple[int], 可选) - 输出 shape。默认 None。如果 *output_size* 为 ()，那么输出 shape 根据 *kernel_size*、*stride* 和 *padding* 计算得出。如果 *output_size* 不为 ()，那么 *output_size* 必须满足格式 (N, C, D, H, W) ， (C, D, H, W) 或 (D, H, W) ，取值范围需满足： $[(N, C, D_{out} - stride[0], H_{out} - stride[1], W_{out} - stride[2]), (N, C, D_{out} + stride[0], H_{out} + stride[1], W_{out} + stride[2])]$ 。

返回：

shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 或 $(C, D_{out}, H_{out}, W_{out})$ 的 Tensor，数据类型与输入 *x* 相同。

异常：

- **TypeError** - *x* 或 *indices* 的数据类型不支持。
- **TypeError** - *kernel_size*、*stride* 或 *padding* 既不是整数也不是 tuple。
- **ValueError** - *stride*、*kernel_size* 或 *padding* 的值不是非负的。
- **ValueError** - *x* 和 *indices* 的 shape 不一致。
- **ValueError** - *kernel_size*、*stride* 或 *padding* 为 tuple 时长度不等于 3。
- **ValueError** - *x* 的长度不为 4 或 5。
- **ValueError** - *output_size* 的类型不是 tuple。
- **ValueError** - *output_size* 的长度不为 0、4 或 5。
- **ValueError** - *output_size* 的取值与根据 *kernel_size*, *stride*, *padding* 计算得到的结果差距太大。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> indices= Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> output = ops.max_unpool3d(x, indices, kernel_size=2, stride=1, padding=0)
>>> print(output)
[[[[[0. 1. 8.]
      [9. 0. 0.]
      [0. 0. 0.]]
     [0. 0. 0.]
     [0. 0. 0.]
     [0. 0. 0.]
     [0. 0. 0.]]]]]
```

mindspore.ops.moe_token_permute

`mindspore.ops.moe_token_permute(tokens, indices, num_out_tokens=None, padded_mode=False)`

根据 *indices* 对 *tokens* 进行排列。具有相同索引的 token 将被分组在一起。

警告：

- 仅支持 Atlas A2 训练系列产品。
- 当前版本下，输入 *tokens* 仅支持 bfloat16 类型。
- *indices* 为二维的时候，第二维的 size 需要小于等于 512。

参数：

- **tokens** (Tensor) - 输入的 token Tensor，将被进行排列。数据类型是 bfloat16。shape 为 (*num_tokens*, *hidden_size*)，其中 *num_tokens* 和 *hidden_size* 是正整数。
- **indices** (Tensor) - 用于排列 token 的索引 Tensor。数据类型是 int32 或 int64。shape 为 (*num_tokens*, *topk*) 或 (*num_tokens*,)，其中 *num_tokens* 和 *topk* 是正整数。如果 shape 为后者，隐含 *topk* 为 1。
- **num_out_tokens** (int, 可选) - 有效的输出 token 数量，当启用容量因子时，应等于未丢弃的 token 数量，应该大于等于 0。默认 None，表示不丢弃任何 token。

- **padded_mode** (bool, 可选) - 如果为 True, 表示索引已填充, 用于表示每个专家选择的 token。目前只能设置为 False。默认 False。

返回:

tuple (Tensor), 表示 2 个 Tensor 组成的 tuple, 分别为排列后的 *tokens* 和排序后的 *indices*。

- **permuted_tokens** (Tensor) - 排列后的 Tensor, 数据类型与 *tokens* 相同。
- **sorted_indices** (Tensor) - 索引 Tensor, 数据类型为 int32, 表示排列后的 Tensor 对应的索引。

异常:

- **TypeError** - 如果 *tokens* 或 *indices* 不是 Tensor。
- **TypeError** - 如果 *tokens* 的数据类型不是 bfloat16。
- **TypeError** - 如果 *indices* 的数据类型不是 int32 或 int64。
- **TypeError** - 如果指定的 *num_out_tokens* 不是整数。
- **TypeError** - 如果指定的 *padded_mode* 不是布尔值。
- **ValueError** - 如果 *indices* 的第二维度大于 512 (如果存在)。
- **ValueError** - 如果 *padded_mode* 设置为 True。
- **ValueError** - 如果 *tokens* 不是 2 维, 或 *indices* 不是 1 维或 2 维的 Tensor。
- **RuntimeError** - 如果 *tokens* 和 *indices* 的第一维度不一致。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> tokens = Tensor([[1, 1, 1],
...                  [7, 7, 7],
...                  [2, 2, 2],
...                  [1, 1, 1],
...                  [2, 2, 2],
...                  [3, 3, 3]], dtype=mindspore.bfloat16)
>>> indices = Tensor([5, 0, 3, 1, 2, 4], dtype=mindspore.int32)
>>> out = ops.moe_token_permute(tokens, indices)
>>> print(out)
(Tensor(shape=[6, 3], dtype=BFLOAT16, value=
[[7, 7, 7],
 [1, 1, 1],
 [2, 2, 2],
 [2, 2, 2],
 [3, 3, 3],
 [1, 1, 1]]), Tensor(shape=[6], dtype=INT32, value= [5, 0, 3, 1, 2, 4]))
```

mindspore.ops.moe_token_unpermute

`mindspore.ops.moe_token_unpermute` (*permuted_tokens*, *sorted_indices*, *probs=None*, *padded_mode=False*,
restore_shape=None)

根据排序的索引对已排列的标记进行反排列，并可选择将标记与其对应的概率合并。

警告：

- 仅支持 Atlas A2 训练系列产品。
- 当前版本下，输入 *permuted_tokens* 和 *probs* 仅支持 `bfloat16` 类型。
- *sorted_indices* 必须不存在重复值，否则结果未定义。

参数：

- **permuted_tokens** (Tensor) - 要进行反排列的已排列标记的 Tensor。shape 为 `[num_tokens * topkhidden_size]`，其中 *num_tokens*、*topk* 和 *hidden_size* 都是正整数。
- **sorted_indices** (Tensor) - 用于反排列标记的排列索引 Tensor。shape 为 `[num_tokens * topk]`，其中 *num_tokens* 和 *topk* 都是正整数。仅支持 `int32` 的数据类型。
- **probs** (Tensor, 可选) - 与已排列标记对应的概率 Tensor。如果提供，反排列的标记将与其相应的概率合并。shape 为 `[num_tokenstopk]`，其中 *num_tokens* 和 *topk* 都是正整数。默认 `None`。
- **padded_mode** (bool, 可选) - 如果为 `True`，表示索引被填充，以表示每个专家选择的标记。默认 `False`。
- **restore_shape** (Union[tuple[int], list[int]], 可选) - 排列之前的输入形状，仅在填充模式下使用。默认 `None`。

返回：

Tensor，类型与 *permuted_tokens* 一致。如果 *padded_mode* 为 `False`，则 shape 为 `[num_tokens, hidden_size]`。如果 *padded_mode* 为 `True`，则 shape 由 *restore_shape* 指定。

异常：

- **TypeError** - *permuted_tokens* 不是 Tensor。
- **ValueError** - 仅支持 *padded_mode* 为 `False`。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> permuted_token = Tensor([
...     [1, 1, 1],
...     [0, 0, 0],
...     [0, 0, 0],
...     [3, 3, 3],
...     [2, 2, 2],
...     [1, 1, 1],
...     [2, 2, 2],
...     [3, 3, 3]], dtype=mindspore.bfloat16)
>>> sorted_indices = Tensor([0, 6, 7, 5, 3, 1, 2, 4], dtype=mindspore.int32)
>>> out = ops.moe_token_unpermute(permuted_token, sorted_indices)
>>> out.shape
(8, 3)
```

mindspore.ops.incre_flash_attention

`mindspore.ops.incre_flash_attention` (*query, key, value, attn_mask=None, actual_seq_lengths=None, pse_shift=None, dequant_scale1=None, quant_scale1=None, dequant_scale2=None, quant_scale2=None, quant_offset2=None, antiquant_scale=None, antiquant_offset=None, block_table=None, num_heads=1, input_layout='BSH', scale_value=1.0, num_key_value_heads=0, block_size=0, inner_precise=1, kv_padding_size=None*)

增量推理场景接口。

- **b**: batch 维
- **N**: 注意力头数
- **kvN**: *key / value* 头数
- **S**: 序列长度
- **D**: 头维度
- **H**: 隐藏层大小
- **kvH**: *key / value* 的隐藏层大小

其中 $H = N \times D$, $kvH = kvN \times D$ 。

self-attention (自注意力) 利用输入样本自身的关系构建了一种注意力模型。其原理是假设有一个长度为 n 的输入样本序列 x , x 的每个元素都是一个 d 维向量, 可以将每个 d 维向量看作一个 **token embedding**, 将这样一条序列经过 3 个权重矩阵变换得到 3 个维度为 $n \times d$ 的矩阵。

self-attention 的计算公式一般定义如下,

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d}})V$$

其中 Q 和 K^T 的乘积代表输入 x 的注意力, 为避免该值变得过大, 通常除以 d 的平方根进行缩放, 并对每行进行 softmax 归一化, 与 V 相乘后得到一个 $n \times d$ 的矩阵。

说明:

- 如果没有输入参数且没有默认值, 需要传入 `None`。
- 参数 *key* 和 *value* 对应的 **Tensor** 的 **shape** 需要完全一致。
- 参数 *query* 的 N 等于 *num_heads*, 而参数 *key* 和 *value* 的 N 等于 *num_key_value_heads*。*num_heads* 必须是 *num_key_value_heads* 的倍数。
- 量化
 - 当 *query*、*key* 和 *value* 的数据类型为 `float16`, 且输出为 `int8` 时, 需要传入 *quant_scale2*, *quant_offset2* 可选。
 - 当 *antiquant_scale* 存在时, *key* 和 *value* 需要以 `int8` 传入, *antiquant_offset* 可选。
 - *antiquant_scale* 和 *antiquant_offset* 的数据类型需要与 *query* 一致。
- *pse_shift*:
 - *pse_shift* 的数据类型需与 *query* 一致, 并且仅支持 **D** 轴对齐, 即 **D** 轴可以被 16 整除。
- **Page attention**:
 - 开启 **page attention** 的必要条件是 *block_table* 存在, 并且 *key* 和 *value* 根据 *block_table* 中的索引在内存中连续排列。支持的 *key* 和 *value* 数据类型为 `float16/bfloat16/int8`。

- 在 page attention 开启场景下, 当 *key* 和 *value* 的输入类型为 float16/bfloat16 时, 需要 16 对齐; 当输入类型为 int8 时, 需要 32 对齐, 推荐使用 128 对齐。
- *block_table* 当前支持的 *max_block_num_per_seq* 最大为 16k, 超过 16k 会导致拦截和报错; 如果 *S* 过大导致 *max_block_num_per_seq* 超过 16k, 可通过增大 *block_size* 来解决。
- 在 page attention 场景中, *key* 和 *value* 的 Tensor 的 shape 各维乘积不能超过 int32 的表示范围。
- 在进行 per-channel 后量化时, 不能同时启用 page attention。
- *kv_padding_size*:
 - KV 缓存传输起点的计算公式为 $S - kv\_padding\_size - actual\_seq\_lengths$, KV 缓存传输终点的计算公式为 $S - kv\_padding\_size$ 。当起点或终点小于 0 时, 返回的数据结果为全 0。
 - 当 *kv_padding_size* 小于 0 时, 会设置为 0。
 - *kv_padding_size* 需要与 *actual_seq_lengths* 参数一同启用, 否则被认为是 KV 右填充场景。
 - 需要与 *attn_mask* 参数一同启用, 并确保 *attn_mask* 的意义正确, 即能正确隐藏无效数据, 否则会引入精度问题。
 - *kv_padding_size* 不支持 page attention 场景。

警告: 只支持 Atlas A2 训练系列产品。

参数:

- **query** (Tensor) - 公式中的输入 Q, 数据类型 float16 或 bfloat16。shape 为 $(B, 1, H) / (B, N, 1, D)$ 。
- **key** (Union[tuple, list]) - 公式中的输入 K, 数据类型为 float16、bfloat16 或 int8。shape 为 $(B, S, kvH) / (B, kvN, S, D)$ 。
- **value** (Union[tuple, list]) - 公式中的输入 V, 数据类型为 float16、bfloat16 或 int8。shape 为 $(B, S, kvH) / (B, kvN, S, D)$ 。
- **attn_mask** (Tensor, 可选) - 注意力掩码 Tensor, 数据类型为 bool、int8 或 uint8。shape 为 $(B, S) / (B, 1, S) / (B, 1, 1, S)$ 。默认 None。
- **actual_seq_lengths** (Union[Tensor, tuple[int], list[int]], 可选) - 描述每个输入的实际序列长度, 数据类型为 int64。shape 为 $(B,)$ 。默认值: None。
- **pse_shift** (Tensor, 可选) - 位置编码 Tensor, 数据类型为 float16 或 bfloat16。输入 Tensor shape 为 $(1, N, 1, S) / (B, N, 1, S)$ 。默认值: None。
- **dequant_scale1** (Tensor, 可选) - 量化参数, 数据类型为 uint64 或 float32。当前未使能。默认值: None。
- **quant_scale1** (Tensor, 可选) - 量化参数, 数据类型为 float32。当前未使能。默认值: None。
- **dequant_scale2** (Tensor, 可选) - 量化参数, 数据类型为 uint64 或 float32。当前未使能。默认值: None。
- **quant_scale2** (Tensor, 可选) - 后量化参数, 数据类型为 float32。shape 为 $(1,)$ 。默认值: None。
- **quant_offset2** (Tensor, 可选) - 后量化参数, 数据类型为 float32。shape 为 $(1,)$ 。默认值: None。
- **antiquant_scale** (Tensor, 可选) - 伪量化参数, 数据类型为 float16 或 bfloat16。当 *input_layout* 为 'BNSD' 时, shape 为 $(2, kvN, 1, D)$; 当 *input_layout* 为 'BSH' 时, shape 为 $(2, kvH)$ 。默认值: None。
- **antiquant_offset** (Tensor, 可选) - 伪量化参数, 数据类型为 float16 或 bfloat16。当 *input_layout* 为 'BNSD' 时, shape 为 $(2, kvN, 1, D)$; 当 *input_layout* 为 'BSH' 时, shape 为 $(2, kvH)$ 。默认值: None。
- **block_table** (Tensor, 可选) - Tensor, 数据类型为 int32。shape 为 $(B, max\_block\_num\_per\_seq)$, 其中 $max\_block\_num\_per\_seq = \text{ceil}(\frac{max(actual\_seq\_length)}{block\_size})$ 。默认值: None。

- **num_heads** (int, 可选) - 头的数量。默认值: 1。
- **input_layout** (str, 可选) - 输入 qkv 的数据布局, 支持 'BSH' 和 'BNSD'。默认值: 'BSH'。
- **scale_value** (double, 可选) - 表示缩放系数的值, 用作计算中的标量。默认值: 1.0。
- **num_key_value_heads** (int, 可选) - 用于 GQA 算法的 *key / value* 头数。如果 *key* 和 *value* 具有相同的头数, 则值为 0, 使用 *num_heads*。默认值: 0。
- **block_size** (int, 可选) - 页注意力中存储在每个 KV 块中的最大标记数。默认值: 0。
- **inner_precise** (int, 可选) - 一个 {0, 1} 中的整数, 指定计算模式。0 为高精度模式 (适用于 float16 数据类型), 1 为高性能模式。默认值为 1。
- **kv_padding_size** (Tensor, 可选) - Tensor, 数据类型为 int64。值的范围为 $0 \leq kv\_padding\_size \leq S - \max(actual\_seq\_length)$ 。shape 为 () 或 (1,)。默认值: None。

返回:

attention_out (Tensor), 注意力输出 Tensor, shape 为 $(B, 1, H) / (B, N, 1, D)$ 。

异常:

- **TypeError** - *query* 的数据类型不是 float16 或 bfloat16。
- **TypeError** - *key* 和 *value* 的数据类型不同。
- **TypeError** - *attn_mask* 的数据类型不是 bool、int8 或 uint8。
- **TypeError** - *pse_shift* 的数据类型不是 bfloat16 或 float16。
- **TypeError** - *scale_value* 不是 double 类型。
- **TypeError** - *input_layout* 不是字符串。
- **TypeError** - *num_key_value_heads* 或 *num_heads* 不是整数。
- **TypeError** - *inner_precise* 不是整数。
- **TypeError** - *quant_scale1* 不是 float32 类型的 Tensor。
- **TypeError** - *quant_scale2* 不是 float32 类型的 Tensor。
- **TypeError** - *quant_offset2* 不是 float32 类型的 Tensor。
- **ValueError** - *actual_seq_lengths* 的长度不是 1 或者 B。
- **ValueError** - *input_layout* 是字符串但不是 BSH 或 BNSD。
- **ValueError** - *num_heads* 不能被 *Q_H* 整除。
- **ValueError** - *num_heads* 不能被 *num_key_value_heads* 整除。
- **RuntimeError** - *num_heads* 小于等于 0。
- **RuntimeError** - *attn_mask* 的 shape 不合法。

支持平台:

Ascend

样例:

```
>>> from mindspore import ops
>>> from mindspore.common import Tensor
>>> from mindspore.common import dtype as mstype
>>> import numpy as np
>>> B, N, S, D, kvN = 1, 4, 10, 128, 1
```

(下页继续)

(续上页)

```
>>> query = Tensor(np.random.randn(B, 1, N * D), mstype.float16)
>>> key = [Tensor(np.random.randn(B, S, kvN * D), mstype.float16)]
>>> value = [Tensor(np.random.randn(B, S, kvN * D), mstype.float16)]
>>> ifa_ms = ops.incre_flash_attention
>>> attn_out = ifa_ms(query, key, value, num_heads=N, num_key_value_heads=kvN)
>>> attn_out
Tensor(shape=[1, 1, 512], dtype=Float16, value=
[[[ 1.6104e+00,  7.3438e-01,  1.0684e+00 ... -8.7891e-01,  1.7695e+00,  1.0264e+00]]])
```

`mindspore.ops.prompt_flash_attention`

`mindspore.ops.prompt_flash_attention` (*query, key, value, attn_mask=None, actual_seq_lengths=None, actual_seq_lengths_kv=None, pse_shift=None, deq_scale1=None, quant_scale1=None, deq_scale2=None, quant_scale2=None, quant_offset2=None, num_heads=1, scale_value=1.0, pre_tokens=2147483647, next_tokens=0, input_layout='BSH', num_key_value_heads=0, sparse_mode=0, inner_precise=1*)

全量推理场景接口。

- B: batch 维
- N: 注意力头数
- S: 序列长度
- D: 头维度
- H: 隐藏层大小

self-attention（自注意力）利用输入样本自身的关系构建了一种注意力模型。其原理是假设有一个长度为 n 的输入样本序列 x ， x 的每个元素都是一个 d 维向量，可以将每个 d 维向量看作一个 token embedding，将这样一条序列经过 3 个权重矩阵变换得到 3 个维度为 $n \times d$ 的矩阵。

self-attention 的计算公式一般定义如下，

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d}})V$$

其中 Q 和 K^T 的乘积代表输入 x 的注意力，为避免该值变得过大，通常除以 d 的平方根进行缩放，并对每行进行 softmax 归一化，与 V 相乘后得到一个 $n \times d$ 的矩阵。

警告:

- 未来将不再支持 `attn_mask` 的 float16 数据类型。
- 当 `sparse_mode` 为 2、3 或 4 时，`attn_mask` 的 shape 必须为 (2048, 2048) / (B, 1, 2048, 2048) / (1, 1, 2048, 2048)。

说明:

- 各轴的最大支持值:
 - 支持 B 轴值小于等于 65536(64k)。当输入类型包括 D 轴未对齐到 32 的 int8 或 float16 或 bfloat16 时，B 轴支持最多为 128。
 - 支持 N 轴值小于等于 256。

- 支持 S 轴值小于等于 20971520(20M)。
- 支持 D 轴值小于等于 512。
- 量化:
 - int8 输入, int8 输出: 必须提供 *deq_scale1*、*quant_scale1*、*deq_scale2* 和 *quant_scale2* 参数。*quant_offset2* 是可选的 (如果未提供, 默认值为 0)。
 - int8 输入, float16 输出: 必须提供 *deq_scale1*、*quant_scale1* 和 *deq_scale2* 参数。如果提供 *quant_offset2* 或 *quant_scale2*, 将导致错误。
 - float16 或 bfloat16 输入, int8 输出: 必须提供 *quant_scale2* 参数。*quant_offset2* 是可选的 (如果未提供, 默认值为 0)。如果提供 *deq_scale1*、*quant_scale1* 或 *deq_scale2* 参数, 将导致错误。
 - int8 输出:
 - * per-channel 格式的 *quant_scale2* 和 *quant_offset2* 不支持有左填充、Ring Attention 或 D 轴未对齐到 32 字节的情况。
 - * 在 GE 模式中, per-tensor 格式的 *quant_scale2* 和 *quant_offset2* 不支持 D 轴未对齐到 32 字节的情况。
 - * 不支持带有负值的带状稀疏和 *pre_tokens/next_tokens*。
 - *quant_scale2* 和 *quant_offset2* 只有在 *query* 为 bfloat16 时才可以是 bfloat16。
- 其他使用注意事项:
 - *query* 参数的 *N* 必须等于 *num_heads*。*key* 和 *value* 参数的 *N* 必须等于 *num_key_value_heads*。
 - *num_heads* 必须可以被 *num_key_value_heads* 整除, 并且 *num_heads* 除以 *num_key_value_heads* 不得大于 64。
 - 当 *query* 数据类型为 bfloat16 时, D 轴应与 16 对齐。
 - *actual_seq_lengths* 的每个元素不得超过 *q_S*, *actual_seq_lengths_kv* 的每个元素不得超过 *kv_S*。

警告: 只支持 Atlas A2 训练系列产品。

参数:

- **query** (Tensor) - 公式中的输入 Q, 数据类型可以是 int8、float16 或 bfloat16。shape 为 (B, q_S, q_H) 或 (B, q_N, q_S, q_D) 。
- **key** (Tensor) - 公式中的输入 K, 数据类型与 *query* 相同。shape 为 (B, kv_S, kv_H) 或 (B, kv_N, kv_S, kv_D) 。
- **value** (Tensor) - 公式中的输入 V, 数据类型与 *query* 相同。shape 为 (B, kv_S, kv_H) 或 (B, kv_N, kv_S, kv_D) 。
- **attn_mask** (Tensor, 可选) - 注意力掩码 Tensor, 数据类型为 bool、int8、uint8 或 float16。每个元素中, 0/False 表示保留, 1/True 表示丢弃。如果 *sparse_mode* 为 0 或 1, 其 shape 可以是 (q_S, kv_S) 、 (B, q_S, kv_S) 、 $(1, q_S, kv_S)$ 、 $(B, 1, q_S, kv_S)$ 或 $(1, 1, q_S, kv_S)$ 。如果 *sparse_mode* 为 2、3 或 4, 其 shape 应为 $(2048, 2048)$ 、 $(1, 2048, 2048)$ 或 $(1, 1, 2048, 2048)$ 。默认值为 None。
- **actual_seq_lengths** (Union[Tensor, tuple[int], list[int]], 可选) - 描述 *query* 每个批次的实际序列长度, 数据类型为 int64。shape 为 $(B,)$, 每个元素应为正整数。默认值为 None。
- **actual_seq_lengths_kv** (Union[Tensor, tuple[int], list[int]], 可选) - 描述 *key* 或 *value* 每个批次的实际序列长度, 数据类型为 int64。shape 为 $(B,)$, 每个元素应为正整数。默认值为 None。
- **pse_shift** (Tensor, 可选) - 位置编码 Tensor, 数据类型为 float16 或 bfloat16。输入 Tensor shape 为 (B, N, q_S, kv_S) 或 $(1, N, q_S, kv_S)$ 。默认值为 None。
 - *q_S* 必须大于等于 *query* 的 S 长度, *kv_S* 必须大于等于 *key* 的 S 长度。

- 如果 *pse_shift* 的数据类型为 float16, *query* 应为 float16 或 int8, 这种情况下会自动启用高精度模式。
- 如果 *pse_shift* 的数据类型为 bfloat16, *query* 应为 bfloat16。
- **deq_scale1** (Tensor, 可选) - 量化参数, 数据类型为 uint64 或 float32。输入 Tensor shape 为 (1,)。默认值为 None。
- **quant_scale1** (Tensor, 可选) - 量化参数, 数据类型为 float32。输入 Tensor shape 为 (1,)。默认值为 None。
- **deq_scale2** (Tensor, 可选) - 量化参数, 输入 Tensor shape 为 (1,) 并与 *deq_scale1* 类型相同。默认值为 None。
- **quant_scale2** (Tensor, 可选) - 量化参数, 数据类型为 float32 或 bfloat16。当输出 layout 为 BSH 时, 建议 shape 为 (1,)、(1, 1, q_H)、(q_H ,); 当 layout 为 BNSD 时, 建议 shape 为 (1,)、(1, q_N , 1, D)、(q_N , D)。默认值为 None。
- **quant_offset2** (Tensor, 可选) - 量化参数, 数据类型为 float32 或 bfloat16。数据类型和 shape 与 *quant_scale2* 相同。默认值为 None。
- **num_heads** (int, 可选) - 头的数量, 范围为 [0, 256]。默认值为 1。
- **scale_value** (double, 可选) - 表示缩放系数的值, 用作计算中的乘数标量。默认值为 1.0。
- **pre_tokens** (int, 可选) - 用于稀疏计算, 表示注意力需要关联的前序列元素个数。默认值为 2147483647。
- **next_tokens** (int, 可选) - 用于稀疏计算, 表示注意力需要关联的后序列元素个数。默认值为 0。
- **input_layout** (str, 可选) - 输入 qkv 的数据 layout, 支持 BSH 和 BNSD。默认值为 BSH。
- **num_key_value_heads** (int, 可选) - 一个整数, 表示在 GQA 算法中 *key* 和 *value* 的头数量。0 表示 *key* 和 *value* 具有与 *query* 相同的头数。如果指定 (非 0), 其必须是 *num_heads* 的因子, 并且等于 *kv_n*。默认值为 0。
- **sparse_mode** (int, 可选) - 一个整数, 指定稀疏模式, 可以是 {0, 1, 2, 3, 4} 中的值。默认值为 0。
 - *sparseMode* = 0: 如果 *attn_mask* 为空指针, 则忽略 *pre_tokens* 和 *next_tokens* 输入 (内部设置为 INT_MAX)。
 - *sparseMode* = 2, 3, 4: *attn_mask* shape 必须为 (S , S)、(1, S , S) 或 (1, 1, S , S), S 固定为 2048。用户必须确保 *attn_mask* 为下三角。shape 不正确或未提供将导致错误。
 - *sparseMode* = 1, 2, 3: 忽略 *pre_tokens* 和 *next_tokens* 输入, 并根据特定规则设置值。
 - *sparseMode* = 4: *pre_tokens* 和 *next_tokens* 必须是非负的。
- **inner_precise** (int, 可选) - 一个 {0, 1} 中的整数, 指定计算模式。0 为高精度模式 (适用于 float16 数据类型), 1 为高性能模式。默认值为 1。

返回:

- **attention_out** (Tensor) - 输出 Tensor, 与 *query* 的 shape 相同: (B , q_S , q_H) 或 (B , q_N , q_S , q_D)。输出数据类型由多种因素决定, 请参阅上文说明部分获取详细信息。

异常:

- **TypeError** - *query* 的数据类型不是 int8、float16 或 bfloat16。
- **TypeError** - *query*、*key* 和 *value* 的数据类型不同。
- **TypeError** - *attn_mask* 的数据类型不是 bool、int8 或 uint8。
- **TypeError** - *pse_shift* 的数据类型不是 bfloat16 或 float16。
- **TypeError** - *scale_value* 不是 double 类型。
- **TypeError** - *input_layout* 不是字符串。
- **TypeError** - *num_key_value_heads* 不是整数。
- **TypeError** - *sparse_mode* 不是整数。
- **TypeError** - *inner_precise* 不是整数。

- **TypeError** - *quant_scale1* 不是 float32 类型的 Tensor。
- **TypeError** - *deq_scale1* 不是 uint64 或 float32 类型的 Tensor。
- **TypeError** - *quant_scale2* 不是 float32 类型的 Tensor。
- **TypeError** - *deq_scale2* 不是 uint64 或 float32 类型的 Tensor。
- **TypeError** - *quant_offset2* 不是 float32 类型的 Tensor。
- **ValueError** - *input_layout* 是字符串但不是 BSH 或 BNSD。
- **RuntimeError** - *num_heads* 不能被 *num_key_value_heads* 整除。
- **RuntimeError** - *num_heads* 小于等于 0。
- **RuntimeError** - *num_key_value_heads* 小于等于 0。
- **RuntimeError** - *kv_n* 不等于 *num_key_value_heads*。
- **RuntimeError** - *attn_mask* 的 shape 不合法。
- **RuntimeError** - *sparse_mode* 被指定的值不是 0、1、2、3 或 4。
- **RuntimeError** - *query* 的数据类型为 bfloat16 并且 D 轴未对齐到 16。
- **RuntimeError** - 输入 layout 为 BSH 并且 *kv_h* 不能被 *num_key_value_heads* 整除。
- **RuntimeError** - *query*、*key* 和 *value* 的 D 轴不相同。
- **RuntimeError** - 后量化 per-channel 情况下, D 轴未对齐到 32 字节。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> B = 1
>>> N = 16
>>> S = 256
>>> D = 16
>>> query = Tensor(np.ones((B, N, S, D), dtype=np.float16))
>>> key = Tensor(np.ones((B, N, S, D), dtype=np.float16))
>>> value = Tensor(np.ones((B, N, S, D), dtype=np.float16))
>>> out = ops.prompt_flash_attention(query, key, value, None, None, None, None, None, None,
↪None, None,
...                               None, N, input_layout='BNSD')
>>> print(out.shape)
(1, 16, 256, 16)
```

mindspore.ops.flash_attention_score

```
mindspore.ops.flash_attention_score(query, key, value, head_num, real_shift=None, drop_mask=None,
                                     padding_mask=None, attn_mask=None, prefix=None, actual_seq_glen=None,
                                     actual_seq_kvlen=None, keep_prob=1.0, scalar_value=1.0,
                                     pre_tokens=2147483647, next_tokens=2147483647, inner_precise=0,
                                     input_layout='BSH', sparse_mode=0)
```

训练场景下，使用 FlashAttention 算法实现 self-attention 的计算。

- B: batch 维，取值范围 1 至 2k。
- S1: *query* 的序列长度，取值范围 1 至 512k。
- S2: *key* 和 *value* 的序列长度，取值范围 1 至 512k。
- N1: *query* 的多头数量，取值范围 1 至 256。
- N2: *key* 和 *value* 的多头数量，且 N2 必须为 N1 的因子。
- D: 单头大小，取值范围为 16 的倍数，最大值为 512。
- H1: *query* 的隐藏层大小，等于 $N1 * D$ 。
- H2: *key* 和 *value* 的隐藏层大小，等于 $N2 * D$ 。

`flash_attention_score` 的计算公式一般定义如下：

$$\text{attention\_out} = \text{Dropout}(\text{Softmax}(\text{Mask}(\text{scale} * (\text{query} * \text{key}^T) + \text{pse}), \text{attn\_mask}), \text{keep\_prob}) * \text{value}$$

警告：

- 这是一个实验性 API，可能会发生变更或被删除。
- 只支持 Atlas A2 训练系列产品。

参数：

- **query** (Tensor) - query Tensor, shape 支持: $(B, S1, H1)$ 、 $(B, N1, S1, D)$ 、 $(S1, B, H1)$ 、 $(B, S1, N1, D)$ 或 $(T1, N1, D)$ 。支持的数据类型为 float16 以及 bfloat16。
- **key** (Tensor) - key Tensor, 数据类型与 *query* 一致。shape 支持: $(B, S2, H2)$ 、 $(B, N2, S2, D)$ 、 $(S2, B, H2)$ 、 $(B, S2, N2, D)$ 或 $(T2, N2, D)$ 。
- **value** (Tensor) - value Tensor, 数据类型以及 shape 与 *key* 一致。
- **head_num** (int) - *query* 的多头数量，等于 N1。
- **real_shift** (Tensor, 可选) - 位置编码 Tensor, 也叫做 *pse*, 数据类型与 *query* 一致。默认值: None。如果 *S* 大于 1024 并且使用了下三角的掩码，则仅使用下三角的逆向 1024 行以进行内存优化。支持的 shape 为: $(B, N1, S1, S2)$ 、 $(1, N1, S1, S2)$ 、 $(B, N1, 1024, S2)$ 或 $(1, N1, 1024, S2)$ 。
 - ALiBi 场景: *real_shift* 必须满足 ALiBi 规则，并且在下三角场景下 *sparse_mode* 为 2 或 3。在此场景中，*real_shift* 的 shape 为 $(B, N1, 1024, S2)$ 或 $(1, N1, 1024, S2)$ 。
 - 非 ALiBi 场景: *real_shift* 的 shape 为 $(B, N1, S1, S2)$ 或 $(1, N1, S1, S2)$ 。
 - *input_layout* 为 TND 时: shape 应为 $(B, N1, 1024, S2)$ 或 $(1, N1, 1024, S2)$ 。
- **drop_mask** (Tensor, 可选) - Dropout 掩码 Tensor, 数据类型为 uint8, shape 为 $(B, N1, S1, S2//8)$ 。当不为 None 时，*S2* 需要为 8 的倍数。默认值为: None。
- **padding_mask** (Tensor, 可选) - 保留参数，尚未使能。默认值为: None。

- **attn_mask** (Tensor, 可选) - 注意力掩码 Tensor, 数据类型为 bool、uint8。对于每个元素, 0/False 表示保留, 1/True 表示丢弃。shape 可以是 $(B, N1, S1, S2)$ 、 $(B, 1, S1, S2)$ 、 $(S1, S2)$ 或 $(2048, 2048)$ 。默认值为: None。
 - 在压缩场景中, 当 *sparse_mode* 为 2、3 或 4 时, *attn_mask* 必须为 $(2048, 2048)$ 。
 - 当 *sparse_mode* 为 5 时, *attn_mask* 应该为 $(B, N1, S1, S2)$ 或 $(B, 1, S1, S2)$ 。
 - 当 *sparse_mode* 为 0 和 1 时, *attn_mask* 应该是 $(B, N1, S1, S2)$ 、 $(B, 1, S1, S2)$ 或 $(S1, S2)$ 。
- **prefix** (Union[Tensor, tuple[int], list[int]], 可选) - 在前缀稀疏计算场景下, 每个 batch 的 N 值。输入为 Tensor 时的 shape 为 $(B,)$, 其中 B 的最大值为 32。仅当 *sparse_mode* 为 5 时, 此参数不为 None。默认值为: None。如果 $S1 > S2$, N 的范围为 0 到 $S2$ 。如果 $S1 \leq S2$, N 的范围为 $S2 - S1$ 到 $S2$ 。
- **actual_seq_qlen** (Union[Tensor, tuple[int], list[int]], 可选) - 每个 batch 中 query 序列的大小, 用一个递增值数组表示, 最后一个值等于 T1。默认值为: None。
- **actual_seq_kvlen** (Union[Tensor, tuple[int], list[int]], 可选) - 每个 batch 中 key 和 value 序列的大小, 用一个递增值数组表示, 最后一个值等于 T2。默认值为: None。
- **keep_prob** (double, 可选) - Dropout 的保留概率, 取值范围为 $(0.0, 1.0]$ 。当 *keep_prob* 为 1.0 时, *drop_mask* 应为 None。默认值为: 1.0。
- **scalar_value** (double, 可选) - 缩放因子。通常, 该值为 $1.0 / (D ** 0.5)$ 。默认值为: 1.0。
- **pre_tokens** (int, 可选) - 用于稀疏计算, 表示向前计算的 token 数量。当 *sparse_mode* 设置为 1、2、3 或 5 时, 此参数无效。默认值为: 2147483647。
- **next_tokens** (int, 可选) - 稀疏计算的参数, 表示向后计算的 token 数量。当 *sparse_mode* 设置为 1、2、3 或 5 时, 此参数无效。默认值为: 2147483647。*pre_tokens* 的值对应于 $S1$, 而 *next_tokens* 的值对应于 $S2$ 。它们定义了 *attn_mask* 矩阵上的有效区域, 并必须确保带宽不为空。以下情况是非法的:
 - $pre\_tokens < 0$ 且 $next\_tokens < 0$ 。
 - $(pre\_tokens < 0$ 且 $next\_tokens \geq 0)$ 且 $(next\_tokens < abs(pre\_tokens)$ 或 $abs(pre\_tokens) \geq S2)$ 。
 - $(pre\_tokens \geq 0$ 且 $next\_tokens < 0)$ 且 $(abs(next\_tokens) > pre\_tokens$ 或 $abs(next\_tokens) \geq S1)$ 。
- **inner_precise** (int, 可选) - 保留参数, 尚未使能。默认值为: 0。
- **input_layout** (str, 可选) - 指定输入 *query*、*key* 和 *value* 的数据格式。值可以为 "BSH"、"BNSD"、"SBH"、"BSND" 或 "TND"。其中 "TND" 是实验性格式。默认值为: "BSH"。

当 *input_layout* 为 "TND" 时, 必须满足以下限制: 假设有两个列表 *list_seq_q* 和 *list_seq_k* 表示输入序列的长度。每个列表的值表示 batch 中每个序列的长度。例如, *list_seq_q* = [4, 2, 6], *list_seq_k* = [10, 3, 9]。T1 = sum(*list_seq_q*) = 12, T2 = sum(*list_seq_k*) = 22。max_seqlen_q = max(*list_seq_q*), max_seqlen_k = max(*list_seq_k*)。qk_pointer = sum(*list_seq_q* * *list_seq_k*), 表示元素乘积的总和。

- 两个列表的长度需要相同, 列表大小等于 batch 数量。batch 数量最大为 1024。
- 当 *input_layout* 为 "TND" 时, *actual_seq_qlen* 和 *actual_seq_kvlen* 必须不为 None。否则, 它们为 None。
- *actual_seq_qlen* 和 *actual_seq_kvlen* 是 key/value 序列的累计和, 因此必须是非递减的。
- 如果 *real_shift* 不为 None, 则 *list_seq_q* 和 *list_seq_k* 必须相同。*list_seq_q* 和 *list_seq_k* 的最大值必须大于 1024。*real_shift* 应为 $(B, N1, 1024, S2)$ 和 $(1, N1, 1024, S2)$, 其中 $S2$ 等于 max_seqlen_k。
- *attn_mask* 必须是下三角矩阵, 因此 *sparse_mode* 应为 2 或 3, 且 *attn_mask* 的 shape 应为 $(2048, 2048)$ 。
- *drop_mask* 的 shape 为 $(qk\_pointer * N1 / 8,)$ 。
- *prefix* 为 None。
- *next_tokens* 为 0, *pre_tokens* 不小于 max_seqlen_q。
- 当 *sparse_mode* 为 3 时, 每个 batch 的 $S1$ 应小于或等于 $S2$ 。
- *list_seq_k* 中不应存在 0。

- **sparse_mode** (int, 可选) - 指定稀疏模式。默认值为：0。
 - 0: 表示 defaultMask 模式。如果未传入 *attn_mask*，则不执行掩码操作，并忽略 *pre_tokens* 和 *next_tokens* (内部分配为 INT_MAX)。如果传入，需传递完整的 *attn_mask* 矩阵 ($S1 * S2$)，表示需计算的部分在 *pre_tokens* 和 *next_tokens* 之间。
 - 1: 表示 allMask 模式，即传入完整的 *attn_mask* 矩阵。
 - 2: 表示 leftUpCausal 模式，适用于以左顶点划分的下三角场景，需优化的 *attn_mask* 矩阵为 ($2048 * 2048$)。
 - 3: 表示 rightDownCausal 模式，适用于以右下顶点划分的下三角场景，需优化的 *attn_mask* 矩阵为 ($2048 * 2048$)。
 - 4: 表示带状场景，即计算 *pre_tokens* 和 *next_tokens* 之间的部分，需优化的 *attn_mask* 矩阵为 ($2048 * 2048$)。
 - 5: 表示前缀场景，即在 rightDownCausal 的基础上，矩阵左侧添加长度为 $S1$ 、宽度为 N 的矩阵。 N 的值由新输入 *prefix* 获得，每个 batch 轴的 N 值不同，目前尚未使能。
 - 6: 表示全局场景，尚未使能。
 - 7: 表示膨胀场景，尚未使能。
 - 8: 表示局部块场景，尚未使能。

返回：

- **attention_out** (Tensor) - 输出 Tensor，其 shape 和 dtype 与 *query* 相同。

异常：

- **TypeError** - *query* 的数据类型不是 float16 或 bfloat16。
- **TypeError** - *query*、*key* 和 *value* 的数据类型不同。
- **TypeError** - *attn_mask* 的数据类型不是 bool 或 uint8。
- **TypeError** - *real_shift* 的数据类型与 *query* 不一致。
- **TypeError** - *scalar_value* 或 *keep_prob* 不是 double 类型。
- **TypeError** - *input_layout* 不是字符串。
- **TypeError** - *num_key_value_heads* 不是整数。
- **TypeError** - *sparse_mode* 不是整数。
- **TypeError** - *real_shift* 不是 Tensor。
- **TypeError** - *drop_mask* 不是 Tensor。
- **TypeError** - *padding_mask* 不是 Tensor。
- **TypeError** - *attn_mask* 不是 Tensor。
- **ValueError** - *input_layout* 是字符串但不是合法值。
- **RuntimeError** - *head_num* 不能被 $N2$ 整除。
- **RuntimeError** - *head_num* 小于等于 0。
- **RuntimeError** - *attn_mask* 的 shape 不合法。
- **RuntimeError** - *sparse_mode* 被指定的值不合法。
- **RuntimeError** - *query*、*key* 和 *value* 的 D 轴不相同。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> import mindspore.common.dtype as mstype
>>> import numpy as np
>>> from mindspore import ops, Tensor
>>> query = Tensor(np.ones([2, 4, 64]), dtype=mstype.float16)
>>> key = Tensor(np.ones([2, 4, 64]), dtype=mstype.float16)
>>> value = Tensor(np.ones([2, 4, 64]), dtype=mstype.float16)
>>> head_num = 4
>>> output = ops.flash_attention_score(query, key, value, head_num)
>>> print(output.shape)
(2, 4, 64)
```

mindspore.ops.rms_norm

`mindspore.ops.rms_norm(x, gamma, epsilon=1e-6)`

RmsNorm(Root Mean Square Layer Normalization) 即均方根标准化。与 LayerNorm 相比，其保留了缩放不变性，而舍弃了平移不变性。其公式如下：

$$y = \frac{x_i}{\sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2 + \epsilon}} \gamma_i$$

警告：这是一个实验性 API，后续可能修改或删除。该 API 目前只支持在 Atlas A2 训练系列产品上使用。

参数：

- **x** (Tensor) - RmsNorm 的输入, 支持的数据类型为: float16、float32、bfloat16。
- **gamma** (Tensor) - 可训练参数 (γ)，支持的数据类型：float16、float32、bfloat16。
- **epsilon** (float, 可选) - 一个取值范围为 (0, 1] 的浮点值，用于避免除零。默认 1e-6。

返回：

- Tensor，归一化后的结果，shape 和数据类型与 *x* 相同。
- Tensor，类型为 float，表示输入数据标准差的倒数，用于反向梯度计算。

异常：

- **TypeError** - *x* 的数据类型不是 float16、float32、bfloat16 中的一种。
- **TypeError** - *gamma* 的数据类型不是 float16、float32、bfloat16 中的一种。
- **TypeError** - *x* 和 *gamma* 的数据类型不一致。
- **ValueError** - *epsilon* 不是一个 0 到 1 之间的 float 值。
- **ValueError** - *gamma* 的秩大于 *x* 的秩。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[1, 2, 3], [1, 2, 3]]), mindspore.float32)
>>> gamma = Tensor(np.ones([3]), mindspore.float32)
>>> y, rststd = ops.rms_norm(x, gamma)
>>> print(y)
[[0.46290997  0.92581993  1.3887299]
 [0.46290997  0.92581993  1.3887299]]
>>> print(rststd)
[[0.46290997]
 [0.46290997]]
```

mindspore.ops.unfold

`mindspore.ops.unfold(input, kernel_size, dilation=1, padding=0, stride=1)`

从一个 batch 的输入 Tensor 中提取滑动局部块。

考虑一个 batch 的输入 Tensor，其 shape 为 $(N, C, *)$ ，其中 N 是 batch 维度， C 是 channel 维度， $*$ 表示任意的空间维度。此操作将展平输入 x 空间维度内每个滑动的 $kernel\_size$ 大小的滑块为输出的 3-D Tensor 中的一列（如，最后一维）。

输出 Tensor 的 shape 为 $(N, C \times \prod(kernel\_size), L)$ ，其中 $C \times \prod(kernel\_size)$ 为每个滑块内值的总数量（一个滑块有 $\prod(kernel\_size)$ 个空间位置，每个位置都包含一个 C 通道的向量），共有 L 个这样的滑块：

$$L = \prod_d \left\lfloor \frac{\text{spatial\_size}[d] + 2 \times \text{pads}[d] - \text{dilations}[d] \times (\text{kernel\_size}[d] - 1) - 1}{\text{strides}[d]} + 1 \right\rfloor,$$

其中， spatial_size 由输入 x 的空间维度（上面的 $*$ ）决定， d 表示遍历所有的空间维度。

因此，在最后一个维度（列维度）上 *output* 包含特定块内的所有值。

dilation、*padding* 和 *stride* 决定了滑块如何被取出。

警告：

- 该算子的输出为三维 Tensor，其 shape 为 $(N, C \times \prod(kernel\_size), L)$ 。
- 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 四维 Tensor，支持的数据类型：float16、float32、float64、complex64 和 complex128。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 滑窗大小。如果是两个 int，则分别为滑窗的高度和宽度；如果是一个 int，则高度和宽度均为这个 int 值。
- **dilation** (Union[int, tuple[int], list[int]])，可选 - 窗口的空洞卷积的扩充率，如果是两个 int，则分别作用于滑窗的高度和宽度；如果是一个 int，则这个值作用于化窗的高度和宽度。默认 1。
- **padding** (Union[int, tuple[int], list[int]])，可选 - 滑窗的隐式零填充量。为单个 int 或者包含一/二个整数的 tuple/list。默认值：0。
 - 如果是一个 int，则 `pad_height = pad_width`。
 - 如果是两个 int，则 `pad_height = padding[0]`, `pad_width = padding[1]`。
- **stride** (Union[int, tuple[int], list[int]])，可选 - 空间维度上滑动的步长，如果是两个 int，则分别为滑窗的高和宽方向上的步长；如果是一个 int，则高和宽方向上的步长均为这个 int 值。默认值：1。

返回：

Tensor，数据类型与 *input* 相同，其 *shape* 如上所述。

异常：

- **TypeError** - *kernel_size*、*stride*、*dilation*、*padding* 的数据类型不是 *int*、*tuple* 或者 *list*。
- **ValueError** - *kernel_size*、*dilation*、*stride* 包含元素的值小于等于 0 或者元素个数大于 2 。
- **ValueError** - *padding* 包含元素的值小于零。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.rand(4, 4, 32, 32), mindspore.float64)
>>> output = ops.unfold(x, kernel_size=3, dilation=1, stride=1)
>>> print(output.shape)
(4, 36, 900)
```

2.3.2 损失函数

接口名	概述	支持平台
<code>mindspore.ops.binary_cross_entropy</code>	计算预测值 <i>logits</i> 和目标值 <i>labels</i> 之间的二值交叉熵（度量两个概率分布间的差异性信息）损失。	Ascend GPU CPU
<code>mindspore.ops.binary_cross_entropy_with_logits</code>	输入经过 <i>sigmoid</i> 激活函数后作为预测值， <i>binary_cross_entropy_with_logits</i> 计算预测值和目标值之间的二值交叉熵损失。	Ascend GPU CPU
<code>mindspore.ops.cosine_embedding_loss</code>	余弦相似度损失函数，用于测量两个 Tensor 之间的相似性。	Ascend GPU CPU
<code>mindspore.ops.cross_entropy</code>	获取预测值和目标值之间的交叉熵损失。	Ascend GPU CPU
<code>mindspore.ops.ctc_loss</code>	计算 CTC (Connectist Temporal Classification) 损失和梯度。	Ascend GPU CPU
<code>mindspore.ops.gaussian_nll_loss</code>	服从高斯分布的负对数似然损失。	Ascend GPU CPU
<code>mindspore.ops.hinge_embedding_loss</code>	Hinge Embedding 损失函数，衡量输入 <i>inputs</i> 和标签 <i>targets</i> （包含 1 或 -1）之间的损失值。	Ascend GPU CPU
<code>mindspore.ops.huber_loss</code>	计算预测值和目标值之间的误差，兼具 <code>mindspore.ops.l1_loss()</code> 和 <code>mindspore.ops.mse_loss()</code> 的优点。	Ascend GPU CPU
<code>mindspore.ops.kl_div</code>	计算输入 <i>logits</i> 和 <i>labels</i> 的 KL 散度。	Ascend GPU CPU
<code>mindspore.ops.l1_loss</code>	用于计算预测值和目标值之间的平均绝对误差。	Ascend GPU CPU
<code>mindspore.ops.margin_ranking_loss</code>	排序损失函数，用于创建一个衡量给定损失的标准。	Ascend GPU CPU
<code>mindspore.ops.mse_loss</code>	计算预测值和标签值之间的均方误差。	Ascend GPU CPU
<code>mindspore.ops.multi_margin_loss</code>	用于优化多分类问题的合页损失。	Ascend GPU CPU

下页继续

表 12 – 续上页

<code>mindspore.ops.multilabel_margin_loss</code>	用于优化多标签分类问题的合页损失。	Ascend GPU
<code>mindspore.ops.multilabel_soft_margin_loss</code>	基于最大熵计算用于多标签优化的损失。	Ascend GPU CPU
<code>mindspore.ops.nll_loss</code>	获取预测值和目标值之间的负对数似然损失。	Ascend GPU CPU
<code>mindspore.ops.smooth_l1_loss</code>	计算平滑 L1 损失, 该 L1 损失函数有稳健性。	Ascend GPU CPU
<code>mindspore.ops.soft_margin_loss</code>	计算 <i>input</i> 和 <i>target</i> 之间的 soft margin loss。	Ascend GPU
<code>mindspore.ops.triplet_margin_loss</code>	三元组损失函数。	GPU

mindspore.ops.binary_cross_entropy

`mindspore.ops.binary_cross_entropy(logits, labels, weight=None, reduction='mean')`

计算预测值 *logits* 和目标值 *labels* 之间的二值交叉熵（度量两个概率分布间的差异性信息）损失。

将 *logits* 设置为 *x*, *labels* 设置为 *y*, 输出设置为 $\ell(x, y)$, 第 *n* 个 batch 二值交叉熵的权重为 w_n 。则,

$$L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

其中, *L* 表示所有 batch_size 的 loss 值, *l* 表示一个 batch_size 的 loss 值, *n* 表示在 1 - *N* 范围内的一个 batch_size。

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

警告:

- *logits* 的值必须要在 0-1 范围内。

参数:

- **logits** (Tensor) - 输入预测值, 其数据类型为 float16 或 float32。
- **labels** (Tensor) - 输入目标值, shape 与 *logits* 相同。数据类型为 float16 或 float32。
- **weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。支持广播, 使其 shape 与 *logits* 的 shape 保持一致。数据类型必须为 float16 或 float32。默认 None。若为 None, 损失函数将不会考虑任何样本的权重, 每个样本在计算损失时被视为具有相同的重要性。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。

返回:

Tensor 或 Scalar, 如果 *reduction* 为 'none', 则为 shape 和数据类型与输入 *logits* 相同的 Tensor。否则, 输出为 Scalar。

异常:

- **TypeError** - 输入 *logits*, *labels*, *weight* 不为 Tensor。
- **TypeError** - 输入 *logits*, *labels*, *weight* 的数据类型既不是 float16 也不是 float32。

- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。
- **ValueError** - *labels* 的 shape 大小与 *logits* 或者 *weight* 不相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> logits = Tensor(np.array([0.2, 0.7, 0.1]), mindspore.float32)
>>> labels = Tensor(np.array([0., 1., 0.]), mindspore.float32)
>>> weight = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = ops.binary_cross_entropy(logits, labels, weight)
>>> print(output)
0.38240486
```

mindspore.ops.binary_cross_entropy_with_logits

mindspore.ops.binary_cross_entropy_with_logits (*input*, *target*, *weight=None*, *pos_weight=None*, *reduction='mean'*)

输入经过 **sigmoid** 激活函数后作为预测值，*binary_cross_entropy_with_logits* 计算预测值和目标值之间的二值交叉熵损失。

将输入 *input* 设置为 *X*，输入 *target* 设置为 *Y*，输入 *weight* 设置为 *W*，输出设置为 *L*。则，

$$p_{ij} = \text{sigmoid}(X_{ij}) = \frac{1}{1+e^{-X_{ij}}}$$

$$L_{ij} = -[Y_{ij}\log(p_{ij}) + (1 - Y_{ij})\log(1 - p_{ij})]$$

i 表示 i^{th} 样例，*j* 表示类别。则，

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

ℓ 表示计算损失的方法。有三种方法：第一种方法是直接提供损失值，第二种方法是计算所有损失的平均值，第三种方法是计算所有损失的总和。

该算子会将输出乘以相应的权重。*weight* 表示一个 **batch** 中的每条数据分配不同的权重，*pos_weight* 为每个类别的正例子添加相应的权重。

此外，它可以通过向正例添加权重来权衡召回率和精度。在多标签分类的情况下，损失可以描述为：

$$p_{ij,c} = \text{sigmoid}(X_{ij,c}) = \frac{1}{1+e^{-X_{ij,c}}}$$

$$L_{ij,c} = -[P_c Y_{ij,c} * \log(p_{ij,c}) + (1 - Y_{ij,c})\log(1 - p_{ij,c})]$$

其中 *c* 是类别数目 (*c*>1 表示多标签二元分类，*c*=1 表示单标签二元分类)，*n* 是批次中样本的数量，*P_c* 是第 *c* 类正例的权重。*P_c* > 1 增大召回率，*P_c* < 1 增大精度。

参数：

- **input** (Tensor) - 输入预测值。其数据类型为 float16、float32 或 bfloat16（仅 Atlas A2 训练系列产品支持）。

- **target** (Tensor) - 输入目标值, `shape` 与 `input` 相同。数据类型为 `float16`、`float32` 或 `bfloat16` (仅 Atlas A2 训练系列产品支持)。
- **weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。支持广播, 使其 `shape` 与 `input` 的 `shape` 保持一致。数据类型必须为 `float16`、`float32` 或 `bfloat16` (仅 Atlas A2 训练系列产品支持)。默认 `None`, 此时 `weight` 等价于值为 1 的 Tensor。
- **pos_weight** (Tensor, 可选) - 指定正类的权重。是一个长度等于分类数的向量。支持广播, 使其 `shape` 与 `input` 的 `shape` 保持一致。数据类型必须为 `float16`、`float32` 或 `bfloat16` (仅 Atlas A2 训练系列产品支持)。默认 `None`, 此时 `pos_weight` 等价于值为 1 的 Tensor。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 `'none'`、`'mean'`、`'sum'`, 默认 `'mean'`。
 - `'none'`: 不应用规约方法。
 - `'mean'`: 计算输出元素的加权平均值。
 - `'sum'`: 计算输出元素的总和。

返回:

Tensor 或 Scalar, 如果 `reduction` 为 `'none'`, 则为 `shape` 和数据类型与输入 `input` 相同的 Tensor。否则, 输出为 Scalar。

异常:

- **TypeError** - 输入 `input`, `target`, `weight`, `pos_weight` 不为 Tensor。
- **TypeError** - 输入 `input`, `target`, `weight`, `pos_weight` 的数据类型不是 `float16`, `float32` 或 `bfloat16`。
- **TypeError** - `reduction` 输入数据类型不为 `string`。
- **ValueError** - `weight` 或 `pos_weight` 不能广播到 `shape` 为 `input` 的 Tensor。
- **ValueError** - `reduction` 不为 `'none'`、`'mean'` 或 `'sum'`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([[[-0.8, 1.2, 0.7], [-0.1, -0.4, 0.7]]], mindspore.float32)
>>> target = Tensor(np.array([[0.3, 0.8, 1.2], [-0.6, 0.1, 2.2]]], mindspore.float32)
>>> weight = Tensor(np.array([1.0, 1.0, 1.0]), mindspore.float32)
>>> pos_weight = Tensor(np.array([1.0, 1.0, 1.0]), mindspore.float32)
>>> output = ops.binary_cross_entropy_with_logits(input, target, weight, pos_weight)
>>> print(output)
0.3463612
```

mindspore.ops.cosine_embedding_loss

`mindspore.ops.cosine_embedding_loss(input1, input2, target, margin=0.0, reduction='mean')`

余弦相似度损失函数，用于测量两个 Tensor 之间的相似性。

给定两个 Tensor *input1* 和 *input2*，以及一个 Tensor 标签 *target*，值为 1 或 -1，公式如下：

$$\text{loss}(\text{input1}, \text{input2}, \text{target}) = \begin{cases} 1 - \cos(\text{input1}, \text{input2}), & \text{if } \text{target} = 1 \\ \max(0, \cos(\text{input1}, \text{input2}) - \text{margin}), & \text{if } \text{target} = -1 \end{cases}$$

参数：

- **input1** (Tensor) - 输入 Tensor，shape $(N, *)$ ，其中 $*$ 代表任意数量的附加维度。
- **input2** (Tensor) - 输入 Tensor，shape $(N, *)$ 。shape 和数据类型与 *input1* 相同。
- **target** (Tensor) - 输入值为 1 或 -1。假设 *input1* 的 shape 是 $(x_1, x_2, x_3, \dots, x_R)$ ，那么 *target* 的 shape 必须是 $(x_1, x_3, x_4, \dots, x_R)$ 。
- **margin** (float, 可选) - 指定运算的调节因子，取值范围 $[-1.0, 1.0]$ 。默认 0.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

返回：

Tensor 或 Scalar，如果 *reduction* 为 "none"，其 shape 与 *target* 相同。否则，将返回为 Scalar。

异常：

- **TypeError** - *margin* 不是 float。
- **ValueError** - *reduction* 不为 "none"、"mean" 或 "sum"。
- **ValueError** - *margin* 的值不在范围 $[-1.0, 1.0]$ 内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input1 = Tensor(np.array([[0.3, 0.8], [0.4, 0.3]]), mindspore.float32)
>>> input2 = Tensor(np.array([[0.4, 1.2], [-0.4, -0.9]]), mindspore.float32)
>>> target = Tensor(np.array([1, -1]), mindspore.int32)
>>> output = ops.cosine_embedding_loss(input1, input2, target)
>>> print(output)
0.0003425479
```

mindspore.ops.cross_entropy

`mindspore.ops.cross_entropy` (*input*, *target*, *weight=None*, *ignore_index=-100*, *reduction='mean'*, *label_smoothing=0.0*)
获取预测值和目标值之间的交叉熵损失。

`cross_entropy` 方法支持两种不同的目标值 (*target*):

- 类别索引 (int), 取值范围为 $[0, C)$, 其中 C 为类别数, 当 `reduction` 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n, y_n})}{\sum_{c=1}^C \exp(x_{n, c})} \cdot \mathbb{1}\{y_n \neq \text{ignore\_index}\}$$

其中, x 表示预测值, y 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 `reduction` 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{y_n} \cdot \mathbb{1}\{y_n \neq \text{ignore\_index}\}} l_n}{\sum_{n=1}^N l_n}, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

- 类别概率 (float), 用于目标值为多个类别标签的情况。当 `reduction` 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = - \sum_{c=1}^C w_c \log \frac{\exp(x_{n, c})}{\sum_{i=1}^C \exp(x_{n, i})} y_{n, c}$$

其中, x 表示预测值, y 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 `reduction` 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N l_n}{N}, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

参数:

- input** (Tensor) - 输入预测值, shape 为 (N) 或 (N, C) 或 (N, C, H, W) (针对二维数据), 或 $(N, C, d_1, d_2, \dots, d_K)$ (针对高维数据)。input 需为对数概率。数据类型支持 float16 或 float32。
- target** (Tensor) - 输入目标值。若目标值为类别索引, 则 shape 为 $()$ 、 (N) 或 $(N, d_1, d_2, \dots, d_K)$, 数据类型仅支持 int32。若目标值为类别概率, 则 shape 为 $(C,)$ 、 (N, C) 或 $(N, C, d_1, d_2, \dots, d_K)$, 数据类型仅支持 float32 或 float16。
- weight** (Tensor) - 指定各类别的权重。若值不为 None, 则 shape 为 $(C,)$ 。数据类型仅支持 float32 或 float16。默认 None。
- ignore_index** (int) - 指定 target 中需要忽略的值 (一般为填充值), 使其不对梯度产生影响。默认 -100。
- reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。
- label_smoothing** (float) - 标签平滑值, 用于计算 Loss 时防止模型过拟合的正则化手段。取值范围为 $[0.0, 1.0]$ 。默认 0.0。

返回:

Tensor, 数据类型与 *input* 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> # Case 1: Indices labels
>>> inputs = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> target = ms.Tensor(np.array([1, 0, 4]), ms.int32)
>>> output = ms.ops.cross_entropy(inputs, target)
>>> # Case 2: Probability labels
>>> inputs = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> target = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> output = ms.ops.cross_entropy(inputs, target)
```

mindspore.ops.ctc_loss

`mindspore.ops.ctc_loss(log_probs, targets, input_lengths, target_lengths, blank=0, reduction='mean', zero_infinity=False)`

计算 CTC (Connectist Temporal Classification) 损失和梯度。

CTC 是序列标注问题中的一种损失函数, 主要用于处理序列标注问题中的输入与输出标签的对齐问题。传统序列标注算法需要每一时刻输入与输出符号完全对齐, 而 CTC 拓展了标签集合, 添加了空元素。在使用拓展标签集合对序列进行标注后, 所有可以通过映射函数转换为真实序列的预测序列, 都是正确的预测结果, 也就是说无需数据对齐处理, 即可得到预测序列。其目标函数就是最大化所有正确的预测序列的概率和。

关于 CTCLoss 算法详细介绍, 请参考 [Connectionist Temporal Classification: Labeling Unsegmented Sequence Data with Recurrent Neural Networks](#)。

参数:

- **log_probs** (Tensor) - 输入 Tensor, shape (T, N, C) 。其中 T 表示输入长度, N 表示批次大小, C 是分类数, 包含空白。
- **targets** (Tensor) - 目标 Tensor, shape (N, S) 。其中 S 表示最大目标长度。
- **input_lengths** (Union(tuple, Tensor)) - 输入长度, shape 为 (N) 的 Tensor 或 tuple。
- **target_lengths** (Union(tuple, Tensor)) - 目标长度, shape 为 (N) 的 Tensor 或 tuple。
- **blank** (int, 可选) - 空白标签。默认 0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。
 - 'sum': 计算输出元素的总和。
- **zero_infinity** (bool, 可选) - 是否设置无限损失和相关梯度为零。默认 False。

返回:

- **neg_log_likelihood** (Tensor) - 对每一个输入节点可微调的损失值, shape 是 (N) 。
- **log_alpha** (Tensor) - shape 为 $(N, T, 2 * S + 1)$ 的输入到输出的轨迹概率。

异常:

- **TypeError** - *zero_infinity* 不是布尔值, *reduction* 不是字符串。
- **TypeError** - *log_probs* 的数据类型不是 `float` 或 `double`。
- **TypeError** - *targets*、*input_lengths* 或 *target_lengths* 数据类型不是 `int32` 或 `int64`。
- **ValueError** - *log_probs* 的秩不是 3。
- **ValueError** - *targets* 的秩不是 2。
- **ValueError** - *input_lengths* 的 `shape` 大小不等于 `N`。`N` 是 *log_probs* 的批次大小。
- **ValueError** - *target_lengths* 的 `shape` 大小不等于 `N`。`N` 是 *log_probs* 的批次大小。
- **ValueError** - *blank* 值不介于 0 到 `C` 之间。`C` 是 *log_probs* 的分类数。
- **RuntimeError** - *input_lengths* 的值大于 `T`。`T` 是 *log_probs* 的长度。
- **RuntimeError** - *target_lengths[i]* 的取值范围不在 0 到 *input_length[i]* 之间。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> log_probs = Tensor(np.array([[[[0.3, 0.6, 0.6]],
...                               [[0.9, 0.4, 0.2]]]).astype(np.float32))
>>> targets = Tensor(np.array([[0, 1]]), mstype.int32)
>>> input_lengths = Tensor(np.array([2]), mstype.int32)
>>> target_lengths = Tensor(np.array([1]), mstype.int32)
>>> loss, log_alpha = ops.ctc_loss(log_probs, targets, input_lengths,
...                               target_lengths, 0, 'mean', True)
>>> print(loss)
-2.2986124
>>> print(log_alpha)
[[[0.3      0.3      -inf      -inf      -inf]
  [1.2      1.8931472 1.2      -inf      -inf]]]
```

mindspore.ops.gaussian_nll_loss

`mindspore.ops.gaussian_nll_loss(x, target, var, full=False, eps=1e-6, reduction='mean')`

服从高斯分布的负对数似然损失。

目标值被认为是高斯分布的采样, 其中期望和方差通过神经网络来预测。对于以高斯分布为模型的 `Tensor x` 和记录期望的 `Tensor target`, 以及均为正数的方差 `Tensor var` 来说, 计算的 `loss` 为:

$$\text{loss} = \frac{1}{2} \left(\log(\max(\text{var}, \text{eps})) + \frac{(\text{x} - \text{target})^2}{\max(\text{var}, \text{eps})} \right) + \text{const.}$$

其中, `eps` 用于 `log` 的稳定性。当 `full = True` 时, 一个常数会被添加到 `loss` 中。如果 `var` 和 `logits` 的 `shape` 不一致 (出于同方差性的假设), 那么它们必须能够正确地广播。

参数:

- **x** (`Tensor`) - `shape` 为 $(N, *)$ 或 $(*)$ 。`*` 代表着任意数量的额外维度。
- **target** (`Tensor`) - `shape` 为 $(N, *)$ 或 $(*)$ 。和 `x` 具有相同 `shape`, 或者相同 `shape` 但有一个维度为 1 (以允许广播)。

- **var** (Tensor) - shape 为 $(N, *)$ 或 $(*)$ 。和 x 具有相同 shape，或者相同 shape 但有一个维度为 1，或者少一个维度（以允许广播）。
- **full** (bool, 可选) - 指定损失函数中的常数部分。如果 $full = True$ ，则常数为 $const = 0.5 * \log(2\pi)$ 。默认 `False`。
- **eps** (float, 可选) - 用于提高 $\log$ 的稳定性，必须大于 0。默认 $1e-6$ 。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 `'none'`、`'mean'`、`'sum'`，默认 `'mean'`。
 - `"none"`: 不应用规约方法。
 - `"mean"`: 计算输出元素的平均值。
 - `"sum"`: 计算输出元素的总和。

返回：

Tensor 或 Tensor scalar，根据 *reduction* 计算的 loss。

异常：

- **TypeError** - x ， $target$ 或者 var 不是 Tensor。
- **TypeError** - $full$ 不是 bool。
- **TypeError** - eps 不是 float。
- **ValueError** - eps 不是在 $(0, \text{inf})$ 区间的 float。
- **ValueError** - $reduction$ 不是 `'none'`、`'mean'` 或者 `'sum'`。

参考：

Nix, D. A. and Weigend, A. S., "Estimating the mean and variance of the target probability distribution", Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94), Orlando, FL, USA, 1994, pp. 55-60 vol.1, doi: 10.1109/ICNN.1994.374138.

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> import mindspore.common.dtype as mstype
>>> arr1 = np.arange(8).reshape((4, 2))
>>> arr2 = np.array([2, 3, 1, 4, 6, 4, 4, 9]).reshape((4, 2))
>>> x = Tensor(arr1, mstype.float32)
>>> var = Tensor(np.ones((4, 1)), mstype.float32)
>>> target = Tensor(arr2, mstype.float32)
>>> output = ops.gaussian_nll_loss(x, target, var)
>>> print(output)
1.4374993
```

mindspore.ops.hinge_embedding_loss

`mindspore.ops.hinge_embedding_loss(inputs, targets, margin=1.0, reduction='mean')`

Hinge Embedding 损失函数，衡量输入 *inputs* 和标签 *targets*（包含 1 或-1）之间的损失值。

mini-batch 中的第 n 个样例的损失函数为：

$$l_n = \begin{cases} x_n, & \text{if } y_n = 1, \\ \max\{0, \Delta - x_n\}, & \text{if } y_n = -1, \end{cases}$$

总损失值为：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction} = \text{'mean'}; \\ \text{sum}(L), & \text{if reduction} = \text{'sum'}. \end{cases}$$

其中 $L = \{l_1, \dots, l_N\}^T$ 。

参数：

- **inputs** (Tensor) - 预测值，公式中表示为 x 。
- **targets** (Tensor) - 标签值，公式中表示为 y ，和 *inputs* 具有相同 shape，包含 1 或-1。
- **margin** (float, int) - Hinge Embedding Loss 公式定义的阈值 *margin*。公式中表示为 Δ 。默认 1.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - "none"：不应用规约方法。
 - "mean"：计算输出元素的平均值。
 - "sum"：计算输出元素的总和。

返回：

Tensor 或 Tensor scalar，根据 *reduction* 计算的 loss。

异常：

- **TypeError** - *inputs* 不是 Tensor。
- **TypeError** - *targets* 不是 Tensor。
- **TypeError** - *margin* 不是 float 或者 int。
- **ValueError** - *inputs* 和 *targets* shape 不一致且不能广播。
- **ValueError** - *reduction* 不是 "none"、"mean" 或者 "sum"。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore.common.dtype as mstype
>>> from mindspore import ops
>>> from mindspore import Tensor
>>> arr1 = np.array([0.9, -1.2, 2, 0.8, 3.9, 2, 1, 0, -1]).reshape((3, 3))
>>> arr2 = np.array([1, 1, -1, 1, -1, 1, -1, 1, 1]).reshape((3, 3))
>>> logits = Tensor(arr1, mstype.float32)
>>> labels = Tensor(arr2, mstype.float32)
```

(下页继续)

(续上页)

```
>>> loss = ops.hinge_embedding_loss(logits, labels, margin=1.0, reduction='mean')
>>> print(loss)
0.16666666
```

mindspore.ops.huber_loss

`mindspore.ops.huber_loss(input, target, reduction='mean', delta=1.0)`

计算预测值和目标值之间的误差，兼具 `mindspore.ops.l1_loss()` 和 `mindspore.ops.mse_loss()` 的优点。

假设 x 和 y 为一维 Tensor，长度 N ，`reduction` 参数设置为 'none'，计算 x 和 y 的 loss 而不进行降维操作。公式如下：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T$$

以及

$$l_n = \begin{cases} 0.5 * (x_n - y_n)^2, & \text{if } |x_n - y_n| < \text{delta}; \\ \text{delta} * (|x_n - y_n| - 0.5 * \text{delta}), & \text{otherwise.} \end{cases}$$

其中， N 为 batch size。

如果 `reduction` 是 'mean' 或 'sum'，则：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = "mean"}; \\ \text{sum}(L), & \text{if reduction = "sum"} \end{cases}$$

参数：

- **input** (Tensor) - 输入预测值，任意维度的 Tensor。
- **target** (Tensor) - 目标值，通常情况下与 `input` 的 shape 和 dtype 相同。但是当 `target` 和 x 的 shape 不同时，需要保证他们之间可以互相广播。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。
 - 'sum': 计算输出元素的总和。
- **delta** (Union[int, float]) - 两种损失之间变化的阈值。该值必须大于零。默认 1.0。

返回：

Tensor 或 Scalar，如果 `reduction` 为 `none`，则返回与 `input` 具有相同 shape 和 dtype 的 Tensor。否则，将返回 Scalar。

异常：

- **TypeError** - `input` 或 `target` 不是 Tensor。
- **TypeError** - `delta` 不是 float 或 int。
- **ValueError** - `delta` 的值小于或等于 0。
- **ValueError** - `reduction` 不为 "mean"、"sum" 或 "none"。
- **ValueError** - `input` 和 `target` 有不同的 shape，且不能互相广播。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> x = Tensor([1, 2, 10, 2], mindspore.float32)
>>> target = Tensor([1, 5, 1, 20], mindspore.float32)
>>> output = ops.huber_loss(x, target, reduction="mean", delta=2)
>>> print(output)
13.5
```

mindspore.ops.kl_div

`mindspore.ops.kl_div(logits, labels, reduction='mean')`

计算输入 *logits* 和 *labels* 的 KL 散度。

对于相同 shape 的 Tensor *x* 和 *target* , `kl_div` 的计算公式如下:

$$L(x, target) = target \cdot (\log target - x)$$

可得

$$\ell(x, target) = \begin{cases} L(x, target), & \text{if reduction = 'none';} \\ \text{mean}(L(x, target)), & \text{if reduction = 'mean';} \\ \text{sum}(L(x, target))/x.\text{shape}[0], & \text{if reduction = 'batchmean';} \\ \text{sum}(L(x, target)), & \text{if reduction = 'sum'}. \end{cases}$$

其中 *x* 代表 *logits* 。 *target* 代表 *labels* 。 $\ell(x, target)$ 为 *output* 。

说明:

- 目前 Ascend 平台不支持数据类型 float64。
- 仅当 *reduction* 设置为 "batchmean" 时输出才与 Kullback-Leibler 散度的数学定义一致。

参数:

- **logits** (Tensor) - 数据类型支持 float16、float32 或 float64。
- **labels** (Tensor) - 标签 Tensor, 与 *logits* 的 shape 和数据类型相同。
- **reduction** (str) - 指定输出结果的计算方式, 可选值为 'mean'、'batchmean'、'none' 或 'sum'。默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。
 - 'batchmean': 计算输出元素的总和除以 batch 值。

返回:

Tensor 或标量。如果 *reduction* 为 'none' , 则输出为 Tensor 且与 *logits* 的 shape 相同。否则为标量。

异常:

- **TypeError** - *reduction* 不是 str。
- **TypeError** - *logits* 或 *labels* 不是 Tensor。
- **TypeError** - *logits* 或 *labels* 的数据类型不是支持的类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> logits = Tensor(np.array([0.2, 0.7, 0.1]), mindspore.float32)
>>> labels = Tensor(np.array([0., 1., 0.]), mindspore.float32)
>>> output = mindspore.ops.kl_div(logits, labels, 'mean')
>>> print(output)
-0.23333333
```

mindspore.ops.l1_loss

`mindspore.ops.l1_loss(input, target, reduction='mean')`

用于计算预测值和目标值之间的平均绝对误差。

假设 x 和 y 为预测值和目标值，均为一维 Tensor，长度为 N ，*reduction* 设置为 'none'，则计算 x 和 y 的 loss 不进行降维操作。

公式如下：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad \text{with } l_n = |x_n - y_n|,$$

其中， N 为 batch size。

如果 *reduction* 设置为 'mean' 或者 'sum'，则：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = 'mean'}; \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数：

- **input** (Tensor) - 预测值，任意维度的 Tensor。
- **target** (Tensor) - 目标值，通常情况与 *input* 的 shape 相同。如果 *input* 和 *target* 的 shape 不同，需要保证他们之间可以互相广播。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。
 - 'sum': 计算输出元素的总和。

返回：

Tensor 或 Scalar，如果 *reduction* 为 'none'，则返回与 *input* 具有相同 shape 和 dtype 的 Tensor。否则，将返回 Scalar。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *target* 不是 Tensor。
- **ValueError** - *reduction* 不为 'mean'、'sum' 或 'none'。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor([[1, 2, 3], [4, 5, 6]], mstype.float32)
>>> target = Tensor([[6, 5, 4], [3, 2, 1]], mstype.float32)
>>> output = ops.l1_loss(x, target, reduction="mean")
>>> print(output)
3.0
```

mindspore.ops.margin_ranking_loss

`mindspore.ops.margin_ranking_loss(input1, input2, target, margin=0.0, reduction='mean')`

排序损失函数，用于创建一个衡量给定损失的标准。

给定两个 Tensor *input1* 和 *input2*，以及一个 Tensor 标签 *target*，值为 1 或 -1，公式如下：

$$\text{loss}(\text{input1}, \text{input2}, \text{target}) = \max(0, -\text{target} * (\text{input1} - \text{input2}) + \text{margin})$$

参数：

- **input1** (Tensor) - 输入 Tensor，shape (*N*,*)，其中 * 代表任意数量的附加维度。
- **input2** (Tensor) - 输入 Tensor，shape (*N*,*)。shape 和数据类型与 *input1* 相同。
- **target** (Tensor) - 输入值为 1 或 -1。假设 *input1* 的 shape 是 ($x_1, x_2, x_3, \dots, x_R$)，那么 *target* 的 shape 必须是 ($x_1, x_2, x_3, \dots, x_R$)。
- **margin** (float, 可选) - 指定运算的调节因子。默认 0.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 "none"、"mean"、"sum"，默认 "mean"。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

返回：

Tensor 或 Scalar，如果 *reduction* 为 "none"，其 shape 与 *input1* 相同。否则，将返回为 Scalar。

异常：

- **TypeError** - *margin* 不是 float。
- **TypeError** - *input1*，*input2* 和 *target* 不是 Tensor。
- **TypeError** - *input1* 和 *input2* 类型不一致。
- **TypeError** - *input1* 和 *target* 类型不一致。
- **ValueError** - *input1* 和 *input2* shape 不一致。

- **ValueError** - *input1* 和 *target* shape 不一致。
- **ValueError** - *reduction* 不为 "none"、"mean" 或 "sum"。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> input1 = Tensor(np.array([0.3864, -2.4093, -1.4076]), ms.float32)
>>> input2 = Tensor(np.array([-0.6012, -1.6681, 1.2928]), ms.float32)
>>> target = ops.Sign()(Tensor(np.array([-2, -2, 3]), ms.float32))
>>> output = ops.margin_ranking_loss(input1, input2, target)
>>> print(output)
1.2293333
```

mindspore.ops.mse_loss

`mindspore.ops.mse_loss(input, target, reduction='mean')`

计算预测值和标签值之间的均方误差。

更多参考详见[mindspore.nn.MSELoss](#)。

参数：

- **input** (Tensor) - 任意维度的 Tensor。
- **target** (Tensor) - 输入标签，任意维度的 Tensor。大多数场景下与 *input* 具有相同的 shape。但是，也支持在两者 shape 不相同的情况下，通过广播保持一致。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

返回：

Tensor，数据类型为 float，如果 *reduction* 为 'mean' 或 'sum' 时，shape 为 0；如果 *reduction* 为 'none'，输入的 shape 则是广播之后的 shape。

异常：

- **ValueError** - *reduction* 的值不是以下其中之一时：'none'、'mean'、'sum'。
- **ValueError** - *input* 和 *target* 的 shape 不相同且无法广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
```

(下页继续)

(续上页)

```
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
>>> output = ops.mse_loss(logits, labels, reduction='none')
>>> print(output)
[[0. 1. 4.]
 [0. 0. 1.]]
```

mindspore.ops.multi_margin_loss

`mindspore.ops.multi_margin_loss(input, target, p=1, margin=1, weight=None, reduction='mean')`

用于优化多分类问题的合页损失。

优化输入和输出之间的多级分类合页损耗（基于边缘损失）。

对于每个小批量样本，1D 输入 x 和标量输出 y 的损失为：

$$\text{loss}(x, y) = \frac{\sum_i \max(0, \text{margin} - x[y] + x[i])^p}{x.size(0)}$$

其中 $i \in \{0, \dots, x.size(0) - 1\}$ 并且 $i \neq y$ 。

参数：

- **input** (Tensor) - 输入，shape 为 (N, C) 。数据类型只支持 float32、float16 或 float64。即上述公式中的 x 。
- **target** (Tensor) - 真实标签，shape 为 $(N,)$ 。数据类型只支持 int64。值应为非负值，且小于 C 。即上述公式中的 y 。
- **p** (int, 可选) - 对偶距离的范数度。必须为 1 或 2。默认 1。
- **margin** (int, 可选) - 改变对偶距离的参数。默认 1。
- **weight** (Tensor, 可选) - 每个类别的缩放权重，shape 为 $(C,)$ 。数据类型只支持 float32、float16 或 float64。默认 None。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式。可选值为 'none'、'mean'、'sum'。默认 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的加权平均值。
 - "sum": 计算输出元素的总和。

返回：

- **outputs** - 当 *reduction* 为 "none" 时，类型为 Tensor，shape 和 *target* 相同。否则，为标量。

异常：

- **TypeError** - *p* 或者 *target* 数据类型不是 int。
- **TypeError** - *margin* 数据类型不是 int。
- **TypeError** - *reduction* 数据类型不是 str。
- **TypeError** - *input* 数据类型不是以下之一：float16、float、float64。
- **TypeError** - *weight* 和 *input* 的数据类型不相同。
- **ValueError** - *p* 的值不是以下之一：1、2。
- **ValueError** - *reduction* 的值不是以下之一：{"none", "sum", "mean"}。

- **ValueError** - *input* 的 *shape*[0] 和 *target* 的 *shape*[0] 不相等。
- **ValueError** - *input* 的 *shape*[1] 和 *weight* 的 *shape*[0] 不相等。
- **ValueError** - 如果有以下情形：*weight* 的维度不是 1、*target* 的维度不是 1、*input* 的维度不是 2。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> inputs = Tensor(np.ones(shape=[3, 3]), mindspore.float32)
>>> target = Tensor(np.array([1, 2, 1]), mindspore.int64)
>>> weight = Tensor(np.array([1, 1, 1]), mindspore.float32)
>>> output = ops.multi_margin_loss(inputs, target, weight=weight)
>>> print(output)
0.6666667
```

mindspore.ops.multilabel_margin_loss

`mindspore.ops.multilabel_margin_loss(input, target, reduction='mean')`

用于优化多标签分类问题的合页损失。

创建一个标准，用于优化输入 x （一个 2D 小批量 Tensor）和输出 y （一个目标类别索引的 2DTensor）之间的多标签分类合页损失（基于边距的损失）：对于每个小批量样本：

$$\text{loss}(x, y) = \sum_{ij} \frac{\max(0, 1 - (x[y[j]] - x[i]))}{x.size(0)}$$

其中 $x \in \{0, \dots, x.size(0) - 1\}$, $y \in \{0, \dots, y.size(0) - 1\}$, $0 \leq y[j] \leq x.size(0) - 1$, 并且 $i \neq y[j]$ 对于所有 i 和 j 。 y 和 x 的 *shape* 必须相同。该标准仅考虑从前方开始的连续非负目标块。这允许不同的样本具有不同数量的目标类别。

参数：

- **input** (Tensor) - 预测值。上述公式中的 x ，*shape* 为 (C) 或 (N, C) ，其中 N 为批量大小， C 为类别数。数据类型必须为：float16 或 float32。
- **target** (Tensor) - 真实标签，上述公式中的 y ，*shape* 与 *input* 相同，数据类型必须为 int32，标签目标由-1 填充。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。
 - 'sum': 计算输出元素的总和。

返回：

- **outputs** (Union[Tensor, Scalar]) - MultilabelMarginLoss 损失。如果 *reduction* 的值为 'none'，那么返回 *shape* 为 (N) 的 Tensor 类型数据。否则返回一个标量。

异常：

- **TypeError** - 当 *input* 或者 *target* 数据不是 Tensor 时。
- **TypeError** - 当 *input* 数据类型不是以下其中之一时：float16、float32。

- **TypeError** - 当 *target* 数据类型不是 `int32` 时。
- **ValueError** - 当 *input* 的数据维度不是以下其中之一时：1、2。
- **ValueError** - 当 *input* 和 *target* 的 *shape* 不相同。
- **ValueError** - 当 *reduction* 的值不是以下其中之一时：'none'、'mean'、'sum'。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> inputs = Tensor(np.array([[0.1, 0.2, 0.4, 0.8], [0.2, 0.3, 0.5, 0.7]]), mindspore.
    ↳float32)
>>> target = Tensor(np.array([[1, 2, 0, 3], [2, 3, -1, 1]]), mindspore.int32)
>>> output = ops.multilabel_margin_loss(inputs, target)
>>> print(output)
0.325
```

mindspore.ops.multilabel_soft_margin_loss

`mindspore.ops.multilabel_soft_margin_loss(input, target, weight=None, reduction='mean')`

基于最大熵计算用于多标签优化的损失。

多标签软间隔损失通常用于多标签分类任务中，其中输入样本可以属于多个目标类别。给定输入 *input* 和二元标签 *output*，其 *shape* 为 (N, C) ，*N* 表示样本数量，*C* 表示样本类别数，损失计算公式如下：

$$\text{loss}(\text{input}, \text{output}) = -\frac{1}{N} \frac{1}{C} \sum_{i=1}^N \sum_{j=1}^C \left(\text{output}_{ij} \log \frac{1}{1 + e^{-\text{input}_{ij}}} + (1 - \text{output}_{ij}) \log \frac{e^{-\text{input}_{ij}}}{1 + e^{-\text{input}_{ij}}} \right)$$

其中， input_{ij} 表示样本 *i* 在 *j* 类别的概率得分。 output_{ij} 表示样本 *i* 是否属于类别 *j*，当 $\text{output}_{ij} = 1$ 时表示属于，为 0 时表示不属于。对于多标签分类任务，每个样本可以属于多个类别，即标签 *output* 中可以含有多个 1。如果 *weight* 不为 `None`，将会和每个分类的 *loss* 相乘。

参数：

- **input** (Tensor) - *shape* 为 (N, C) 的 Tensor，*N* 为 batch size，*C* 为类别个数。
- **target** (Tensor) - 目标值，数据类型和 *shape* 与 *input* 的相同。
- **weight** (Union[Tensor, int, float]，可选) - 每个类别的缩放权重。默认 `None`。
- **reduction** (str，可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。

返回：

Tensor，数据类型和 *input* 相同。如果 *reduction* 为 'none'，其 *shape* 为 (N) 。否则，其 *shape* 为 0。

支持平台：

Ascend GPU CPU

样例:

```
>>> from mindspore import Tensor, ops
>>> input = Tensor([[0.3, 0.6, 0.6], [0.9, 0.4, 0.2]])
>>> target = Tensor([[0.0, 0.0, 1.0], [0.0, 0.0, 1.0]])
>>> loss = ops.multilabel_soft_margin_loss(input, target, reduction='mean')
>>> print(loss.asnumpy())
0.84693956
```

mindspore.ops.nll_loss

`mindspore.ops.nll_loss(inputs, target, weight=None, ignore_index=-100, reduction='mean', label_smoothing=0.0)`

获取预测值和目标值之间的负对数似然损失。

`reduction` 为 'none' 时, 负对数似然损失公式如下:

$$\ell(x, t) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{t_n} x_{n, t_n}, \quad w_c = \text{weight}[c] \cdot \mathbb{1}\{c \neq \text{ignore\_index}\},$$

其中, x 表示预测值, t 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 `reduction` 不为 'none' (默认为 'mean'), 则

$$\ell(x, t) = \begin{cases} \sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{t_n}} l_n, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'} \end{cases}$$

参数:

- **inputs** (Tensor) - 输入预测值, `shape` 为 (N, C) 或 (N, C, H, W) (针对二维数据), 或 $(N, C, d_1, d_2, \dots, d_K)$ (针对高维数据)。 `inputs` 需为对数概率。数据类型仅支持 float32 或 float16。
- **target** (Tensor) - 输入目标值, `shape` 为 (N) 或 $(N, d_1, d_2, \dots, d_K)$ (针对高维数据)。数据类型仅支持 int32。
- **weight** (Tensor) - 指定各类别的权重。若值不为 None, 则 `shape` 为 $(C,)$ 。数据类型仅支持 float32 或 float16。默认 None。
- **ignore_index** (int) - 指定 `target` 中需要忽略的值 (一般为填充值), 使其不对梯度产生影响。默认 -100。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。
- **label_smoothing** (float) - 标签平滑值, 用于计算 Loss 时防止模型过拟合的正则化手段。取值范围为 $[0.0, 1.0]$ 。默认 0.0。

返回:

Tensor, 数据类型与 `inputs` 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> inputs = mindspore.Tensor(np.random.randn(3, 5), mindspore.float32)
>>> target = mindspore.Tensor(np.array([1, 0, 4]), mindspore.int32)
>>> output = ops.nll_loss(inputs, target)
```

`mindspore.ops.smooth_l1_loss`

`mindspore.ops.smooth_l1_loss` (*input*, *target*, *beta*=1.0, *reduction*='none')

计算平滑 L1 损失, 该 L1 损失函数有稳健性。

平滑 L1 损失是一种类似于 MSELoss 的损失函数, 但对异常值相对不敏感, 可以参阅论文 [Fast R-CNN](#)。

给定长度为 N 的两个输入 x, y , 平滑 L1 损失的计算如下:

$$L_i = \begin{cases} \frac{0.5(x_i - y_i)^2}{\text{beta}}, & \text{if } |x_i - y_i| < \text{beta} \\ |x_i - y_i| - 0.5 * \text{beta}, & \text{otherwise.} \end{cases}$$

当 *reduction* 不是设定为 *none* 时, 计算如下:

$$L = \begin{cases} \text{mean}(L_i), & \text{if reduction = 'mean'}; \\ \text{sum}(L_i), & \text{if reduction = 'sum'}. \end{cases}$$

其中, *beta* 控制损失函数在线性与二次间变换的阈值, $\text{beta} > 0$, 默认值是 1.0。 N 为 batch size。

警告: 该 API 在 CPU 上性能较差, 推荐在 Ascend/GPU 上运行。

参数:

- **input** (Tensor) - shape: ($N, *$), 其中 $*$ 表示任意数量的附加维度。支持数据类型:
 - Ascend: float16、float32、bfloat16。
 - CPU/GPU: float16、float32、float64。
- **target** (Tensor) - shape: ($N, *$)。
 - CPU/Ascend: 与 *input* 的 shape 相同, *input* 和 *target* 遵循隐式类型转换规则, 使数据类型一致。
 - GPU: 与 *input* 的 shape 和数据类型相同。
- **beta** (number, 可选) - 控制损失函数在 L1Loss 和 L2Loss 间变换的阈值, 默认 1.0。
 - Ascend: 该值必须大于等于 0。
 - CPU/GPU: 该值必须大于 0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认 'none'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

返回:

Tensor。如果 *reduction* 为 'none', 则输出为 Tensor 且与 *input* 的 shape 相同。否则 shape 为 ()。

异常：

- **TypeError** - *input* 或 *target* 不是 Tensor。
- **RuntimeError** - *input* 或 *target* 的数据类型不是 float16、float32、float64 和 bfloat16 中的任一者。
- **ValueError** - *input* 与 *target* 的 shape 不同。
- **ValueError** - *reduction* 不是 'none' , 'mean' 和 'sum' 中的任一者。
- **TypeError** - *beta* 不是 float, bool 或 int。
- **RuntimeError** - *beta* 小于等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = ops.smooth_l1_loss(logits, labels)
>>> print(output)
[0.  0.  0.5]
```

mindspore.ops.soft_margin_loss

`mindspore.ops.soft_margin_loss(input, target, reduction='mean')`

计算 *input* 和 *target* 之间的 soft margin loss。

一个二分类任务，计算输入 *x* 和真实标签 *y*（包含 1 或 -1）之间的损失。

$$\text{loss}(x, y) = \sum_i \frac{\log(1 + \exp(-y[i] * x[i]))}{x.\text{nelement}()}$$

其中 *x.nelement()* 是 *x* 的元素数量。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 预测值。数据类型必须为 float16、float32、bfloat16（其中，Atlas 训练系列产品不支持 bfloat16）。
- **target** (Tensor) - 真实标签，shape 与 *input* 相同。GE 模式下类型需要与 *input* 相同。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

返回：

Tensor 或 Scalar。如果 *reduction* 为 'none'，其 shape 与 *input* 相同。否则，将返回 Scalar。

异常:

- **TypeError** - 如果 *input* 或 *target* 不是 Tensor。
- **TypeError** - 如果 *input* 或 *target* 的数据类型不是 float16、float32、bfloat16（其中，Atlas 训练系列产品不支持 bfloat16）。
- **ValueError** - 如果 *input* 与 *target* 的 shape 不相同。
- **ValueError** - 如果 *reduction* 不是 'none'、'mean' 或 'sum'。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> logits = Tensor(np.array([[0.3, 0.7], [0.5, 0.5]]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1], [1, -1]]), mindspore.float32)
>>> output = ops.soft_margin_loss(logits, labels)
>>> print(output)
0.6764238
```

mindspore.ops.triplet_margin_loss

`mindspore.ops.triplet_margin_loss(anchor, positive, negative, margin=1.0, p=2, eps=1e-06, swap=False, reduction='mean')`

三元组损失函数。详情请查看[mindspore.nn.TripletMarginLoss](#)。

参数:

- **anchor** (Tensor) - 从训练集随机选取的样本。数据类型为 BasicType。
- **positive** (Tensor) - 与 *anchor* 为同一类的样本，数据类型与 shape 与 *anchor* 一致。
- **negative** (Tensor) - 与 *anchor* 为异类的样本，数据类型与 shape 与 *anchor* 一致。
- **margin** (float, 可选) - 用于拉近 *anchor* 和 *positive* 之间的距离，拉远 *anchor* 和 *negative* 之间的距离。*margin* 的 shape 必须为 0。默认 1.0。
- **p** (int, 可选) - 成对距离的范数。默认 2。
- **eps** (float, 可选) - 防止除数为 0。默认 1e-06。
- **swap** (bool, 可选) - 距离交换。默认 False。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

返回:

Tensor。如果 *reduction* 为 "none"，其 shape 为 (N)。否则，将返回 Scalar。

异常:

- **TypeError** - *anchor*、*positive* 或者 *negative* 不是 Tensor。
- **TypeError** - *anchor*、*positive* 或者 *negative* 的数据类型不一致。
- **TypeError** - *margin* 的数据类型不是 float。
- **TypeError** - *p* 的数据类型不是 int。
- **TypeError** - *eps* 的数据类型不是 float。
- **TypeError** - *swap* 的数据类型不是 bool。
- **ValueError** - *anchor*、*positive* 和 *negative* 的维度同时小于等于 1。
- **ValueError** - *anchor*、*positive* 或 *negative* 的维度大于等于 8。
- **ValueError** - *anchor*、*positive* 和 *negative* 三者之间的 shape 无法广播。
- **ValueError** - *reduction* 不为“mean”、“sum”或“none”。

支持平台：

GPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> anchor = Tensor(np.array([[0.3, 0.7], [0.5, 0.5]]), mindspore.float32)
>>> positive = Tensor(np.array([[0.4, 0.6], [0.4, 0.6]]), mindspore.float32)
>>> negative = Tensor(np.array([[0.2, 0.9], [0.3, 0.7]]), mindspore.float32)
>>> output = ops.triplet_margin_loss(anchor, positive, negative)
>>> print(output)
0.8881968
```

2.3.3 激活函数

接口名	概述	支持平台
<code>mindspore.ops.celu</code>	<code>celu</code> 激活函数，逐元素计算输入 Tensor 的 <code>celu</code> (Continuously differentiable exponential linear units) 值。	Ascend GPU CPU
<code>mindspore.ops.elu</code>	指数线性单元激活函数。	Ascend GPU CPU
<code>mindspore.ops.fast_gelu</code>	快速高斯误差线性单元激活函数。	Ascend GPU CPU
<code>mindspore.ops.gelu</code>	高斯误差线性单元激活函数。	Ascend GPU CPU
<code>mindspore.ops.glu</code>	门线性单元函数 (Gated Linear Unit function)。	Ascend GPU CPU
<code>mindspore.ops.gumbel_softmax</code>	返回 Gumbel-Softmax 分布的 Tensor。	Ascend GPU CPU
<code>mindspore.ops.hardshrink</code>	Hard Shrink 激活函数。	Ascend GPU CPU
<code>mindspore.ops.hardsigmoid</code>	Hard Sigmoid 激活函数。	Ascend GPU CPU
<code>mindspore.ops.hardswish</code>	Hard Swish 激活函数。	Ascend GPU CPU
<code>mindspore.ops.hardtanh</code>	逐元素计算 <code>hardtanh</code> 激活函数。	Ascend GPU CPU
<code>mindspore.ops.leaky_relu</code>	<code>leaky_relu</code> 激活函数。	Ascend GPU CPU
<code>mindspore.ops.log_softmax</code>	在指定轴上对输入 Tensor 应用 <code>LogSoftmax</code> 函数。	Ascend GPU CPU
<code>mindspore.ops.logsigmoid</code>	按元素计算 <code>LogSigmoid</code> 激活函数。	Ascend GPU CPU

下页继续

表 13 – 续上页

<code>mindspore.ops.mish</code>	逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。	Ascend GPU CPU
<code>mindspore.ops.prelu</code>	带参数的线性修正单元激活函数 (Parametric Rectified Linear Unit activation function)。	Ascend GPU CPU
<code>mindspore.ops.relu</code>	对输入 Tensor 逐元素计算线性修正单元激活函数 (Rectified Linear Unit) 值。	Ascend GPU CPU
<code>mindspore.ops.relu6</code>	计算输入 Tensor 的 ReLU (修正线性单元), 其上限为 6。	Ascend GPU CPU
<code>mindspore.ops.rrelu</code>	Randomized Leaky ReLU 激活函数。	Ascend GPU CPU
<code>mindspore.ops.selu</code>	激活函数 selu (Scaled exponential Linear Unit)。	Ascend GPU CPU
<code>mindspore.ops.sigmoid</code>	逐元素计算 Sigmoid 激活函数。	Ascend GPU CPU
<code>mindspore.ops.silu</code>	按输入逐元素计算激活函数 SiLU (Sigmoid Linear Unit), 也称为 Swish 函数。	Ascend GPU CPU
<code>mindspore.ops.softmax</code>	在指定轴上对输入 Tensor 执行 Softmax 激活函数做归一化操作。	Ascend GPU CPU
<code>mindspore.ops.softmin</code>	在指定轴上对输入 Tensor 执行 Softmin 函数做归一化操作。	Ascend GPU CPU
<code>mindspore.ops.softshrink</code>	逐元素计算 Soft Shrink 激活函数。	Ascend GPU CPU
<code>mindspore.ops.softsign</code>	SoftSign 激活函数。	Ascend GPU CPU
<code>mindspore.ops.swiglu</code>	计算 Swish 门线性单元函数 (Swish Gated Linear Unit function)。	Ascend
<code>mindspore.ops.tanh</code>	逐元素计算输入元素的双曲正切。	Ascend GPU CPU
<code>mindspore.ops.threshold</code>	使用阈值 <i>thr</i> 参数对 <i>input</i> 逐元素阈值化, 并将其结果作为 Tensor 返回。	Ascend GPU CPU

mindspore.ops.celu

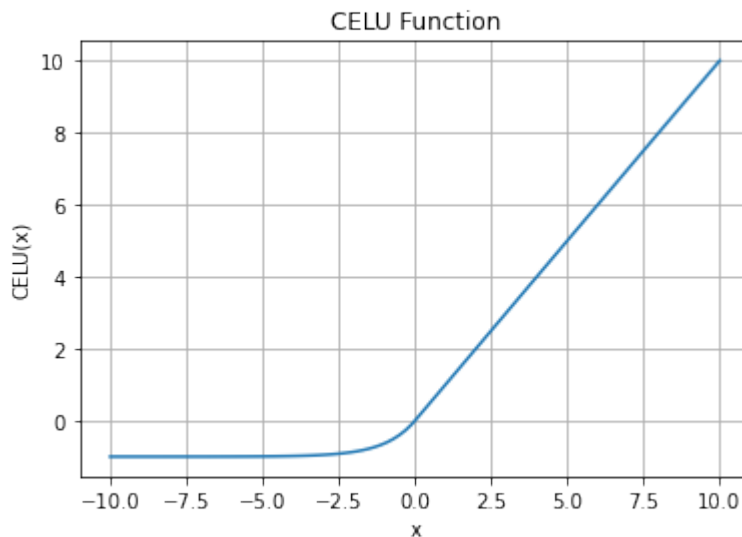
`mindspore.ops.celu(x, alpha=1.0)`
celu 激活函数, 逐元素计算输入 Tensor 的 celu (Continuously differentiable exponential linear units) 值。公式定义如下:

$$\text{CeLU}(x) = \max(0, x) + \min(0, \alpha * (\exp(x/\alpha) - 1))$$

详情请参考: celu。

警告: 这是一个实验性 API, 后续可能修改或删除。

CELU 函数图:



参数：

- **x** (Tensor) - celu 的输入，数据类型为 float16 或 float32。
- **alpha** (float, 可选) - celu 公式定义的阈值 α 。默认 1.0。

返回：

Tensor, shape 和数据类型与输入相同。

异常：

- **TypeError** - *alpha* 不是 float。
- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *x* 的 dtype 既不是 float16 也不是 float32。
- **ValueError** - *alpha* 的值为 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([-2.0, -1.0, 1.0, 2.0]), mindspore.float32)
>>> output = ops.celu(x, alpha=1.0)
>>> print(output)
[-0.86466473 -0.63212055  1.          2.          ]
```

mindspore.ops.elu

`mindspore.ops.elu(input_x, alpha=1.0)`

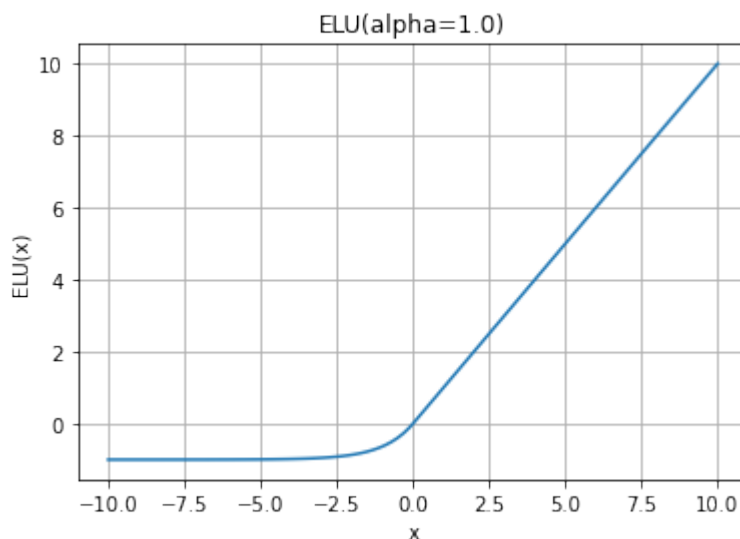
指数线性单元激活函数。

对输入的每个元素计算 ELU。该激活函数定义如下：

$$\text{ELU}(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

其中, x 表示输入 Tensor `input_x`, α 表示 `alpha` 参数, `alpha` 决定 ELU 的平滑度。

ELU 函数图:



参数:

- **input_x** (Tensor) - ELU 的输入, 为任意维度的 Tensor, 数据类型为 float16 或 float32。
- **alpha** (float, 可选) - ELU 的 alpha 值, 数据类型为 float, 目前仅支持 1.0。默认 1.0。

返回:

Tensor, 输出的 shape 和数据类型与 `input_x` 相同。

异常:

- **TypeError** - 如果 `alpha` 不是 float。
- **TypeError** - 如果 `input_x` 的数据类型既不是 float16, 也不是 float32。
- **ValueError** - 如果 `alpha` 不等于 1.0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> output = ops.elu(x)
>>> print(output)
[[-0.63212055  4.          -0.99966455]
 [ 2.          -0.99326205  9.          ]]
```

mindspore.ops.fast_gelu

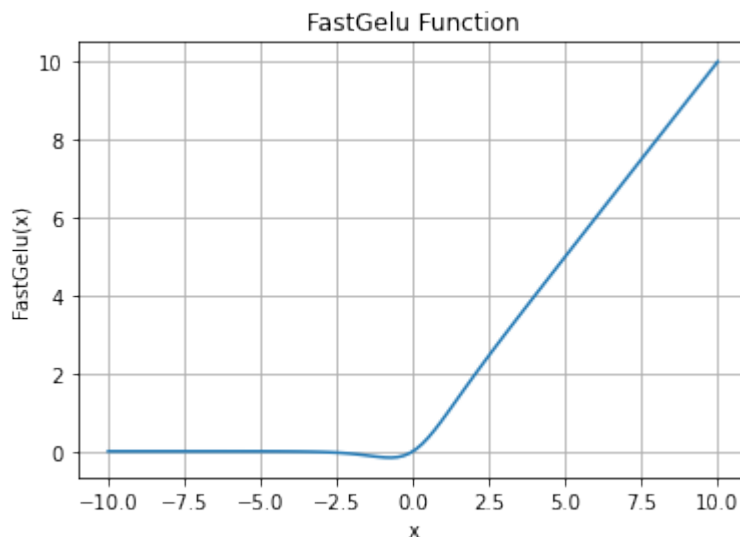
`mindspore.ops.fast_gelu(x)`
快速高斯误差线性单元激活函数。

FastGeLU 定义如下：

$$\text{output} = \frac{x}{1 + \exp(-1.702 * |x|)} * \exp(0.851 * (x - |x|)),$$

其中 x 是输入元素。

FastGelu 函数图：



参数：

- **x (Tensor)** - 计算 FastGeLU 的输入，数据类型为 float16 或者 float32。

返回：

Tensor，其 shape 和数据类型和 x 相同。

异常：

- **TypeError** - x 数据类型不是 float16 或者 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
```

(下页继续)

(续上页)

```
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]], mindspore.float32)
>>> output = ops.fast_gelu(x)
>>> print(output)
[[-1.5418735e-01  3.9921875e+00 -9.7473649e-06]
 [ 1.9375000e+00 -1.0052517e-03  8.9824219e+00]]
```

mindspore.ops.gelu

`mindspore.ops.gelu(input, approximate='none')`

高斯误差线性单元激活函数。

GeLU 的详细描述请参考 [Gaussian Error Linear Units \(GELUs\)](#)。相关应用可参考 [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)。

当 `approximate` 为 `none`，GELU 的定义如下：

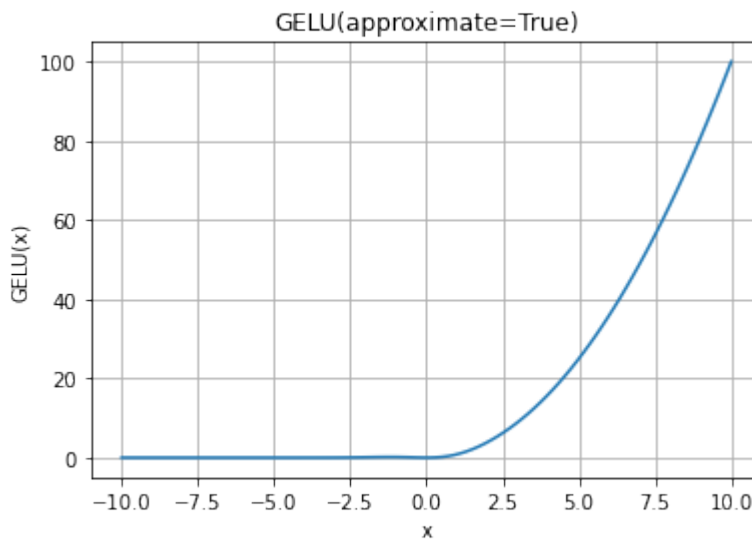
$$GELU(x_i) = x_i * P(X < x_i),$$

其中 P 是标准高斯分布的累积分布函数， x_i 是输入的元素。

当 `approximate` 为 `tanh`，GELU 的定义如下：

$$GELU(x_i) = 0.5 * x_i * (1 + \tanh(\sqrt{2/\pi} * (x_i + 0.044715 * x_i^3)))$$

GELU 函数图：



参数：

- **input** (Tensor) - 用于计算 GELU 的 Tensor。数据类型为 float16、float32 或 float64。
- **approximate** (str, 可选) - GELU 近似算法。可选值为 'none' 和 'tanh'。默认 'none'。

返回：

Tensor，具有与 `input` 相同的数据类型和 shape。

异常：

- **TypeError** - *input* 的数据类型不是 Tensor。
- **TypeError** - *input* 的数据类型不是 bfloat16、float16、float32 或 float64。
- **ValueError** - *approximate* 的值不是 *none* 或 *tanh*。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> x = Tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> result = ops.gelu(x, approximate='none')
>>> print(result)
[0.8413447 1.9544997 2.9959505]
```

mindspore.ops.glu

`mindspore.ops.glu(x, axis=-1)`

门线性单元函数 (Gated Linear Unit function)。

$$GLU(a, b) = a \otimes \sigma(b)$$

其中, a 表示输入 `input_x` 拆分后 Tensor 的前一半元素, b 表示输入拆分 Tensor 的另一半元素。 σ 是 sigmoid 函数, $\otimes$ 是 Hadamard 乘积。请参考 [Language Modeling with Gated Convluational Networks](#)。

参数：

- **x** (Tensor) - 被分 Tensor, 数据类型为 Number, shape 为 $(*_1, N, *_2)$, 其中 $*$ 为任意额外维度。
- **axis** (int, 可选) - 指定分割轴。数据类型为整型, 默认 -1, 输入 x 的最后一维。

返回：

Tensor, 数据类型与输入 x 相同, shape 为 $(*_1, M, *_2)$, 其中 $M = N/2$

异常：

- **TypeError** - x 数据类型不是 Number。
- **TypeError** - x 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> input = Tensor([[0.1, 0.2, 0.3, 0.4], [0.5, 0.6, 0.7, 0.8]])
>>> output = ops.glu(input)
>>> print(output)
[[0.05744425 0.11973753]
 [0.33409387 0.41398472]]
```

mindspore.ops.gumbel_softmax

`mindspore.ops.gumbel_softmax(logits, tau=1.0, hard=False, dim=-1)`

返回 Gumbel-Softmax 分布的 Tensor。当 `hard` 为 `True` 时, 返回 one-hot 形式的离散型 Tensor; 当 `hard` 为 `False` 时, 返回在 `dim` 维度上进行 softmax 运算后的 Tensor。

参数:

- **logits** (Tensor) - 输入, 是一个非标准化的对数概率分布。只支持 `float16` 和 `float32`。
- **tau** (float, 可选) - 标量温度, 正数。默认 `1.0`。
- **hard** (bool, 可选) - 为 `True` 时返回 one-hot 离散型 Tensor, 可反向求导。默认 `False`。
- **dim** (int, 可选) - 给 softmax 使用的参数, 在 `dim` 维上做 softmax 操作。默认 `-1`。

返回:

Tensor, shape 与 dtype 和输入 `logits` 相同。

异常:

- **TypeError** - `logits` 不是 Tensor。
- **TypeError** - `logits` 不是 `float16` 或 `float32`。
- **TypeError** - `tau` 不是 float。
- **TypeError** - `hard` 不是 bool。
- **TypeError** - `dim` 不是 int。
- **ValueError** - `tau` 不是正数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]), mindspore.float32)
>>> output = ops.gumbel_softmax(input_x, 1.0, True, -1)
>>> print(output.shape)
(2, 3)
```

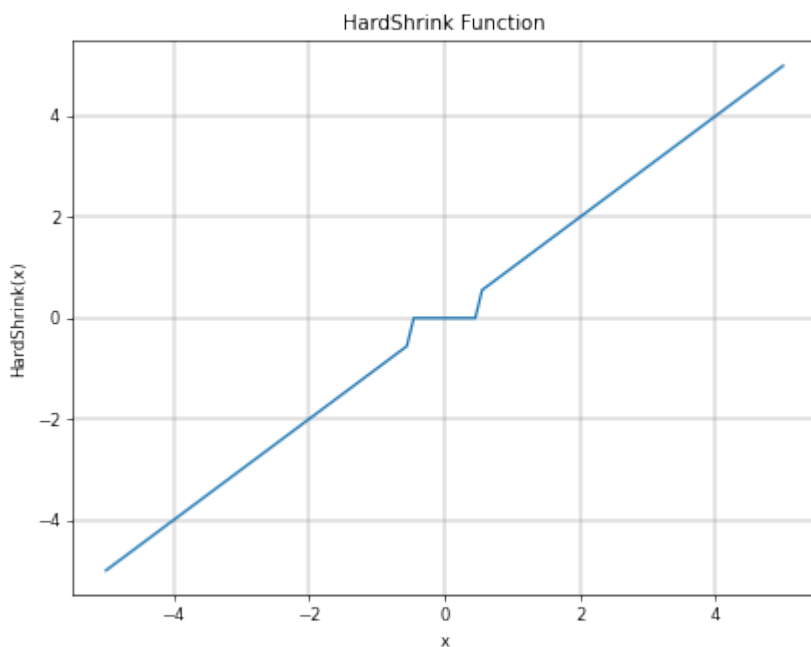
mindspore.ops.hardshrink

`mindspore.ops.hardshrink(input, lambd=0.5)`

Hard Shrink 激活函数。按输入元素计算输出。公式定义如下:

$$\text{HardShrink}(x) = \begin{cases} x, & \text{if } x > \lambda \\ x, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

HardShrink 激活函数图:



参数：

- **input** (Tensor) - Hard Shrink 的输入。支持数据类型：
 - Ascend：float16、float32、bfloat16。
 - CPU/GPU：float16、float32。
- **lambd** (number, 可选) - Hard Shrink 公式定义的阈值 λ 。默认 0.5。

返回：

Tensor, shape 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *lambd* 不是 float、int 或 bool。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的 dtype 不是 float16、float32 或 bfloat16。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([[0.5, 1, 2.0], [0.0533, 0.0776, -2.1233]]), mindspore.float32)
>>> output = ops.hardshrink(input)
>>> print(output)
[[ 0.    1.    2.   ]
 [ 0.    0.   -2.1233]]
```

mindspore.ops.hardsigmoid

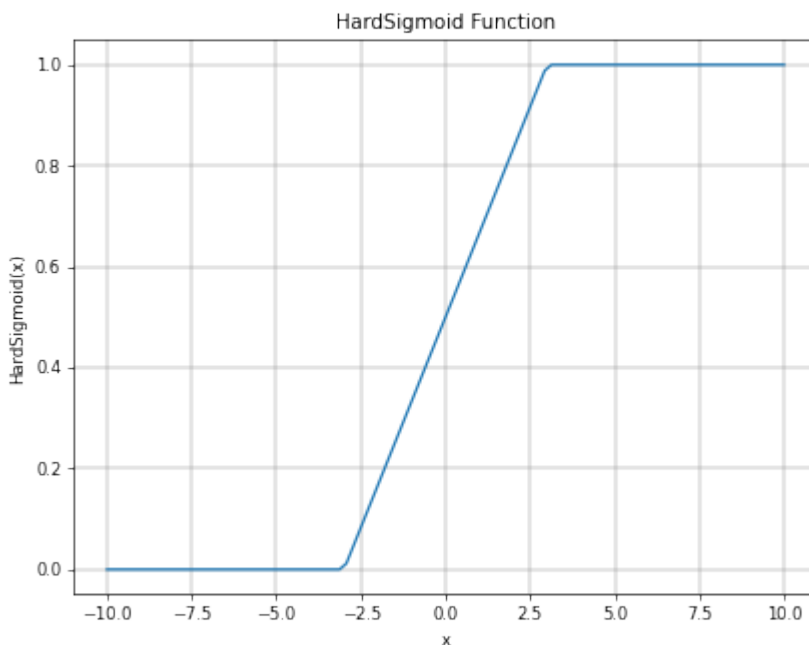
`mindspore.ops.hardsigmoid(input)`

Hard Sigmoid 激活函数。按元素计算输出。

Hard Sigmoid 定义为：

$$\text{HardSigmoid}(input) = \begin{cases} 0, & \text{if } input \leq -3, \\ 1, & \text{if } input \geq +3, \\ input/6 + 1/2, & \text{otherwise} \end{cases}$$

HardSigmoid 函数图：



参数：

- **input** (Tensor) - 输入 Tensor。

返回：

Tensor, shape 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
```

(下页继续)

(续上页)

```
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> output = ops.hardsigmoid(input)
>>> print(output)
[0.3333 0.1666 0.5      0.8335 0.6665]
```

mindspore.ops.hardswish

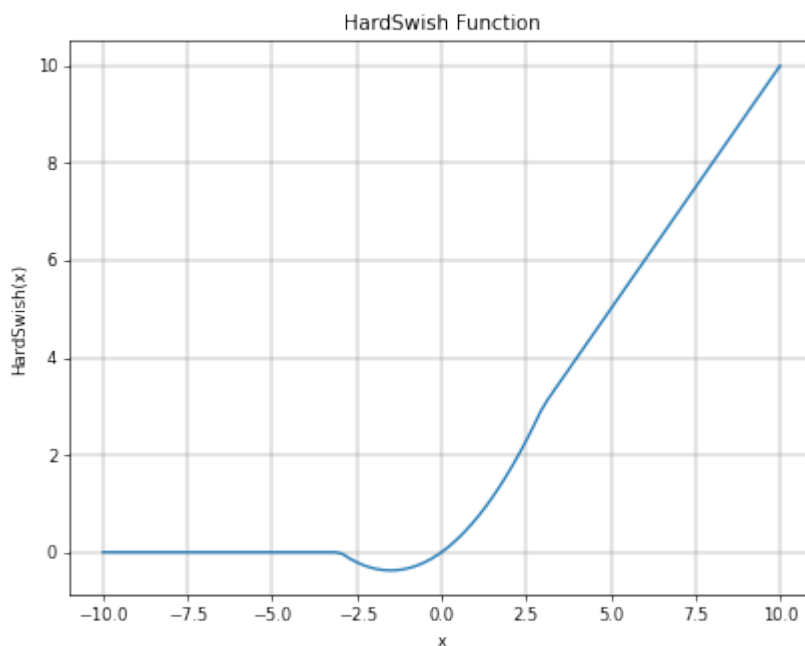
`mindspore.ops.hardswish(input)`

Hard Swish 激活函数。输入是一个 Tensor，具有任何有效的 shape。

Hard Swish 定义如下：

$$\text{Hardswish}(\text{input}) = \begin{cases} 0, & \text{if } \text{input} \leq -3, \\ \text{input}, & \text{if } \text{input} \geq +3, \\ \text{input} * (\text{input} + 3) / 6, & \text{otherwise} \end{cases}$$

HardSwish 函数图：



参数：

- **input** (Tensor) - Hard Swish 的输入。

返回：

Tensor, shape 和数据类型与输入相同。

异常：

- **TypeError** - *input* 不是一个 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> output = ops.hardswish(input)
>>> print(output)
[-0.3333 -0.3333 0 1.667 0.6665]
```

mindspore.ops.hardtanh

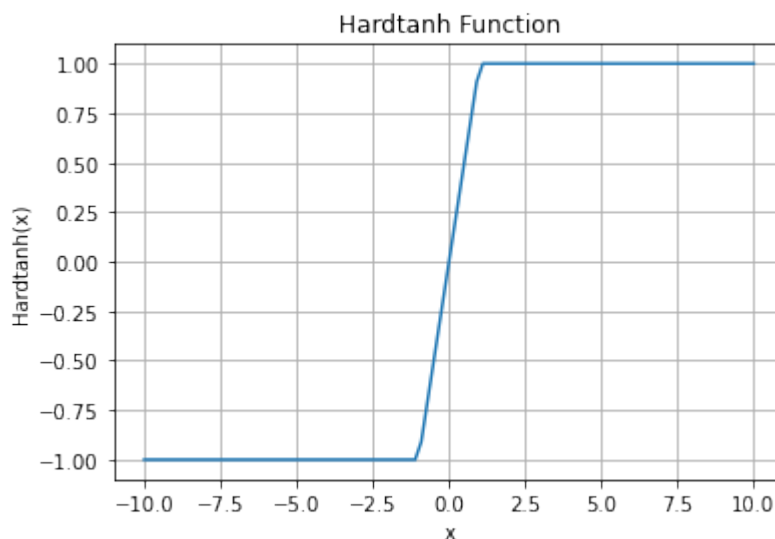
`mindspore.ops.hardtanh(input, min_val=-1.0, max_val=1.0)`

逐元素计算 `hardtanh` 激活函数。`hardtanh` 函数定义为:

$$\text{hardtanh}(\text{input}) = \begin{cases} \text{max\_val}, & \text{if } \text{input} > \text{max\_val} \\ \text{min\_val}, & \text{if } \text{input} < \text{min\_val} \\ \text{input}, & \text{otherwise.} \end{cases}$$

线性区域范围 `[min_val, max_val]` 可以使用 `min_val` 和 `max_val` 进行调整。

Hardtanh 函数图:



参数:

- **input** (Tensor) - 输入 Tensor。
- **min_val** (Union[int, float], 可选) - 线性区域范围的最小值。默认 `-1.0`。
- **max_val** (Union[int, float], 可选) - 线性区域范围的最大值。默认 `1.0`。

返回:

Tensor, 数据类型和 `shape` 与 `input` 的相同。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *min_val* 的数据类型既不是 int 也不是 float。
- **TypeError** - *max_val* 的数据类型既不是 int 也不是 float。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> x = Tensor([-1, -2, 0, 2, 1], mindspore.float16)
>>> output = ops.hardtanh(x, min_val=-1.0, max_val=1.0)
>>> print(output)
[-1. -1.  0.  1.  1.]
```

mindspore.ops.leaky_relu

`mindspore.ops.leaky_relu(input, alpha=0.2)`

`leaky_relu` 激活函数。*input* 中小于 0 的元素乘以 *alpha*。

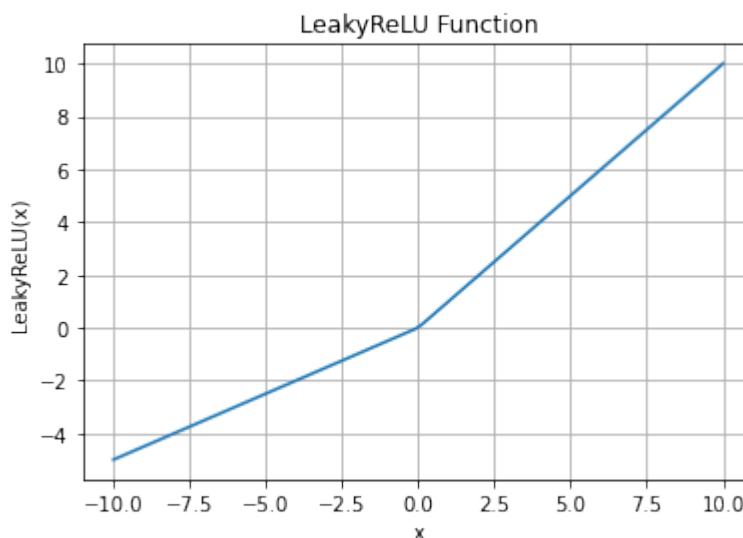
该激活函数定义如下：

$$\text{leaky\_relu}(\text{input}) = \begin{cases} \text{input}, & \text{if } \text{input} \geq 0; \\ \alpha * \text{input}, & \text{otherwise.} \end{cases}$$

其中， α 表示 *alpha* 参数。

更多细节详见 [Rectifier Nonlinearities Improve Neural Network Acoustic Models](#)。

LeakyReLU 函数图：



参数：

- **input** (Tensor) - 计算 `leaky_relu` 的任意维度的 Tensor。
- **alpha** (Union[int, float], 可选) - *input* 的元素小于 0 时激活函数的斜率，默认 0.2。

返回：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *alpha* 不是浮点数或整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]], mindspore.float32)
>>> print(ops.leaky_relu(x, alpha=0.2))
[[-0.2  4.  -1.6]
 [ 2.  -1.   9. ]]
```

mindspore.ops.log_softmax

mindspore.ops.log_softmax (*logits*, *axis=-1*)

在指定轴上对输入 Tensor 应用 LogSoftmax 函数。假设在指定轴上， x 对应每个元素 x_i ，则 LogSoftmax 函数如下所示：

$$\text{output}(x_i) = \log \left(\frac{\exp(x_i)}{\sum_{j=0}^{N-1} \exp(x_j)} \right),$$

其中， N 为 Tensor 长度。

参数：

- **logits** (Tensor) - 输入 Tensor，上述公式中的 x ，shape 为 $(N,*)$ ，其中 $*$ 表示任意数量的附加维度，其数据类型为 float16 或 float32。
- **axis** (int) - 指定进行运算的轴。默认 -1。

返回：

Tensor，数据类型和 shape 与 *logits* 相同。

异常：

- **TypeError** - *axis* 不是 int。
- **TypeError** - *logits* 的数据类型既不是 float16 也不是 float32。
- **ValueError** - *axis* 不在 $[-\text{len}(\text{logits.shape}), \text{len}(\text{logits.shape})$ 范围中。
- **ValueError** - *logits* 的维度小于 1。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> logits = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = ops.log_softmax(logits)
>>> print(output)
[-4.4519143 -3.4519143 -2.4519143 -1.4519144 -0.4519144]
```

mindspore.ops.logsigmoid

`mindspore.ops.logsigmoid(x)`

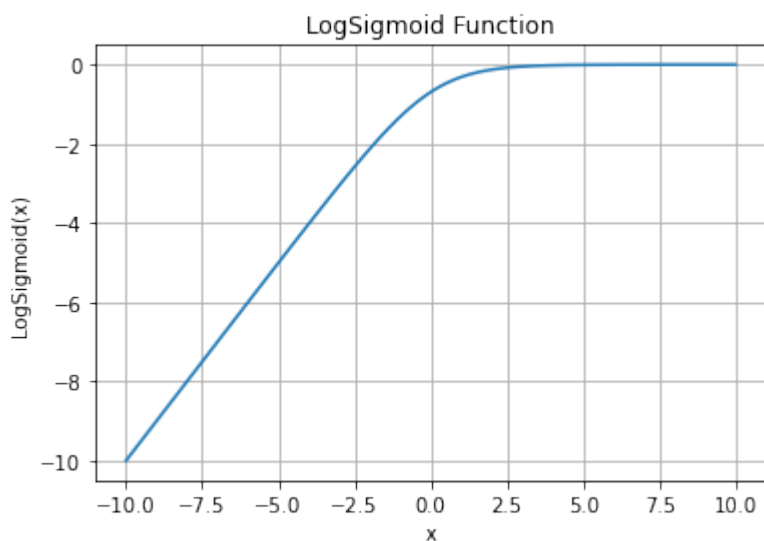
按元素计算 LogSigmoid 激活函数。输入是任意维度的 Tensor。

LogSigmoid 定义为：

$$\text{logsigmoid}(x_i) = \log\left(\frac{1}{1 + \exp(-x_i)}\right),$$

其中, x_i 是输入 Tensor 的一个元素。

LogSigmoid 函数图：



参数：

- **x** (Tensor) - LogSigmoid 的输入，数据类型为 float16 或 float32。shape 为 $(N, *)$ ，其中 * 表示任意的附加维度。

返回：

Tensor，数据类型和 shape 与 x 的相同。

异常：

- **TypeError** - x 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> output = ops.logsigmoid(x)
>>> print(output)
[-0.31326166 -0.12692806 -0.04858734]
```

mindspore.ops.mish

`mindspore.ops.mish(x)`

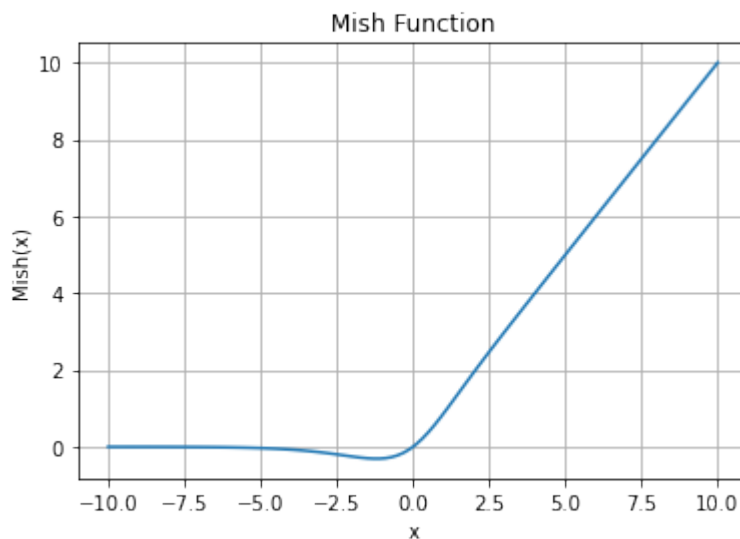
逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。

公式如下:

$$\text{output} = x * \tanh(\log(1 + \exp(x)))$$

更多详细信息请参见 [A Self Regularized Non-Monotonic Neural Activation Function](#)。

Mish 激活函数图:



参数:

- **x** (Tensor) - Mish 的输入。支持数据类型:
 - GPU/CPU: float16、float32、float64。
 - Ascend: float16、float32。

返回:

Tensor, 与 *x* 的 shape 和数据类型相同。

异常:

- **TypeError** - *x* 的数据类型不是 float16、float32 或 float64。

支持平台:

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> output = ops.mish(input_x)
>>> print(output)
[[-3.0340147e-01  3.9974129e+00 -2.68311895e-03]
 [ 1.9439590e+00 -3.3576239e-02  8.99999990e+00]]
>>> input_x = Tensor(2.1, mindspore.float32)
>>> output = ops.mish(input_x)
>>> print(output)
2.050599
```

mindspore.ops.prelu

`mindspore.ops.prelu(input, weight)`

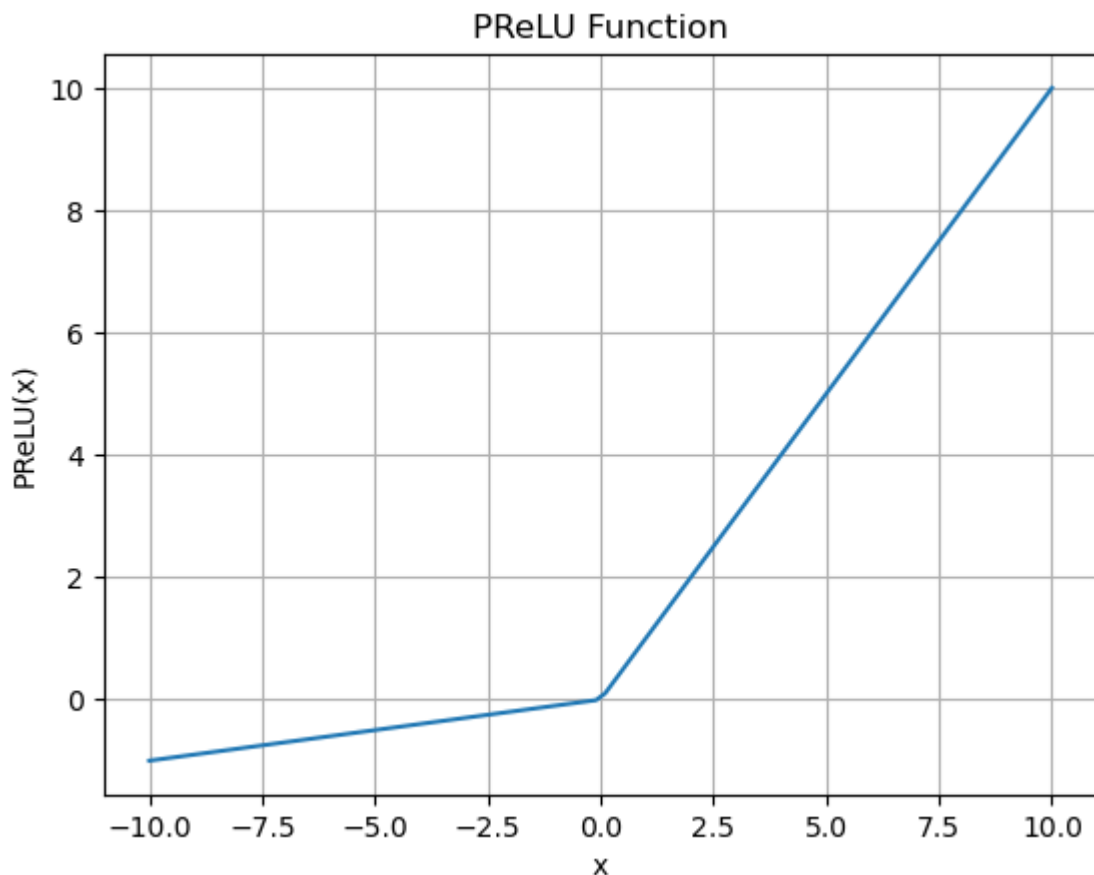
带参数的线性修正单元激活函数（Parametric Rectified Linear Unit activation function）。

[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification](#) 描述了 PReLU 激活函数。定义如下：

$$prelu(x_i) = \max(0, x_i) + \min(0, w * x_i),$$

其中 x_i 是输入的一个通道的一个元素， w 是通道权重。

PReLU 函数图：



说明：通道数是输入的第二个维度值。当输入的维度小于 2 时，则没有通道维度，此时通道数等于 1。

参数：

- **input** (Tensor) - 激活函数的输入 Tensor。
- **weight** (Tensor) - 权重 Tensor。其大小为 1 或者等于输入 Tensor *input* 的通道数。

返回：

Tensor，其 shape 和数据类型与 *input* 相同。有关详细信息，请参考 [mindspore.mint.nn.PReLU](#)。

异常：

- **TypeError** - *input* 或 *weight* 不是 Tensor。
- **ValueError** - *weight* 不是 0-D 或 1-D Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
```

(下页继续)

(续上页)

```
>>> x = Tensor(np.arange(-6, 6).reshape((2, 3, 2)), mindspore.float32)
>>> weight = Tensor(np.array([0.1, 0.6, -0.3]), mindspore.float32)
>>> output = ops.prelu(x, weight)
>>> print(output)
[[[-0.60 -0.50]
  [-2.40 -1.80]
  [ 0.60  0.30]]
 [[ 0.00  1.00]
  [ 2.00  3.00]
  [ 4.00  5.00]]]
```

mindspore.ops.relu

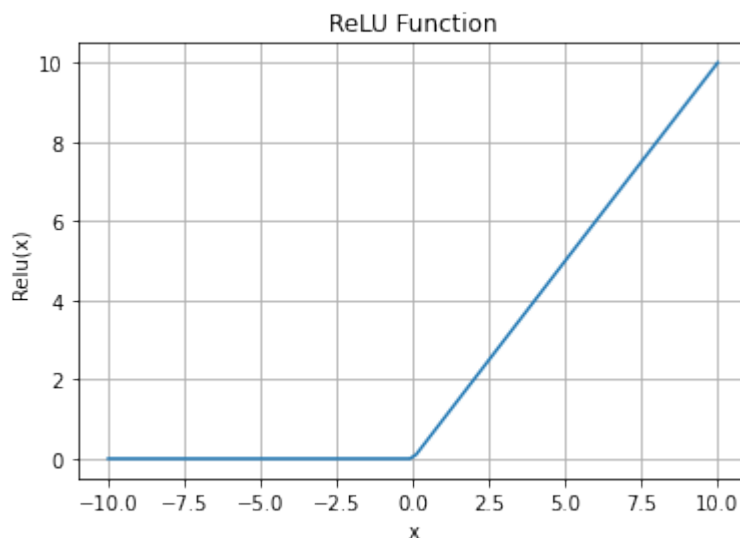
`mindspore.ops.relu(input, inplace=False)`

对输入 **Tensor** 逐元素计算线性修正单元激活函数 (Rectified Linear Unit) 值。

返回 $\max(input, 0)$ 的值。负值神经元将被设置为 0，正值神经元将保持不变。

$$ReLU(input) = (input)^+ = \max(0, input)$$

ReLU 激活函数图：



参数：

- **input** (Tensor) - 输入 Tensor。
- **inplace** (bool, 可选) - 是否采用原地更新模式，默认值为 `False`。

返回：

Tensor，其 `shape` 和数据类型与输入一致。

异常：

- **TypeError** - `input` 的数据类型不是数值型。
- **TypeError** - `input` 不是 Tensor。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> output = ops.relu(input)
>>> print(output)
[[0.  4.  0.]
 [2.  0.  9.]]
```

mindspore.ops.relu6

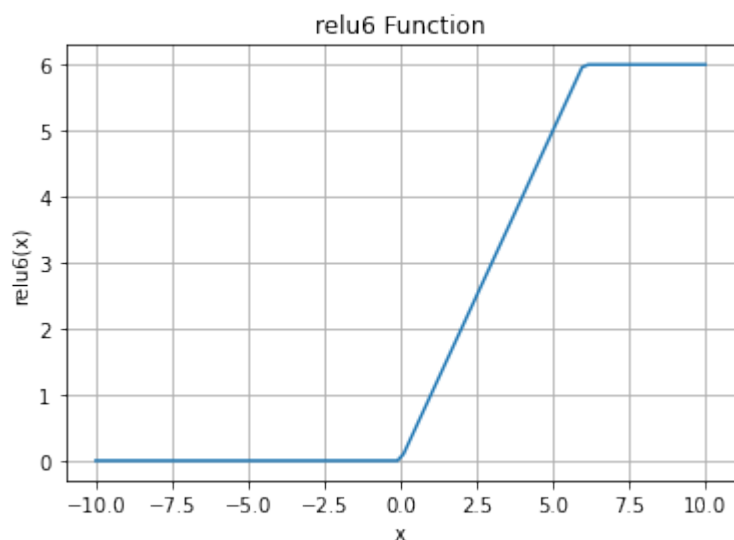
`mindspore.ops.relu6(x)`

计算输入 Tensor 的 ReLU（修正线性单元），其上限为 6。

$$\text{ReLU6}(x) = \min(\max(0, x), 6)$$

返回 $\min(\max(0, x), 6)$ 元素的值。

ReLU6 函数图:



参数:

- **x** (Tensor) - 输入 Tensor。shape 为 $(N, *)$ ，其中 $*$ 表示任意的附加维度数。数据类型必须为 float16、float32。

返回:

Tensor，数据类型和 shape 与 x 相同。

异常:

- **TypeError** - x 的数据类型不是 float16 或 float32。
- **TypeError** - x 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> result = ops.relu6(x)
>>> print(result)
[[0.  4.  0.]
 [2.  0.  6.]]
```

mindspore.ops.rrelu

`mindspore.ops.rrelu(input, lower=1.0/8, upper=1.0/3)`

Randomized Leaky ReLU 激活函数。

该激活函数定义如下:

$$\text{rrelu}(input_{ji}) = \begin{cases} input_{ji}, & \text{if } input_{ji} \geq 0; \\ \alpha_{ji} * input_{ji}, & \text{otherwise.} \end{cases}$$

其中, $\alpha_{ji} \sim U(l, u), l \leq u$ 。

更多细节详见 [Empirical Evaluation of Rectified Activations in Convolution Network](#)。

参数:

- **input** (Tensor) - 计算 RReLU 的任意维度的 Tensor。
- **lower** (Union[int, float]) - *input* 中的数据小于 0 时激活函数的斜率的下界, 默认 1.0 / 8。
- **upper** (Union[int, float]) - *input* 中的数据小于 0 时激活函数的斜率的上界, 默认 1.0 / 3。

返回:

Tensor, 数据类型和 shape 与 *input* 相同。

异常:

- **TypeError** - *lower* 不是浮点数或整数。
- **TypeError** - *upper* 不是浮点数或整数。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 内的数据类型不是 mindspore.float16 或 mindspore.float32。
- **ValueError** - *lower* 大于 *upper*。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[ -1.0,  4.0], [ 2.0,  0.]]), mindspore.float32)
>>> output = ops.rrelu(x)
>>> print(output)
[[-0.31465699  4.          ]
 [ 2.          0.          ]]
```

mindspore.ops.selu

`mindspore.ops.selu(input_x)`

激活函数 `selu` (Scaled exponential Linear Unit)。

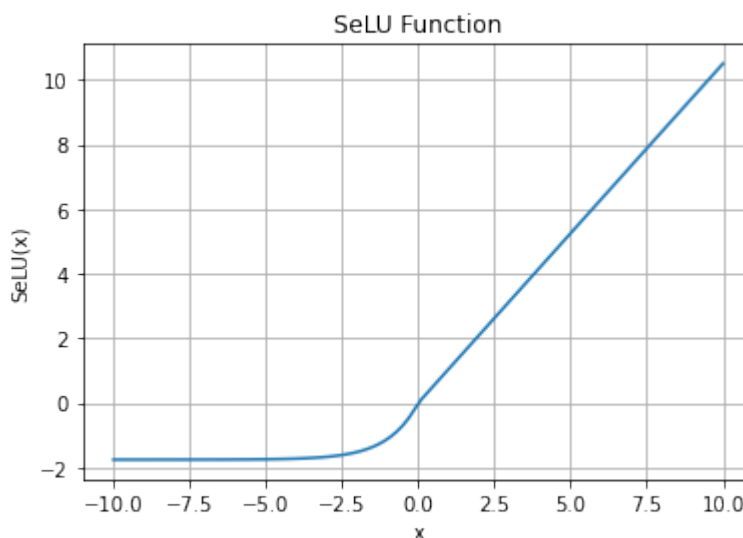
该激活函数定义为：

$$E_i = scale * \begin{cases} x_i, & \text{if } x_i \geq 0; \\ \alpha * (\exp(x_i) - 1), & \text{otherwise.} \end{cases}$$

其中, α 和 $scale$ 是预定义的常量 ($\alpha = 1.67326324$, $scale = 1.05070098$)。

更多详细信息, 请参见 [Self-Normalizing Neural Networks](#)。

SeLU 函数图：



参数：

- **input_x** (Tensor) - 任意维度的 Tensor, 数据类型为 int8、int32、float16、float32、float64 (仅 CPU、GPU)。

返回：

Tensor, 数据类型和 shape 与 `input_x` 的相同。

异常：

- **TypeError** - `input_x` 的数据类型不是 int8、int32、float16、float32、float64。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> output = ops.selu(input_x)
>>> print(output)
[[-1.1113307  4.202804 -1.7575096]
 [ 2.101402 -1.7462534  9.456309  ]]
```

mindspore.ops.sigmoid

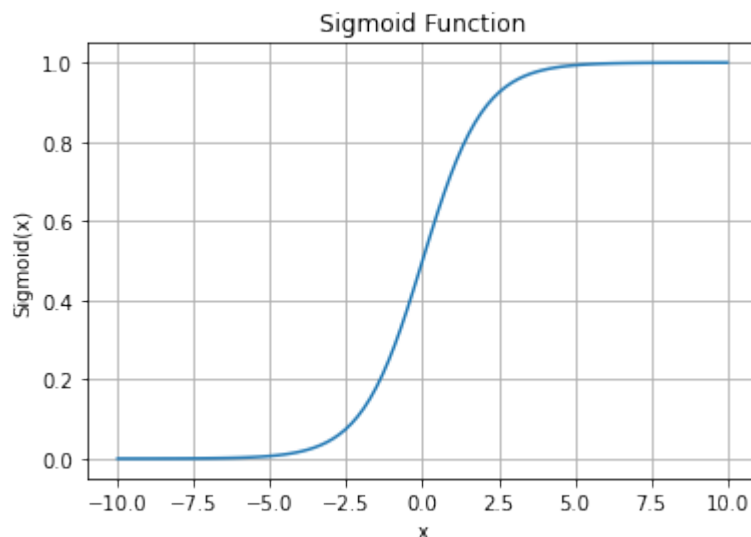
`mindspore.ops.sigmoid(input)`

逐元素计算 Sigmoid 激活函数。Sigmoid 函数定义为：

$$\text{sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)}$$

其中 x_i 是 x 的一个元素。

Sigmoid 函数图：



参数：

- **input** (Tensor) - *input* 即为上述公式中的 x 。任意维度的 Tensor，数据类型为 float16、float32、float64、complex64 或 complex128。

返回：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *input* 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = ops.sigmoid(input)
>>> print(output)
[0.7310586  0.880797  0.95257413 0.98201376 0.9933072 ]
```

mindspore.ops.silu

`mindspore.ops.silu(input)`

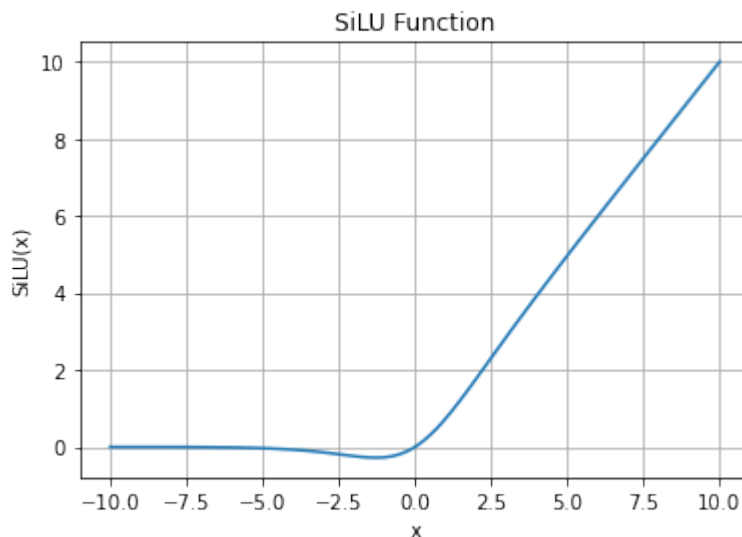
按输入逐元素计算激活函数 SiLU (Sigmoid Linear Unit), 也称为 Swish 函数。该激活函数定义为:

$$\text{SiLU}(x) = x * \sigma(x),$$

其中 x 是输入的元素, $\sigma(x)$ 是 Sigmoid 函数。

$$\text{sigma}(x_i) = \frac{1}{1 + \exp(-x_i)},$$

SiLU 函数图:



参数:

- **input** (Tensor) - *input* 即为上述公式中的 x 。数据类型为 float16 或 float32 的输入。

返回:

Tensor, 数据类型和 shape 与 *input* 的相同。

异常:

- **TypeError** - *input* 的数据类型不是 float16 或 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> input = Tensor(np.array([-1, 2, -3, 2, -1]), mindspore.float16)
>>> output = ops.silu(input)
>>> print(output)
[-0.269  1.762 -0.1423  1.762 -0.269]
```

mindspore.ops.softmax

`mindspore.ops.softmax(input, axis=-1, *, dtype=None)`

在指定轴上对输入 Tensor 执行 Softmax 激活函数做归一化操作。假设指定轴 *axis* 上有切片，那么每个元素 $input_i$ 所对应的 Softmax 函数如下所示：

$$output(input_i) = \frac{\exp(input_i)}{\sum_{j=0}^{N-1} \exp(input_j)},$$

其中 N 代表 Tensor 的长度。

参数：

- **input** (Tensor) - Softmax 的输入，shape 为 $(N, *)$ ，其中 $*$ 为任意额外维度。支持 float16 和 float32。
- **axis** (int, 可选) - 指定 Softmax 操作的轴。默认 -1。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 如果设置此参数，则会在执行之前将 *input* 转换为指定的类型，返回的 Tensor 类型也将为指定类型 *dtype*。默认 None。

返回：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *axis* 不是 int。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = ops.softmax(input)
>>> print(output)
[0.01165623 0.03168492 0.08612854 0.23412167 0.6364086 ]
```

mindspore.ops.softmin

`mindspore.ops.softmin(x, axis=-1, *, dtype=None)`

在指定轴上对输入 Tensor 执行 Softmin 函数做归一化操作。假设指定轴 *x* 上有切片，那么每个元素 x_i 所对应的 Softmin 函数如下所示：

$$output(x_i) = \frac{\exp(-x_i)}{\sum_{j=0}^{N-1} \exp(-x_j)},$$

其中 N 代表 Tensor 的长度。

参数：

- **x** (Tensor) - Softmin 的输入，其数据类型为 float16 或 float32。其 shape 为 $(N, *)$ ，其中 $*$ 为任意数量的额外维度。
- **axis** (Union[int, tuple[int]], 可选) - 指定 Softmin 操作的轴。默认 -1。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 如果设置此参数，则会在执行之前将 x 转换为指定的类型，返回的 Tensor 类型也将为指定类型 *dtype*。默认 `None`。

返回：

Tensor，数据类型和 *shape* 与 x 相同。

异常：

- **TypeError** - *axis* 不是 int 或者 tuple。
- **TypeError** - x 的数据类型既不是 float16 也不是 float32。
- **ValueError** - *axis* 是长度小于 1 的 tuple。
- **ValueError** - *axis* 是一个 tuple，其元素不全在 $[-\text{len}(x.\text{shape}), \text{len}(x.\text{shape})]$ 范围中。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> output = ops.softmin(x)
>>> print(output)
[0.2341  0.636  0.0862  0.01165  0.03168 ]
```

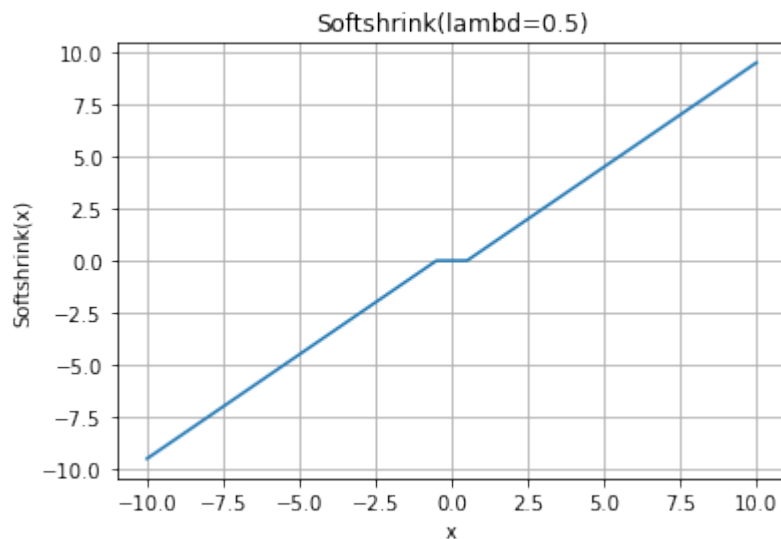
mindspore.ops.softshrink

`mindspore.ops.softshrink(input, lambd=0.5)`

逐元素计算 Soft Shrink 激活函数。

$$\text{SoftShrink}(x) = \begin{cases} x - \lambda, & \text{if } x > \lambda \\ x + \lambda, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

SoftShrink 激活函数图：



参数：

- **input** (Tensor) - Soft Shrink 的输入。支持数据类型：
 - Ascend：float16、float32、bfloat16。
- **lambd** (number, 可选) - Soft Shrink 公式定义的阈值 λ ，必须不小于零。默认 0.5。

返回：

Tensor, shape 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *lambd* 不是 float、bool 或 int。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的 dtype 不是 float16、float32 或 bfloat16。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> import numpy as np
>>> x = Tensor(np.array([[ 0.5297,  0.7871,  1.1754], [ 0.7836,  0.6218, -1.1542]]),
↳mindspore.float32)
>>> output = ops.softshrink(x)
>>> print(output)
[[ 0.02979  0.287   0.676  ]
 [ 0.2837   0.1216 -0.6543  ]]
```

mindspore.ops.softsign

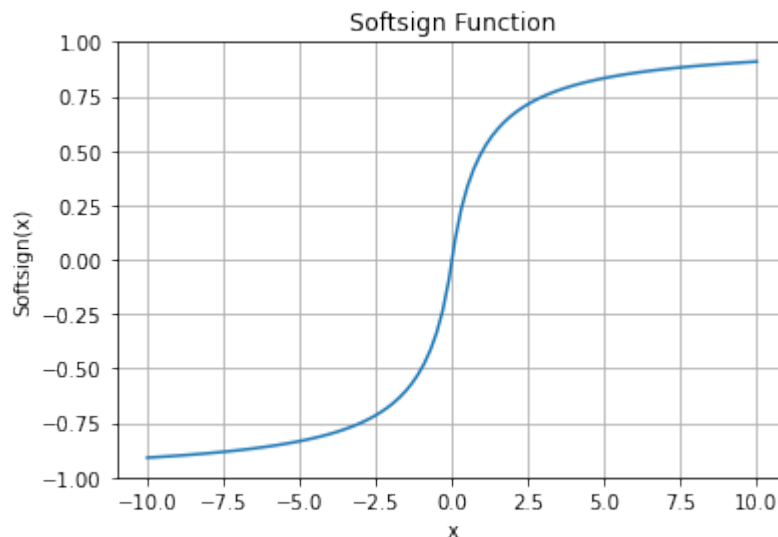
`mindspore.ops.softsign(x)`

SoftSign 激活函数。

SoftSign 函数定义为：

$$\text{SoftSign}(x) = \frac{x}{1 + |x|}$$

Softsign 函数图：



参数：

- **x** (Tensor) - shape 为 $(N,*)$ 的 Tensor，其中 $*$ 表示任意个数的维度。它的数据类型必须为 float16 或 float32。

返回：

Tensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - x 不是 Tensor。
- **TypeError** - x 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([0, -1, 2, 30, -30]), mindspore.float32)
>>> output = ops.softsign(x)
>>> print(output)
[ 0.          -0.5          0.6666667   0.9677419  -0.9677419]
```

mindspore.ops.swiglu

`mindspore.ops.swiglu(input, dim=-1)`

计算 Swish 门线性单元函数 (Swish Gated Linear Unit function)。SwiGLU 是 `mindspore.ops.GLU` 激活函数的变体, 定义为:

警告: 这是一个实验性 API, 后续可能修改或删除。

$$\text{SwiGLU}(a, b) = \text{Swish}(a) \otimes b$$

其中, a 表示输入 `input` 拆分后 Tensor 的前一半元素, b 表示输入拆分 Tensor 的另一半元素, $\text{Swish}(a) = a \sigma(a)$, σ 是 `mindspore.ops.sigmoid()` 函数, $\otimes$ 是 Hadamard 乘积。

参数:

- **input** (Tensor) - 被分 Tensor, shape 为 $(*_1, N, *_2)$, 其中 $*$ 为任意额外维度。 N 必须能被 2 整除。
- **dim** (int, 可选) - 指定分割轴。数据类型为整型, 默认 -1, 输入 `input` 的最后一维。

返回:

Tensor, 数据类型与输入 `input` 相同, shape 为 $(*_1, M, *_2)$, 其中 $M = N/2$ 。

异常:

- **TypeError** - `input` 数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - `input` 不是 Tensor。
- **RuntimeError** - `dim` 指定维度不能被 2 整除。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input = Tensor([[[-0.12, 0.123, 31.122], [2.1223, 4.1212121217, 0.3123]], dtype=mindspore.
↳float32)
>>> output = ops.swiglu(input, 0)
>>> print(output)
[[-0.11970687 0.2690224 9.7194 ]]
```

mindspore.ops.tanh

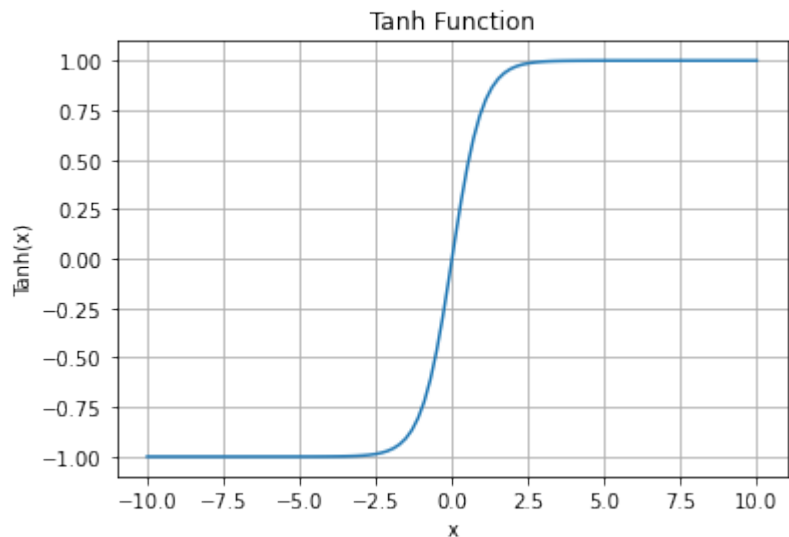
`mindspore.ops.tanh(input)`

逐元素计算输入元素的双曲正切。Tanh 函数定义为:

$$\tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 Tensor 的元素。

Tanh 函数图:



参数:

- **input** (Tensor) - Tanh 的输入。

返回:

Tensor, 数据类型和 shape 与 *input* 相同。

异常:

- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = ops.tanh(input)
>>> print(output)
[0.7615941 0.9640276 0.9950547 0.9993293 0.9999092]
```

mindspore.ops.threshold

`mindspore.ops.threshold(input, thr, value)`

使用阈值 *thr* 参数对 *input* 逐元素阈值化, 并将其结果作为 Tensor 返回。

`threshold` 定义为:

$$y = \begin{cases} input, & \text{if } input > thr \\ value, & \text{otherwise} \end{cases}$$

参数:

- **input** (Tensor) - 输入 Tensor, 数据类型为 float16 或 float32。
- **thr** (Union[int, float]) - 阈值。

- **value** (Union[int, float]) - 输入 Tensor 中 element 小于阈值时的填充值。

返回：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *thr* 不是浮点数或整数。
- **TypeError** - *value* 不是浮点数或整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> inputs = mindspore.Tensor([0.0, 2, 3], mindspore.float32)
>>> outputs = ops.threshold(inputs, 1, 100)
>>> print(outputs)
[100.  2.  3.]
```

2.3.4 距离函数

接口名	概述	支持平台
<code>mindspore.ops.cdist</code>	计算两个 Tensor 每对行向量之间的 p-norm 距离。	Ascend GPU CPU
<code>mindspore.ops.dist</code>	计算输入中每对行向量之间的 <i>p</i> -norm 距离。	Ascend GPU CPU
<code>mindspore.ops.pdist</code>	计算输入中每对行向量之间的 <i>p</i> -范数距离。	GPU CPU

mindspore.ops.cdist

`mindspore.ops.cdist` (*x1*, *x2*, *p*=2.0)
计算两个 Tensor 每对行向量之间的 p-norm 距离。

说明：

- Ascend 上支持的输入数据类型为 float16 和 float32。
- CPU 上支持的输入数据类型为 float16 和 float32。
- GPU 上支持的输入数据类型为 float32 和 float64。

参数：

- **x1** (Tensor) - 输入 Tensor，shape 为 (*B*, *P*, *M*)，*B* 表示 0 或者正整数。*B* 维度为 0 时该维度被忽略，shape 为 (*P*, *M*)。
- **x2** (Tensor) - 输入 Tensor，shape 为 (*B*, *R*, *M*)，与 *x1* 的数据类型一致。
- **p** (float, 可选) - 计算向量对 p-norm 距离的 P 值，P >= 0。默认 2.0。

返回：

Tensor, p -范数距离, 数据类型与 $x1$ 一致, shape 为 (B, P, R) 。

异常:

- **TypeError** - $x1$ 或 $x2$ 不是 Tensor。
- **TypeError** - $x1$ 或 $x2$ 的数据类型不符合上述“说明”中的要求。
- **TypeError** - p 不是 float32。
- **ValueError** - p 是负数。
- **ValueError** - $x1$ 与 $x2$ 维度不同。
- **ValueError** - $x1$ 与 $x2$ 的维度既不是 2, 也不是 3。
- **ValueError** - $x1$ 和 $x2$ 的批次维无法广播。
- **ValueError** - $x1$ 和 $x2$ 的列数不一样。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[1.0, 1.0], [2.0, 2.0]]]).astype(np.float32))
>>> y = Tensor(np.array([[[3.0, 3.0], [3.0, 3.0]]]).astype(np.float32))
>>> output = ops.cdist(x, y, 2.0)
>>> print(output)
[[[2.8284273 2.8284273]
  [1.4142137 1.4142137]]]
```

mindspore.ops.dist

`mindspore.ops.dist` (*input*, *other*, $p=2$)

计算输入中每对行向量之间的 p -norm 距离。

说明: 在 MindSpore 中只支持计算整数 p -norm 形式的范数, 如果 p 不是整数会引发类型错误。

参数:

- **input** (Tensor) - 第一个输入 Tensor, 数据类型需为 float16 或 float32。
- **other** (Tensor) - 第二个输入 Tensor, 数据类型需为 float16 或 float32。
- **p** (int, 可选) - 范数的次数。 p 大于或等于 0。默认 2。

返回:

Tensor, 具有与 *input* 相同的 dtype, 其 shape 为: (1)。

异常:

- **TypeError** - *input* 或 *other* 不是 Tensor。
- **TypeError** - *input* 和 *other* 数据类型不是 float16 或 float32。
- **TypeError** - p 不是非负整数。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import Tensor, ops
>>> input_x = Tensor([[[1.0, 1.0], [2.0, 2.0]]])
>>> input_y = Tensor([[[3.0, 3.0], [3.0, 3.0]]])
>>> out = ops.dist(input_x, input_y)
>>> print(out.asnumpy())
3.1622777
```

mindspore.ops.pdist

`mindspore.ops.pdist(input, p=2.0)`

计算输入中每对行向量之间的 p -范数距离。如果输入 Tensor *input* 的 shape 为 (N, M) , 那么输出就是一个 shape 为 $(N * (N - 1) / 2,)$ 的 Tensor。

$$y[n] = \sqrt[p]{|x_i - x_j|^p}$$

其中 x_i, x_j 是输入中两个不同的行向量。

参数:

- **input** (Tensor) - 输入 Tensor。类型: float16、float32 或 float64。
- **p** (float) - 范数距离的阶, $p[0,)$ 。默认 2.0。

返回:

Tensor, 类型与 *input* 一致。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的数据类型不是 float16、float32、float64。
- **TypeError** - *p* 不是 float。
- **ValueError** - *p* 是负数。
- **ValueError** - *input* 的维度小于 2。

支持平台:

GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[1.0, 1.0], [2.0, 2.0], [3.0, 3.0]]).astype(np.float32))
>>> y = ops.pdist(x, p=2.0)
>>> print(y)
[1.4142135 2.828427 1.4142135]
```

2.3.5 采样函数

接口名	概述	支持平台
<code>mindspore.ops.choice_with_mask</code>	对输入进行随机取样，返回取样索引和掩码。	Ascend GPU CPU
<code>mindspore.ops.random_categorical</code>	从一个分类分布中生成随机样本。	Ascend GPU CPU
<code>mindspore.ops.log_uniform_candidate_sampler</code>	使用 <code>log-uniform(Zipfian)</code> 分布对一组类别进行采样。	Ascend CPU
<code>mindspore.ops.uniform_candidate_sampler</code>	使用均匀分布对一组类别进行采样。	Ascend GPU CPU

mindspore.ops.choice_with_mask

`mindspore.ops.choice_with_mask(input_x, count=256, seed=None)`

对输入进行随机取样，返回取样索引和掩码。

输入必须是维度不小于 1 的 `Tensor`。如果其维度大于等于 2，则第一个维度指定样本数。返回的索引 `Tensor` 为非空样本值的索引，掩码 `Tensor` 说明索引 `Tensor` 中的哪些元素是有效的。

警告：Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **input_x** (`Tensor[bool]`) - 输入 `Tensor`，`bool` 类型。秩必须大于等于 1 且小于等于 5。
- **count** (`int`, 可选) - 取样数量，必须大于 0。默认 256。
- **seed** (`int`, 可选) - 指定随机种子，用于随机数生成器生成伪随机数。默认 `None`。

返回：

两个 `Tensor`，第一个为索引，另一个为掩码。

- **index** (`Tensor`) - 二维 `Tensor`。
- **mask** (`Tensor`) - 一维 `Tensor`。

异常：

- **TypeError** - `count` 不是 `int` 类型。
- **TypeError** - `seed` 不是 `int` 类型。
- **TypeError** - `input_x` 不是 `Tensor`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.ones(shape=[240000, 4]).astype(np.bool_))
>>> output_y, output_mask = ops.choice_with_mask(input_x)
>>> result = output_y.shape
>>> print(result)
(256, 2)
```

(下页继续)

(续上页)

```
>>> result = output_mask.shape
>>> print(result)
(256,)
```

mindspore.ops.random_categorical

`mindspore.ops.random_categorical(logits, num_sample, seed=0, dtype=mstype.int64)`

从一个分类分布中生成随机样本。

警告： Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。

参数：

- **logits** (Tensor) - 输入 Tensor。Shape 为 $(batch\_size, num\_classes)$ 的二维 Tensor。
- **num_sample** (int) - 要抽取的样本数。只允许使用常量值。
- **seed** (int) - 随机种子。只允许使用常量值。默认 0。
- **dtype** (mindspore.dtype) - 输出的类型。其值必须是 `mindspore.int16`、`mindspore.int32` 或 `mindspore.int64` 之一。默认 `mstype.int64`。

返回：

Tensor，Shape 为 $(batch\_size, num\_samples)$ 的输出 Tensor。

异常：

- **TypeError** - 如果 `dtype` 不是以下类型之一：`mindspore.int16`、`mindspore.int32`、`mindspore.int64`。
- **TypeError** - 如果 `logits` 不是 Tensor。
- **TypeError** - 如果 `num_sample` 或 `seed` 不是 int 类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import ops
>>> from mindspore import Tensor
>>> import mindspore.common.dtype as mstype
>>> import numpy as np
>>> logits = Tensor(np.random.random((10, 5)).astype(np.float32), mstype.float32)
>>> net = ops.random_categorical(logits, 8)
>>> result = net.shape
>>> print(result)
(10, 8)
```

mindspore.ops.log_uniform_candidate_sampler

`mindspore.ops.log_uniform_candidate_sampler` (*true_classes*, *num_true*=1, *num_sampled*=5, *unique*=True, *range_max*=5, *seed*=0)

使用 log-uniform(Zipfian) 分布对一组类别进行采样。

该操作从整数范围 $[0, \text{range\_max})$ 中随机采样一个采样类 (*sampled_candidates*) 的 Tensor。

警告： Ascend 后端不支持随机数重现功能，*seed* 参数不起作用。

参数：

- **true_classes** (Tensor) - 目标类，其数据类型为 int64，shape 为 (*batch_size*, *num_true*)。
- **num_true** (int, 可选) - 每个训练样本的目标类数。默认 1。
- **num_sampled** (int, 可选) - 随机采样的类数。默认 5。
- **unique** (bool, 可选) - 确认批处理中的所有采样类是否都是唯一的。如果 *unique* 为 True，则批处理中的所有采样类都唯一。默认 True。
- **range_max** (int, 可选) - 可能的类数。当 *unique* 为 True 时，*range_max* 必须大于或等于 *num_sampled*。默认 5。
- **seed** (int, 可选) - 随机种子，必须是非负。默认 0。

返回：

3 个 Tensor 组成的元组。

- **sampled_candidates** (Tensor) - shape 为 (*num_sampled*,) 且数据类型与 *true_classes* 相同的 Tensor。
- **true_expected_count** (Tensor) - shape 与 *true_classes* 相同且数据类型为 float32 的 Tensor。
- **sampled_expected_count** (Tensor) - shape 与 *sampled_candidates* 相同且数据类型为 float32 的 Tensor。

异常：

- **TypeError** - *num_true* 和 *num_sampled* 都不是 int。
- **TypeError** - *unique* 不是 bool。
- **TypeError** - *range_max* 和 *seed* 都不是 int。
- **TypeError** - *true_classes* 不是 Tensor。

支持平台：

Ascend CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> output1, output2, output3 = ops.log_uniform_candidate_sampler(
...   Tensor(np.array([[1, 7], [0, 4], [3, 3]])), 2, 5, True, 5)
>>> print(output1, output2, output3)
[3 2 0 4 1]
[[0.92312991 0.49336370]
 [0.99248987 0.65806371]
 [0.73553443 0.73553443]]
[0.73553443 0.82625800 0.99248987 0.65806371 0.92312991]
```

`mindspore.ops.uniform_candidate_sampler`

`mindspore.ops.uniform_candidate_sampler` (*true_classes, num_true, num_sampled, unique, range_max, seed=0, remove_accidental_hits=False*)

使用均匀分布对一组类别进行采样。

此函数使用均匀分布从 $[0, \text{range\_max}-1]$ 中采样一组类 (`sampled_candidates`)。如果 `unique` 为 `True`，则候选采样没有重复；如果 `unique` 为 `False`，则有重复。

警告:

- Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。
- Ascend 后端暂不支持动态 `shape` 场景。

参数:

- **true_classes** (Tensor) - 输入 Tensor，目标类，其 `shape` 为 $(\text{batch\_size}, \text{num\_true})$ 。其元素值范围需要在 $[0, \text{range\_max})$ 。
- **num_true** (int) - 每个训练样本的目标类数。
- **num_sampled** (int) - 随机采样的类数。`sampled_candidates` 的 `shape` 将为 `num_sampled`。如果 `unique` 为 `True`，则 `num_sampled` 必须小于或等于 `range_max`。
- **unique** (bool) - 表示一个 batch 中的所有采样类是否唯一。
- **range_max** (int) - 可能的类数，该值必须大于 0。
- **seed** (int) - 随机种子，该值必须是非负的。如果 `seed` 的值为 0，则 `seed` 的值将被随机生成的值替换。默认 0。
- **remove_accidental_hits** (bool) - 表示是否移除 accidental hit。accidental hit 表示其中一个 `true_classes` 目标类匹配 `sampled_candidates` 采样类之一，设置为 `True` 表示移除等于目标类的采样类。默认 `False`。

返回:

- **sampled_candidates** (Tensor) - 候选采样与目标类之间不存在联系，其 `shape` 为 $(\text{num\_sampled},)$ 。
- **true_expected_count** (Tensor) - 在每组目标类的采样分布下的预期计数。`shape` 为 $(\text{batch\_size}, \text{num\_true})$ 。
- **sampled_expected_count** (Tensor) - 每个候选采样分布下的预期计数。`shape` 为 $(\text{num\_sampled},)$ 。

异常:

- **TypeError** - `num_true` 和 `num_sampled` 都不是 int。
- **TypeError** - `unique` 和 `remove_accidental_hits` 都不是 bool。
- **TypeError** - `range_max` 和 `seed` 都不是 int。
- **TypeError** - `true_classes` 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> data = Tensor(np.array([[1], [3], [4], [6], [3]], dtype=np.int64))
```

(下页继续)

(续上页)

```
>>> output1, output2, output3 = ops.uniform_candidate_sampler(data, 1, 3, False, 4, 1)
>>> print(output1.shape)
(3,)
>>> print(output2.shape)
(5, 1)
>>> print(output3.shape)
(3,)
```

2.3.6 图像函数

接口名	概述	支持平台
<code>mindspore.ops.affine_grid</code>	基于输入的批量仿射矩阵 <i>theta</i> ，返回一个二维或三维的流场（采样网格）。	Ascend GPU CPU
<code>mindspore.ops.bounding_box_decode</code>	解码边界框位置信息，计算偏移量。	Ascend GPU CPU
<code>mindspore.ops.bounding_box_encode</code>	编码边界框位置信息，计算预测边界框和真实边界框之间的偏移，并将此偏移作为损失变量。	Ascend GPU CPU
<code>mindspore.ops.col2im</code>	将一组滑动局部块组合为一个完整的 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.check_valid</code>	检查边界框是否在图片内。	Ascend GPU CPU
<code>mindspore.ops.crop_and_resize</code>	对输入图像 Tensor 进行裁剪并调整其大小。	Ascend GPU CPU
<code>mindspore.ops.grid_sample</code>	给定一个输入和一个网格，使用网格中的输入值和像素位置计算输出。	Ascend GPU CPU
<code>mindspore.ops.interpolate</code>	按照给定的 <i>size</i> 或 <i>scale_factor</i> 根据 <i>mode</i> 设置的插值方式，对输入 <i>input</i> 调整大小。	Ascend GPU CPU
<code>mindspore.ops.iou</code>	计算矩形的 IOU，即真实区域和预测区域的交并比。	Ascend GPU CPU
<code>mindspore.ops.pad</code>	根据参数 <i>padding</i> 对输入进行填充。	Ascend GPU CPU
<code>mindspore.ops.padding</code>	通过填充 0，将输入 Tensor 的最后一个维度从 1 扩展到指定大小。	Ascend GPU CPU
<code>mindspore.ops.pixel_shuffle</code>	对输入 <i>input</i> 应用像素重组操作，它实现了步长为 1/ <i>r</i> 的子像素卷积。	Ascend GPU CPU
<code>mindspore.ops.pixel_unshuffle</code>	对输入 <i>input</i> 应用逆像素重组操作，这是像素重组的逆操作。	Ascend GPU CPU
<code>mindspore.ops.rotary_position_embedding</code>	旋转位置编码算法的实现。	Ascend
<code>mindspore.ops.rotated_iou</code>	计算旋转矩形之间的重叠面积。	Ascend
<code>mindspore.ops.upsample</code>	<code>mindspore.ops.interpolate()</code> 的别名。	Ascend GPU CPU

mindspore.ops.affine_grid

`mindspore.ops.affine_grid(theta, size, align_corners=False)`
基于输入的批量仿射矩阵 *theta*，返回一个二维或三维的流场（采样网格）。
参数：

- **theta** (Tensor) - 仿射矩阵输入，其 *shape* 为 (*N*, 2, 3) 用于 2D grid 或 (*N*, 3, 4) 用于 3D grid。
- **size** (tuple[int]) - 目标输出图像大小。指格式为 (*N*, *C*, *H*, *W*) 的 2D grid 或 (*N*, *C*, *D*, *H*, *W*) 的 3D grid 的大小。
- **align_corners** (bool, 可选) - 在几何上，我们将输入的像素视为正方形而不是点。如果设置为 `True`，则极值 -1 和 1 指输入像素的中心。如果设置为 `False`，则极值 -1 和 1 指输入像素的边角，从而使采样与分辨

率无关。默认 `False`。

返回：

Tensor，其数据类型与 *theta* 相同，其 shape 为 $(N, H, W, 2)$ 用于 2D grid 或 $(N, D, H, W, 3)$ 用于 3D grid。

异常：

- **TypeError** - *theta* 不是 Tensor 或 *size* 不是 tuple。
- **ValueError** - *theta* 的 shape 不是 $(N, 2, 3)$ 或 $(N, 3, 4)$ 。
- **ValueError** - *size* 的长度不是 4 或 5。
- **ValueError** - *theta* 的 shape 是 $(N, 2, 3)$ ，*size* 的长度却不是 4；*theta* 的 shape 是 $(N, 3, 4)$ ，*size* 的长度却不是 5。
- **ValueError** - *size* 的第一个值不等于 *theta* 的第一维的长度。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> theta = Tensor([[[[0.8, 0.5, 0], [-0.5, 0.8, 0]]], mindspore.float32)
>>> out_size = (1, 3, 2, 3)
>>> output = ops.affine_grid(theta, out_size, False)
>>> print(output)
[[[[-0.78333336 -0.06666666]
[-0.25         -0.4         ]
[ 0.28333336 -0.73333335]]]
 [[-0.28333336  0.73333335]
 [ 0.25         0.4         ]
 [ 0.78333336  0.06666666]]]]
```

`mindspore.ops.bounding_box_decode`

`mindspore.ops.bounding_box_decode` (*anchor_box*, *deltas*, *max_shape*, *means*=(0.0, 0.0, 0.0, 0.0), *stds*=(1.0, 1.0, 1.0, 1.0), *wh_ratio_clip*=0.016)

解码边界框位置信息，计算偏移量。此算子将偏移量转换为 Bbox，用于在后续图像中标记目标等。

参数：

- **anchor_box** (Tensor) - 锚框。锚框的 shape 必须为 $(n, 4)$ 。
- **deltas** (Tensor) - 框的增量。它的 shape 与 *anchor_box* 相同。
- **max_shape** (tuple) - 解码框计算的上限值。
- **means** (tuple, 可选) - 计算 *deltas* 的均值。默认 (0.0, 0.0, 0.0, 0.0)。
- **stds** (tuple, 可选) - 计算 *deltas* 的标准差。默认 (1.0, 1.0, 1.0, 1.0)。
- **wh_ratio_clip** (float, 可选) - 解码框计算的宽高比限制。默认 0.016。

返回：

Tensor，解码框。它的数据类型和 shape 与 *anchor_box* 相同。

异常：

- **TypeError** - 如果 *means*、*stds* 或 *max_shape* 不是 tuple。
- **TypeError** - 如果 *wh_ratio_clip* 不是 float。
- **TypeError** - 如果 *anchor_box* 或 *deltas* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> anchor_box = Tensor([[4, 1, 2, 1], [2, 2, 2, 3]], mindspore.float32)
>>> deltas = Tensor([[3, 1, 2, 2], [1, 2, 1, 4]], mindspore.float32)
>>> output = ops.bounding_box_decode(anchor_box, deltas, max_shape=(768, 1280), means=(0.0, 0.0, 0.0, 0.0),
...                                stds=(1.0, 1.0, 1.0, 1.0), wh_ratio_clip=0.016)
>>> print(output)
[[ 4.1953125  0.          0.          5.1953125]
 [ 2.140625  0.          3.859375  60.59375  ]]
```

mindspore.ops.bounding_box_encode

mindspore.ops.bounding_box_encode (*anchor_box*, *groundtruth_box*, *means*=(0.0, 0.0, 0.0, 0.0), *stds*=(1.0, 1.0, 1.0, 1.0))

编码边界框位置信息，计算预测边界框和真实边界框之间的偏移，并将此偏移作为损失变量。

参数：

- **anchor_box** (Tensor) - 锚框。锚框的 *shape* 必须为 $(n, 4)$ 。
- **groundtruth_box** (Tensor) - 真实边界框。它的 *shape* 与锚框相同。
- **means** (tuple, 可选) - 计算编码边界框的均值。默认 (0.0, 0.0, 0.0, 0.0)。
- **stds** (tuple, 可选) - 计算增量的标准偏差。默认 (1.0, 1.0, 1.0, 1.0)。

返回：

Tensor，编码边界框。数据类型和 *shape* 与输入 *anchor_box* 相同。

异常：

- **TypeError** - 如果 *means* 或 *stds* 不是 tuple。
- **TypeError** - 如果 *anchor_box* 或 *groundtruth_box* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> anchor_box = Tensor([[2, 2, 2, 3], [2, 2, 2, 3]], mindspore.float32)
>>> groundtruth_box = Tensor([[1, 2, 1, 4], [1, 2, 1, 4]], mindspore.float32)
>>> output = ops.bounding_box_encode(anchor_box, groundtruth_box, means=(0.0, 0.0, 0.0, 0.0),
...                                stds=(1.0, 1.0, 1.0, 1.0))
>>> print(output)
```

(下页继续)

(续上页)

```
[[ -1.   0.25  0.   0.40551758]
 [ -1.   0.25  0.   0.40551758]]
```

mindspore.ops.col2im

`mindspore.ops.col2im(input_x, output_size, kernel_size, dilation, padding_value, stride)`

将一组滑动局部块组合为一个完整的 Tensor。

参数：

- **input_x** (Tensor) - 四维 Tensor，输入的批量滑动局部块，数据类型支持 float16 和 float32。
- **output_size** (Tensor) - 包含两个 int 元素的一维 Tensor，输出 Tensor 的后两维的 shape。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 滑动窗口的大小。tuple 的两个元素分别对应 kernel 的高度与宽度。如果为一个 int，则 kernel 的高度与宽度均为该值。
- **dilation** (Union[int, tuple[int], list[int]]) - 滑动窗口扩张的大小。
- **padding_value** (Union[int, tuple[int], list[int]]) - 填充的大小。
- **stride** (Union[int, tuple[int], list[int]]) - 步长的大小。

返回：

Tensor，输出的 Tensor，与输入维度相同且类型一致。

异常：

- **TypeError** - 如果 `kernel_size`, `dilation`, `padding_value`, `stride` 不属于 Union[int, tuple[int], list[int]]。
- **ValueError** - 如果 `kernel_size`, `dilation`, `stride` 值小于等于 0 或者个数大于 2。
- **ValueError** - 如果 `padding_value` 值小于 0 或者个数大于 2。
- **ValueError** - 如果 `input_x` 的第三维长度不等于 `kernel_size[0] * kernel_size[1]`。
- **ValueError** - 如果 `input_x` 的第四维长度与计算出的滑动块数量不匹配。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(input_data=np.random.rand(16, 16, 4, 25), dtype=mstype.float32)
>>> output_size = Tensor(input_data=[8, 8], dtype=mstype.int32)
>>> output = ops.col2im(x, output_size, [2, 2], [2, 2], [2, 2], [2, 2])
>>> print(output.shape)
(16, 16, 8, 8)
```

mindspore.ops.check_valid

`mindspore.ops.check_valid(bboxes, img metas)`

检查边界框是否在图片内。

bboxes 里包含了多组边界框，每一组边界框用两个横坐标点 (x_0, x_1) 和两个纵坐标点 (y_0, y_1) 表示。*img metas* 提供了原始图片的信息，包含 (*height*, *width*, *ratio*) 三个参数，用于指定图片的有效边界。当满足：

$x_0 \geq 0$

$y_0 \geq 0$

$x_1 \leq \text{width} * \text{ratio} - 1$

$y_1 \leq \text{height} * \text{ratio} - 1$

时，认为边界框在图片内。

警告：由 *bboxes* 指定的边界框和由 *img metas* 指定的图片信息需要是有效的，即： $x_0 \leq x_1$ ， $y_0 \leq y_1$ ，*img metas* 中的信息 (*height*, *width*, *ratio*) 均为正数。

参数：

- **bboxes** (Tensor) - shape 大小为 ($N, 4$)。 N 表示边界框的数量，4 表示 (x_0, y_0, x_1, y_1) 四个坐标点。数据类型必须是 float16 或 float32。
- **img metas** (Tensor) - 原始图片的信息 (*height*, *width*, *ratio*)，指定有效边界为 ($\text{height} * \text{ratio} - 1, \text{width} * \text{ratio} - 1$)。数据类型必须是 float16 或 float32。

返回：

Tensor，shape 为 (N)，类型为 bool，表示边界框是否在图片内。*True* 表示在，*False* 表示不在。

异常：

- **TypeError** - 如果 *bboxes* 或者 *img metas* 不是 Tensor。
- **TypeError** - 如果 *bboxes* 或者 *img metas* 的数据类型既不是 float16，也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bboxes = Tensor(np.linspace(0, 6, 12).reshape(3, 4), mindspore.float32)
>>> img_metas = Tensor(np.array([2, 1, 3]), mindspore.float32)
>>> output = ops.check_valid(bboxes, img_metas)
>>> print(output)
[ True False False]
```


mindspore.ops.crop_and_resize

`mindspore.ops.crop_and_resize` (*image*, *boxes*, *box_indices*, *crop_size*, *method*='bilinear', *extrapolation_value*=0.0)
对输入图像 Tensor 进行裁剪并调整其大小。

说明：当输出的 shape 依赖 *crop_size* 的时候，*crop_size* 必须为常量。当前该算子的反向仅支持“bilinear”模式，其他模式将会返回 0。

参数：

- **image** (Tensor) - shape 为 (*batch*, *image_height*, *image_width*, *depth*) 的图像 Tensor。
- **boxes** (Tensor) - shape 为 (*num_boxes*, 4) 的二维 Tensor，表示归一化的边框坐标，坐标格式为 [*y1*, *x1*, *y2*, *x2*]。其中 (*y1*, *x1*) 为第一个角点，(*y2*, *x2*) 为第二个角点。如果 *y1* > *y2*，就是对图像进行的上下翻转，当 *x1* > *x2*，宽度方向操作类似。如果归一化的坐标值超出 [0, 1] 的区间，采用 *extrapolation_value* 进行填充。数据类型：float32。
- **box_indices** (Tensor) - shape 为 (*num_boxes*) 的一维 Tensor，表示每个方框的索引。数据类型：int32。
- **crop_size** (Tuple[int]) - 2 元组 (*crop_height*, *crop_width*)，指定对裁剪出的图像进行调整时的输出大小，元素均为正值。数据类型：int32。
- **method** (str, 可选) - 指定调整大小的采样方法，为可选字符串。提供的方法有："bilinear"、"nearest" 或 "bilinear_v2"。默认 "bilinear"。
 - "nearest"：最近邻插值。每个输出像素的值为最近的输入像素的值。这种方法简单快速，但可能导致块状或像素化的输出。
 - "bilinear"：双线性插值。每个输出像素是最接近的四个输入像素的加权平均值，使用双线性插值计算。与最近邻插值相比，此方法产生更平滑的结果。
 - "bilinear_v2"：优化后的双线性插值算法，在某些情况下可能会产生更好的结果（更高的精度和速度）。
- **extrapolation_value** (float, 可选) - 指定外插时的浮点值。默认 0.0。

返回：

Tensor, shape 为 (*num_boxes*, *crop_height*, *crop_width*, *depth*)，数据类型：float32。

异常：

- **TypeError** - *image*、*boxes* 或 *box_indices* 不是 Tensor。
- **TypeError** - *crop_size* 不是元素类型为 int32 的 tuple，或 *crop_size* 的长度不为 2。
- **TypeError** - *boxes* 的数据类型不是 float，或 *box_indices* 的数据类型不是 int32。
- **TypeError** - *method* 不是字符串。
- **TypeError** - *extrapolation_value* 不是浮点值。
- **ValueError** - *image* 的维度不是四维。
- **ValueError** - *boxes* 的维度不是二维。
- **ValueError** - *boxes* 的第二维不是 4。
- **ValueError** - *box_indices* 的维度不是一维。
- **ValueError** - *box_indices* 的第一维与 *boxes* 的第一维不相等。
- **ValueError** - *box_indices* 存在元素不在 [0, *batch*) 的范围内。

- **ValueError** - *crop_size* 的数据不是正整数。
- **ValueError** - *method* 不是“bilinear”、“nearest”、“bilinear_v2”之一。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import ops, Tensor
>>> BATCH_SIZE = 1
>>> NUM_BOXES = 5
>>> IMAGE_HEIGHT = 256
>>> IMAGE_WIDTH = 256
>>> CHANNELS = 3
>>> image = np.random.normal(size=[BATCH_SIZE, IMAGE_HEIGHT, IMAGE_WIDTH, CHANNELS]).
↳ astype(np.float32)
>>> boxes = np.random.uniform(size=[NUM_BOXES, 4]).astype(np.float32)
>>> box_indices = np.random.uniform(size=[NUM_BOXES], low=0, high=BATCH_SIZE).astype(np.
↳ int32)
>>> crop_size = (24, 24)
>>> output = ops.crop_and_resize(Tensor(image), Tensor(boxes), Tensor(box_indices), crop_
↳ size)
>>> print(output.shape)
(5, 24, 24, 3)
```

mindspore.ops.grid_sample

`mindspore.ops.grid_sample(input, grid, mode='bilinear', padding_mode='zeros', align_corners=False)`

给定一个输入和一个网格，使用网格中的输入值和像素位置计算输出。*input* 只支持 4-D (GridSampler2D) 和 5-D (GridSampler3D)。

在 4-D 场景下，*input* 的 shape 为 (N, C, H_{in}, W_{in}) ，*grid* 的 shape 为 $(N, H_{out}, W_{out}, 2)$ ，*output* 的 shape 为 (N, C, H_{out}, W_{out}) 。对于每个输出位置 $output[n, :, h, w]$ ， $grid[n, h, w]$ 指定 *input* 像素位置 x 和 y ，用于计算 $output[n, :, h, w]$ 的插值。以 5D 为例， $grid[n, d, h, w]$ 指定 x, y, z 像素位置的插值位置为 $[n, :, d, h, w]$ 。*mode* 参数指定 *nearest* 或 *bilinear* (bicubic 暂不支持) 插值法对输入像素进行采样。

grid 指定由 *input* 归一化的采样像素位置。因此，它应该在 $[-1, 1]$ 范围内的值最多。

如果 *grid* 的值在 $[-1, 1]$ 范围之外，则相应的输出将按照定义的 *padding_mode* 方式处理。如果 *padding_mode* 设置为 0，则使用 0 来表示出界的网格位置。如果 *padding_mode* 设置为 border，对于出界网格位置，则使用 border 值。如果 *padding_mode* 设置为 reflection，请使用边界所反映的位置的值用于指定出界网格位置。对于远离边界的位置，它会一直被反射，直到在边界内。

参数：

- **input** (Tensor) - 4-D 场景下，shape 为 (N, C, H_{in}, W_{in}) ，5-D 场景下，shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 。数据类型为 float32 或 float64。
- **grid** (Tensor) - 4-D 场景下，shape 为 $(N, H_{out}, W_{out}, 2)$ ，5-D 场景下，shape 为 $(N, D_{out}, H_{out}, W_{out}, 3)$ 。数据类型与 *input* 保持一致。
- **mode** (str, 可选) - 插值方法。可选方法为 'bilinear', 'nearest'。默认 'bilinear'。注：'bilinear' 还不支持。当 *mode* 为 'bilinear'，且输入为 5-D，则 *mode* 为 'trilinear'。但是，当输入为 4-D，则 *mode* 为 'bilinear'。默认 'bilinear'。
 - 'nearest'：最近邻插值。每个输出像素的值为最近的输入像素的值。这种方法简单快速，但可能导致块状或像素化的输出。

- 'bilinear': 双线性插值。每个输出像素是最接近的四个输入像素的加权平均值，使用双线性插值计算。与最近邻插值相比，此方法产生更平滑的结果。
- 'trilinear': 三线性插值。这是双线性插值在三维数据上的扩展。它在两个空间维度上执行双线性插值，并沿第三个维度进行线性插值。通常用于体积或三维图像插值。
- **padding_mode** (str, 可选) - 填充方法。可选方法为 'zeros', 'border' 和 'reflection'。默认 'zeros'。
- **align_corners** (bool, 可选) - 如果设置成 *True*，-1 和 1 被视为引用输入角像素的中心点。如果设置为 *False*，将被视为引用到输入角像素的角点，使采样更不受分辨率影响。默认值为 *False*。

返回：

Tensor，数据类型与 *input* 相同，4-D 场景下，shape 为 (N, C, H_{out}, W_{out}) ，5-D 场景下，shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 。

异常：

- **TypeError** - 如果 *input* 或 *grid* 不是 Tensor 类型。
- **TypeError** - 如果 *input* 和 *grid* 的数据类型不一致。
- **TypeError** - 如果 *input* 或 *grid* 的数据类型无效。
- **TypeError** - 如果 *align_corners* 不是一个布尔值。
- **ValueError** - 如果 *input* 或 *grid* 的维度不是四维或五维。
- **ValueError** - 如果 *input* 的第一个维度不等于 *grid* 的第一个维度。
- **ValueError** - 如果 *grid* 最后一个维度不等于 2（4-D 场景）或者 3（5-D 场景）。
- **ValueError** - 如果 *mode* 不是 *bilinear*，*nearest*，数据类型不为 String。
- **ValueError** - 如果 *padding_mode* 不是 *zeros*，*border*，*reflection*，数据类型不为 String。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.arange(16).reshape((2, 2, 2, 2)).astype(np.float32))
>>> grid = Tensor(np.arange(0.2, 1, 0.1).reshape((2, 2, 1, 2)).astype(np.float32))
>>> output = ops.grid_sample(input_x, grid, mode='bilinear', padding_mode='zeros',
...                          align_corners=True)
>>> print(output)
[[[ [ 1.9      ]
     [ 2.1999998]]
  [ [ 5.9      ]
     [ 6.2      ]]]]
[[[10.5      ]
   [10.8      ]]
 [[14.5      ]
   [14.8      ]]]]
```

mindspore.ops.interpolate

`mindspore.ops.interpolate` (*input*, *size=None*, *scale_factor=None*, *mode='nearest'*, *align_corners=None*, *recompute_scale_factor=None*)

按照给定的 *size* 或 *scale_factor* 根据 *mode* 设置的插值方式，对输入 *input* 调整大小。

参数：

- **input** (Tensor) - 被调整大小的 Tensor。输入向量必须为三维、四维或五维，shape 为 $(N, C, [optionalD], [optionalH], W)$ ，数据类型为 float。
- **size** (Union[int, tuple[int], list[int]], 可选) - 目标大小。如果 *size* 为 tuple 或 list，那么其长度应该和 *input* 去掉 *N*, *C* 的维度相同。*size* 和 *scale_factor* 同时只能指定一个。默认 None。
- **scale_factor** (Union[float, tuple[float], list[float]], 可选) - 每个维度的缩放系数。如果 *scale_factor* 为 tuple 或 list，那么其长度应该和 *input* 去掉 *N*, *C* 的维度相同。*size* 和 *scale_factor* 同时只能指定一个。默认 None。
- **mode** (str, 可选) - 采样算法。默认 "nearest"。可采用以下采样方式的一种：
 - 'nearest': 最近邻插值。
 - 'linear': 线性插值，仅三维。
 - 'bilinear': 双线性插值，仅四维。
 - 'trilinear': 三线性插值，仅五维。
 - 'bicubic': 双三次插值，仅四维。
 - 'area': 区域插值。
 - 'nearest-exact': 此方法类似于 Scikit-Image 和 PIL 最邻插值，可解决一些最邻插值的问题，三维和四维。
- **align_corners** (bool, 可选) - 是否使用角对齐进行坐标映射。假设对输入 Tensor 沿 x 轴进行变换，具体计算公式如下：

```
ori_i = new_length != 1 ? new_i * (ori_length - 1) / (new_length - 1) : 0 # 'align_
↪corners' 为 True

ori_i = new_length > 1 ? (new_i + 0.5) * ori_length / new_length - 0.5 : 0 # 'align_
↪corners' 为 False
```

其中，*ori_length* 与 *new_length* 分别表示 Tensor 在 x 轴方向上转换前、后的长度，*new_i* 表示转换后沿 x 轴第 *i* 个元素的坐标，*ori_i* 表示沿 x 轴原始数据的对应坐标。

此选项只对 'linear'、'bilinear' 和 'bicubic' 模式有效，默认 False。

- **recompute_scale_factor** (bool, 可选) - 重计算 *scale_factor*。默认 None。
 - 如果为 True，会使用参数 *scale_factor* 计算参数 *size*，最终使用 *size* 的值进行缩放。
 - 如果为 False，将使用 *size* 或 *scale_factor* 直接进行插值。

说明：'nearest-exact' 模式与 scikit-image 和 PIL 的最近邻插值算法相同。'nearest' 模式与 OpenCV 的 INTER_NEAREST 插值算法结果相同。

参数支持列表和支持平台：

mode	input.dim	align_corners	scale_factor	device
nearest	3	-	×	Ascend,GPU,CPU
	4	-	×	Ascend,GPU,CPU
	5	-	√	Ascend,GPU,CPU
linear	3	√	×	Ascend,GPU,CPU
bilinear	4	√	×	Ascend,GPU,CPU
bicubic	4	√	×	Ascend,GPU,CPU
area	3	-	√	Ascend,GPU,CPU
	4	-	√	Ascend,GPU,CPU
	5	-	√	Ascend,GPU,CPU
nearest-exact	3	-	×	Ascend,CPU
	4	-	×	Ascend,CPU
trilinear	5	√	√	Ascend,GPU,CPU

- - 表示无此参数。
- × 表示当前不支持此参数。
- √ 表示当前支持此参数。

返回：

调整大小之后的 Tensor，维度和数据类型与 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *size* 和 *scale_factor* 都不为空。
- **ValueError** - *size* 和 *scale_factor* 都为空。
- **ValueError** - *size* 为元组或列表类型时长度不等于 *input.ndim - 2*。
- **ValueError** - *scale_factor* 为元组或列表类型时长度不等于 *input.ndim - 2*。
- **ValueError** - *mode* 不在模式支持列表中。
- **ValueError** - *input.ndim* 不在模式对应维度的支持列表中。
- **ValueError** - *size* 不为空，*recompute_scale_factor* 不为空。
- **ValueError** - *scale_factor* 不在对应的支持列表中。
- **ValueError** - *align_corners* 不在对应的支持列表中。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input = Tensor([[[[1, 2, 3], [4, 5, 6]]], mindspore.float32)
>>> output = ops.interpolate(input, size=(6,), mode='nearest')
>>> print(output)
[[[1. 1. 2. 2. 3. 3.]
  [4. 4. 5. 5. 6. 6.]]]
```

mindspore.ops.iou

`mindspore.ops.iou(anchor_boxes, gt_boxes, mode='iou')`

计算矩形的 IOU，即真实区域和预测区域的交并比。

根据真实区域和预测区域计算 IOU(intersection over union) 或 IOF(intersection over foreground)。

$$\text{IOU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

$$\text{IOF} = \frac{\text{Area of Overlap}}{\text{Area of Ground Truth}}$$

警告： 在 Ascend 中，仅支持计算 float16 数据。为避免溢出，输入长度和宽度在内部按 0.2 缩放。

参数：

- **anchor_boxes** (Tensor) - 预测区域，shape 为 $(N, 4)$ 的 Tensor。 N 表示预测区域的数量，4 表示预测区域的四个边界坐标“x0”、“y0”、“x1”和“y1”。数据类型为 float16、float32 或 float64。
- **gt_boxes** (Tensor) - 真实区域，shape 为 $(M, 4)$ 的 Tensor。 M 表示地面真实区域的数量，4 表示真实区域的四个边界坐标“x0”、“y0”、“x1”和“y1”。数据类型为 float16、float32 或 float64。
- **mode** (str) - 指定计算方法，现支持 'iou' (intersection over union) 或 'iof' (intersection over foreground) 模式。默认 'iou'。

返回：

Tensor，IOU/IOF 的计算结果，shape 为 (M, N) 的 Tensor，数据类型与 *anchor_boxes* 的相同。

异常：

- **KeyError** - *mode* 不是 'iou' 或 'iof'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> anchor_boxes = Tensor(np.random.randint(1.0, 5.0, [3, 4]), mindspore.float16)
>>> gt_boxes = Tensor(np.random.randint(1.0, 5.0, [3, 4]), mindspore.float16)
>>> mode = 'iou'
>>> output = ops.iou(anchor_boxes, gt_boxes, mode)
>>> print(output.shape)
(3, 3)
```

`mindspore.ops.pad`

`mindspore.ops.pad(input_x, padding, mode='constant', value=None)`

根据参数 `padding` 对输入进行填充。

参数：

- **input_x** (Tensor) - 输入 Tensor, shape 为 $(N,*)$, * 代表任意附加维度。在 Ascend 后端运行时, 不支持该维度大于 5 的情况。
- **padding** (Union[tuple[int], list[int], Tensor]) - pad 的填充位置。在 Ascend 后端运行时, 不支持 `padding` 包含负值情况。 $\left\lfloor \frac{\text{len(padding)}}{2} \right\rfloor$ 维度的 `input_x` 将会被填充。可根据以下示例以此类推：
 - 示例：若只需要填充输入 tensor 的最后一个维度, 则 `padding` 的填充方式为 `(padding_left, padding_right)`;
 - 示例：若只需要填充输入 tensor 的最后两个维度, 则 `padding` 的填充方式为 `(padding_left, padding_right, padding_top, padding_bottom)`;
 - 示例：若只需要填充输入 tensor 的最后三个维度, 则 `padding` 的填充方式为 `(padding_left, padding_right, padding_top, padding_bottom, padding_front, padding_back)`;
- **mode** (str, 可选) - Pad 的填充模式, 可选择 'constant'、'reflect'、'replicate' 或者 'circular'。默认 'constant'。
 - 对于 'constant' 模式, 请参考 `mindspore.nn.ConstantPad1d` 作为示例来理解这个填充模式, 并将这个模式扩展到 n 维。
 - 对于 'reflect' 模式, 请参考 `mindspore.nn.ReflectionPad1d` 作为示例来理解这个填充模式, reflect 模式用于填充三维或者四维输入的最后两个维度, 或者二维或三维输入的最后一个维度。
 - 对于 'replicate' 模式, 请参考 `mindspore.nn.ReplicationPad1d` 作为示例来理解这个填充模式, replicate 模式用于填充四维或五维输入的最后三个维度、三维或四维输入的最后两个维度, 或者二维或三维输入的最后一个维度。
 - 对于 'circular' 模式, circular 模式用于将图像的像素从一侧循环地填充到另一侧。例如, 右侧的像素将被替换为左侧的像素, 底部的像素将被替换为顶部的像素。circular 模式用于填充四维或五维输入的最后三个维度、三维或四维输入的最后两个维度, 或者二维或三维输入的最后一个维度。
- **value** (Union[int, float, None], 可选) - 仅在 'constant' 模式下生效, 设置在 'constant' 模式下的填充值, 如果值为 None, 则会使用 0 作为默认填充值。默认 None。

返回：

填充后的 Tensor。

异常：

- **TypeError** - `padding` 不是全为 int 的 tuple 或者 list。
- **TypeError** - `input_x` 不是 Tensor。
- **ValueError** - `padding` 的长度不为偶数。
- **ValueError** - `padding` 的长度大于 6。
- **ValueError** - `mode` 不为 'constant' 并且 `value` 不为 None。
- **ValueError** - 在 Ascend 后端运行时, `input_x` 的维度大于 5。
- **ValueError** - 在 Ascend 后端运行时, `padding` 中包含负值。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore as ms
>>> from mindspore import ops
>>> import numpy as np
>>> x = ms.Tensor(np.arange(1 * 2 * 2 * 2).reshape((1, 2, 2, 2)), dtype=ms.float64)
>>> output = ops.pad(x, [1, 0, 0, 1], mode='constant', value=6.0)
>>> print(output)
[[[6. 0. 1.]
  [6. 2. 3.]
  [6. 6. 6.]]
 [[6. 4. 5.]
  [6. 6. 7.]
  [6. 6. 6.]]]]
>>> output1 = ops.pad(x, (1, 0, 0, 1), mode='reflect')
>>> print(output1)
[[[1. 0. 1.]
  [3. 2. 3.]
  [1. 0. 1.]]
 [[5. 4. 5.]
  [7. 6. 7.]
  [5. 4. 5.]]]]
>>> output2 = ops.pad(x, (1, 1, 2, 1), mode='replicate')
>>> print(output2)
[[[0. 0. 1. 1.]
  [0. 0. 1. 1.]
  [0. 0. 1. 1.]
  [2. 2. 3. 3.]
  [2. 2. 3. 3.]]
 [[4. 4. 5. 5.]
  [4. 4. 5. 5.]
  [4. 4. 5. 5.]
  [6. 6. 7. 7.]
  [6. 6. 7. 7.]]]]
>>> output3 = ops.pad(x, (1, 1, 2, 1), mode='circular')
>>> print(output3)
[[[1. 0. 1. 0.]
  [3. 2. 3. 2.]
  [1. 0. 1. 0.]
  [3. 2. 3. 2.]
  [1. 0. 1. 0.]]
 [[5. 4. 5. 4.]
  [7. 6. 7. 6.]
  [5. 4. 5. 4.]
  [7. 6. 7. 6.]
  [5. 4. 5. 4.]]]]

```

mindspore.ops.padding

mindspore.ops.padding (*x*, *pad_dim_size*=8)

通过填充 0，将输入 Tensor 的最后一个维度从 1 扩展到指定大小。

参数：

- **x** (Tensor) - *x* 的 shape 为 $(x_1, x_2, \dots, x_R)$ ，秩至少为 2，它的最后一个维度必须为 1。其数据类型为数值型。
- **pad_dim_size** (int) - 要扩展的 *x* 的最后一个维度的值，该值必须为正数。默认 8。

返回：

Tensor, 其数据类型和维度必须和输入中的 x 保持一致。

异常:

- **TypeError** - pad_dim_size 的数据类型不是 int。
- **ValueError** - pad_dim_size 的值小于 1。
- **ValueError** - x 的最后一个维度不等于 1。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[8], [10]]), mindspore.float32)
>>> pad_dim_size = 4
>>> output = ops.padding(x, pad_dim_size)
>>> print(output)
[[ 8.  0.  0.  0.]
 [10.  0.  0.  0.]
```

mindspore.ops.pixel_shuffle

`mindspore.ops.pixel_shuffle(input, upscale_factor)`

对输入 $input$ 应用像素重组操作, 它实现了步长为 $1/r$ 的子像素卷积。关于 `pixel_shuffle` 算法详细介绍, 请参考 [Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network](#)。

通常情况下, $input$ shape $(*, C \times r^2, H, W)$, 输出 shape $(*, C, H \times r, W \times r)$ 。 r 是缩小因子。 $*$ 是大于等于 0 的维度。

参数:

- **input** (Tensor) - Tensor, shape 为 $(*, C \times r^2, H, W)$ 。 $input$ 的维度需要大于 2, 并且倒数第三维 `length` 可以被 $upscale\_factor$ 的平方整除。
- **upscale_factor** (int) - 打乱输入 Tensor 的因子, 是正整数。 $upscale\_factor$ 是上面提到的 r 。

返回:

- **output** (Tensor) - Tensor, shape 为 $(*, C, H \times r, W \times r)$ 。

异常:

- **ValueError** - $upscale\_factor$ 不是正整数。
- **ValueError** - $input$ 倒数第三维度的 `length` 不能被 $upscale\_factor$ 的平方整除。
- **ValueError** - $input$ 维度小于 3。
- **TypeError** - $input$ 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import ops
>>> input_x = np.arange(3 * 2 * 9 * 4 * 4).reshape((3, 2, 9, 4, 4))
>>> input_x = mindspore.Tensor(input_x, mindspore.dtype.int32)
>>> output = ops.pixel_shuffle(input_x, 3)
>>> print(output.shape)
(3, 2, 1, 12, 12)
```

mindspore.ops.pixel_unshuffle

`mindspore.ops.pixel_unshuffle(input, downscale_factor)`

对输入 *input* 应用逆像素重组操作，这是像素重组的逆操作。关于 `pixel_unshuffle` 算法详细介绍，请参考 [Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network](#)。

通常情况下，*input* shape 为 $(*, C, H \times r, W \times r)$ ，输出 shape 为 $(*, C \times r^2, H, W)$ 。*r* 是缩小因子。*** 是大于等于 0 的维度。

参数：

- **input** (Tensor) - Tensor，shape 为 $(*, C, H \times r, W \times r)$ 。*input* 的维度需要大于 2，并且倒数第一和倒数第二维对应的值可以被 *downscale_factor* 整除。
- **downscale_factor** (int) - 恢复输入 Tensor 的因子，是正整数。*downscale_factor* 是上面提到的 *r*。

返回：

- **output** (Tensor) - Tensor，shape 为 $(*, C \times r^2, H, W)$ 。

异常：

- **ValueError** - *downscale_factor* 不是正整数。
- **ValueError** - *input* 倒数第一和倒数第二维度对应的值不能被 *downscale_factor* 整除。
- **ValueError** - *input* 维度小于 3。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = np.arange(8 * 8).reshape((1, 1, 8, 8))
>>> input_x = mindspore.Tensor(input_x, mindspore.dtype.int32)
>>> output = ops.pixel_unshuffle(input_x, 2)
>>> print(output.shape)
(1, 4, 4, 4)
```

mindspore.ops.rotary_position_embedding

`mindspore.ops.rotary_position_embedding(x, cos, sin, mode=0)`
旋转位置编码算法的实现。请参阅论文 [Enhanced Transformer with Rotary Position Embedding](#)。

警告：这是一个实验性 API，后续可能修改或删除。

- 参数：
- **x** (Tensor) - 4 维 Tensor，数据类型 float16、bfloat16、float32。
 - **cos** (Tensor) - 4 维常量，数据类型和 *x* 一致。取值范围为 [-1,1]。
 - **sin** (Tensor) - 和 *cos* 相同。
 - **mode** (int) - 可选属性，选择计算模式。0 表示 rotate_half (GPT-NeoX style)，1 表示 rotate_interleaved (GPT-J style)。默认 0。

表 17: Layout 配置约束

参数	RotateHalf(mode: 0)	RotateInterleaved(mode: 1)
x	layout 支持 BNSD、BSND、SBND； D < 896，且为 2 的倍数；B, N < 1000；	layout 支持：BNSD、BSND、SBND； B * N < 1000；D < 896，且 D 为 2 的倍数
cos	对应 <i>x</i> layout 的支持情况： <i>x</i> 为 BNSD：11SD、B1SD、BNSD； <i>x</i> 为 BSND：1S1D、BS1D、BSND； <i>x</i> 为 SBND：S11D、SB1D、SBND。	对应 <i>x</i> layout 的支持情况： <i>x</i> 为 BNSD：11SD； <i>x</i> 为 BSND：1S1D； <i>x</i> 为 SBND：S11D。
sin	同 <i>cos</i>	同 <i>cos</i>

说明：layout 是 BNSD，并且 D 是 32bytes 对齐、B * N > 8S 时，因为性能不好，不允许调用。

返回：
Tensor，其 shape 和 dtype 和 *x* 保持一致。

- 异常：
- **TypeError** - *x* 不是 Tensor。
 - **TypeError** - *cos* 不是 Tensor。
 - **TypeError** - *sin* 不是 Tensor。
 - **TypeError** - *mode* 不是 int。

支持平台：
Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.uniform(-2, 2, (4, 8192, 4, 128)))
>>> cos = Tensor(np.random.uniform(-1, 1, (1, 8192, 1, 128)))
>>> sin = Tensor(np.random.uniform(-1, 1, (1, 8192, 1, 128)))
```

(下页继续)

(续上页)

```
>>> output = ops.rotary_position_embedding(x, cos, sin, 0)
>>> print(output.shape)
(4, 8192, 4, 128)
```

mindspore.ops.rotated_iou

`mindspore.ops.rotated_iou` (*boxes*, *query_boxes*, *trans=False*, *mode=0*, *is_cross=True*, *v_threshold=0.0*, *e_threshold=0.0*)
计算旋转矩形之间的重叠面积。

警告：这是一个实验性 API，后续可能修改或删除。

说明：Ascend 平台支持的输入数据类型包括 bfloat16、float16、float32。

参数：

- **boxes** (Tensor) - 第一组矩形，shape 为 $(B, N, 5)$ 。
- **query_boxes** (Tensor) - 第二组矩形，shape 为 $(B, K, 5)$ 。
- **trans** (bool, 可选) - 区分 boxes 与 query_boxes 中矩形表示方法。如果为 True，格式为 'xyxyt'，如果为 False，格式为 'xywht'。默认为 False。
- **mode** (int, 可选) - 区分计算模式。如果为 1，计算方法为 'iof'，如果为 0，计算方法为 'iou'。默认为 0。
- **is_cross** (bool, 可选) - 如果为 True，采用交叉计算，如果为 False，表示一对一计算。默认为 True。
- **v_threshold** (float, 可选) - 顶点判断的容忍阈值。默认为 0.0。
- **e_threshold** (float, 可选) - 边相交判断的容忍阈值。默认为 0.0。

返回：

Tensor, shape 为 (B, N, K) 。

异常：

- **TypeError** - boxes 不是 Tensor。
- **TypeError** - query_boxes 不是 Tensor。
- **ValueError** - boxes 与 query_boxes 第一个维度不相同。
- **ValueError** - boxes 或 query_boxes 第三个维度不为 5。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> a = np.random.uniform(0, 1, (2, 2, 5)).astype(np.float16)
>>> b = np.random.uniform(0, 1, (2, 3, 5)).astype(np.float16)
>>> box1 = Tensor(a)
```

(下页继续)

(续上页)

```
>>> box2 = Tensor(b)
>>> output = ops.rotated_iou(box1, box2, trans=False, mode=0, is_cross=True)
```

mindspore.ops.upsample

`mindspore.ops.upsample` (*input*, *size=None*, *scale_factor=None*, *mode='nearest'*, *align_corners=None*, *recompute_scale_factor=None*)

`mindspore.ops.interpolate()` 的别名。

支持平台：

Ascend GPU CPU

2.4 Parameter 操作函数

接口名	概述	支持平台
<code>mindspore.ops.assign</code>	为网络参数或者 <code>tensor</code> 赋值。	Ascend GPU CPU
<code>mindspore.ops.assign_add</code>	通过加法运算更新网络参数或者 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.assign_sub</code>	通过减法运算更新网络参数或者 <code>tensor</code> 。	Ascend GPU CPU
<code>mindspore.ops.scatter_add</code>	根据指定索引和更新值对 <code>input_x</code> 进行加法更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_div</code>	根据指定索引和更新值对 <code>input_x</code> 进行除法更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_max</code>	根据指定索引和更新值对 <code>input_x</code> 进行最大值更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_min</code>	根据指定索引和更新值对 <code>input_x</code> 进行最小值更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_mul</code>	根据指定索引和更新值对 <code>input_x</code> 进行乘法更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_nd_add</code>	根据指定索引和更新值对 <code>input_x</code> 进行稀疏加法更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_nd_div</code>	根据指定索引和更新值对 <code>input_x</code> 进行稀疏除法更新。	GPU CPU
<code>mindspore.ops.scatter_nd_max</code>	根据指定索引和更新值对 <code>input_x</code> 进行稀疏最大值更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_nd_min</code>	根据指定索引和更新值对 <code>input_x</code> 进行稀疏最小值更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_nd_mul</code>	根据指定索引和更新值对 <code>input_x</code> 进行稀疏乘法更新。	GPU CPU
<code>mindspore.ops.scatter_nd_sub</code>	根据指定索引和更新值对 <code>input_x</code> 进行稀疏减法更新。	Ascend GPU CPU
<code>mindspore.ops.scatter_update</code>	根据指定输入索引和更新值更新输入 <code>tensor</code> 的值。	Ascend GPU CPU

2.4.1 mindspore.ops.assign

`mindspore.ops.assign(variable, value)`

为网络参数或者 `tensor` 赋值。

支持隐式类型转换、类型提升。

参数：

- **variable** (`Union[Parameter, Tensor]`) - 输入的网络参数或者 `tensor`。
- **value** (`Tensor`) - 要分配的值，`shape` 与 `variable` 相同。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> value = mindspore.tensor([2.0], mindspore.float32)
>>> variable = mindspore.Parameter(mindspore.tensor([1.0], mindspore.float32), name="variable")
>>> mindspore.ops.assign(variable, value)
>>> print(variable.asnumpy())
[2.]
```

2.4.2 mindspore.ops.assign_add

`mindspore.ops.assign_add(variable, value)`

通过加法运算更新网络参数或者 `tensor`。

支持隐式类型转换、类型提升。

参数：

- **variable** (`Union[Parameter, Tensor]`) - 输入的网络参数或者 `tensor`。
- **value** (`Union[Tensor, Number]`) - 要和 `variable` 相加的值。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> variable = mindspore.Parameter([1], name="global_step")
>>> value = mindspore.tensor([100], dtype=mindspore.int32)
>>> mindspore.ops.assign_add(variable, value)
>>> print(variable.asnumpy())
[101]
```

2.4.3 mindspore.ops.assign_sub

`mindspore.ops.assign_sub(variable, value)`

通过减法运算更新网络参数或者 `tensor`。

支持隐式类型转换、类型提升。

参数：

- **variable** (`Union[Parameter, Tensor]`) - 输入的网络参数或者 `tensor`。
- **value** (`Tensor`) - 从 *variable* 减去的值。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> variable = mindspore.Parameter([1], name="global_step")
>>> value = mindspore.tensor([100], dtype=mindspore.int32)
>>> mindspore.ops.assign_sub(variable, value)
>>> print(variable.asnumpy())
[-99]
```

2.4.4 mindspore.ops.scatter_add

`mindspore.ops.scatter_add(input_x, indices, updates)`

根据指定索引和更新值对 *input_x* 进行加法更新。

$$\text{input\_x}[\text{indices}[i, \dots, j], :] += \text{updates}[i, \dots, j, :]$$

说明：

- 支持隐式类型转换、类型提升。
- 因 `Parameter` 对象不支持类型转换，当 *input_x* 为低精度数据类型时，会抛出异常。
- *updates* 的 `shape` 为 *indices.shape* + *input_x.shape[1:]*。

参数：

- **input_x** (`Union[Parameter, Tensor]`) - 输入的 `parameter` 或 `tensor`。
- **indices** (`Tensor`) - 指定索引。
- **updates** (`Tensor`) - 更新值。

返回：

`Tensor`

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]],
...                                              mindspore.float32), name="x")
>>> indices = mindspore.tensor([[0, 1], [1, 1]], mindspore.int32)
>>> updates = mindspore.tensor([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]], mindspore.float32)
>>> output = mindspore.ops.scatter_add(input_x, indices, updates)
>>> print(output)
[[ 1.  1.  1.]
 [19. 19. 19.]]
```

2.4.5 mindspore.ops.scatter_div

`mindspore.ops.scatter_div(input_x, indices, updates)`

根据指定索引和更新值对 `input_x` 进行除法更新。

$$\text{input\_x}[\text{indices}[i, \dots, j], :] /= \text{updates}[i, \dots, j, :]$$

说明:

- 支持隐式类型转换、类型提升。
- 因 `Parameter` 对象不支持类型转换, 当 `input_x` 为低精度数据类型时, 会抛出异常。
- `updates` 的 `shape` 为 `indices.shape + input_x.shape[1:]`。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([[6.0, 6.0, 6.0], [2.0, 2.0, 2.0]],
...                                              mindspore.float32), name="x")
>>> indices = mindspore.tensor([0, 1], mindspore.int32)
>>> updates = mindspore.tensor([[2.0, 2.0, 2.0], [2.0, 2.0, 2.0]], mindspore.float32)
>>> output = mindspore.ops.scatter_div(input_x, indices, updates)
>>> print(output)
[[3. 3. 3.]
 [1. 1. 1.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
    ↪ initialized.
```

(下页继续)

(续上页)

```

>>> input_x = mindspore.Parameter(mindspore.tensor([[105.0, 105.0, 105.0],
...                                                    [315.0, 315.0, 315.0]]), mindspore.
↳float32), name="x")
>>> # for indices = [[0, 1], [1, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [105.0, 105.0, 105.0] / [1.0, 1.0, 1.0] = [105.0, 105.0, 105.0]
>>> # input_x[1] = [315.0, 315.0, 315.0] / [3.0, 3.0, 3.0] = [105.0, 105.0, 105.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [105.0, 105.0, 105.0] / [5.0, 5.0, 5.0] = [21.0, 21.0, 21.0]
>>> # input_x[1] = [21.0, 21.0, 21.0] / [7.0, 7.0, 7.0] = [3.0, 3.0, 3.0]
>>> indices = mindspore.tensor([[0, 1], [1, 1]], mindspore.int32)
>>> updates = mindspore.tensor([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[5.0, 5.0, 5.0], [7.0, 7.0, 7.0]]], mindspore.float32)
>>> output = mindspore.ops.scatter_div(input_x, indices, updates)
>>> print(output)
[[105. 105. 105.]
 [ 3.   3.   3.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳initialized.
>>> input_x = mindspore.Parameter(mindspore.tensor([[105.0, 105.0, 105.0],
...                                                    [315.0, 315.0, 315.0]]), mindspore.
↳float32), name="x")
>>> # for indices = [[1, 0], [1, 1]]
>>> # step 1: [1, 0]
>>> # input_x[0] = [105.0, 105.0, 105.0] / [3.0, 3.0, 3.0] = [35.0, 35.0, 35.0]
>>> # input_x[1] = [315.0, 315.0, 315.0] / [1.0, 1.0, 1.0] = [315.0, 315.0, 315.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [315.0, 315.0, 315.0] / [5.0, 5.0, 5.0] = [63.0 63.0 63.0]
>>> # input_x[1] = [63.0 63.0 63.0] / [7.0, 7.0, 7.0] = [9.0, 9.0, 9.0]
>>> indices = mindspore.tensor([[1, 0], [1, 1]], mindspore.int32)
>>> updates = mindspore.tensor([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[5.0, 5.0, 5.0], [7.0, 7.0, 7.0]]], mindspore.float32)
>>> output = mindspore.ops.scatter_div(input_x, indices, updates)
>>> print(output)
[[35. 35. 35.]
 [ 9.  9.  9.]]

```

2.4.6 mindspore.ops.scatter_max

`mindspore.ops.scatter_max(input_x, indices, updates)`

根据指定索引和更新值对 `input_x` 进行最大值更新。

$$\text{input\_x}[\text{indices}[i, \dots, j], :] = \max(\text{input\_x}[\text{indices}[i, \dots, j], :], \text{updates}[i, \dots, j, :])$$

说明:

- 支持隐式类型转换、类型提升。
- 因 `Parameter` 对象不支持类型转换, 当 `input_x` 为低精度数据类型时, 会抛出异常。
- `updates` 的 `shape` 为 `indices.shape + input_x.shape[1:]`。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]],
...                                              mindspore.float32), name="input_x")
>>> indices = mindspore.tensor([[0, 0], [1, 1]], mindspore.int32)
>>> updates = mindspore.tensor(mindspore.ops.ones([2, 2, 3]) * 88, mindspore.float32)
>>> output = mindspore.ops.scatter_max(input_x, indices, updates)
>>> print(output)
[[88. 88. 88.]
 [88. 88. 88.]
```

2.4.7 mindspore.ops.scatter_min

`mindspore.ops.scatter_min(input_x, indices, updates)`

根据指定索引和更新值对 *input_x* 进行最小值更新。

$$\text{input\_x}[\text{indices}[i, \dots, j], :] = \min(\text{input\_x}[\text{indices}[i, \dots, j], :], \text{updates}[i, \dots, j, :])$$

说明：

- 支持隐式类型转换、类型提升。
 - 因 Parameter 对象不支持类型转换，当 *input_x* 为低精度数据类型时，会抛出异常。
 - *updates* 的 shape 为 *indices.shape* + *input_x.shape[1:]* 。
-

参数：

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.zeros((2, 3)),
...                                              mindspore.float32), name="input_x")
>>> indices = mindspore.tensor([1, 0], mindspore.int32)
>>> update = mindspore.tensor(mindspore.ops.arange(0, 6).reshape((2, 3)), mindspore.float32)
>>> output = mindspore.ops.scatter_min(input_x, indices, update)
>>> print(output)
[[0. 0. 0.]
 [0. 0. 0.]]
```

2.4.8 mindspore.ops.scatter_mul

`mindspore.ops.scatter_mul(input_x, indices, updates)`

根据指定索引和更新值对 `input_x` 进行乘法更新。

$$\text{input\_x}[\text{indices}[i, \dots, j, :]] *= \text{updates}[i, \dots, j, :]$$

说明:

- 支持隐式类型转换、类型提升。
- 因 `Parameter` 对象不支持类型转换，当 `input_x` 为低精度数据类型时，会抛出异常。
- `updates` 的 `shape` 为 `indices.shape + input_x.shape[1:]`。

参数:

- **input_x** (`Union[Parameter, Tensor]`) - 输入的 parameter 或 tensor。
- **indices** (`Tensor`) - 指定索引。
- **updates** (`Tensor`) - 更新值。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([[1.0, 1.0, 1.0],
...                                              [2.0, 2.0, 2.0]], mindspore.float32), name="x")
>>> indices = mindspore.tensor([0, 1], mindspore.int32)
>>> updates = mindspore.tensor([[2.0, 2.0, 2.0], [2.0, 2.0, 2.0]], mindspore.float32)
>>> output = mindspore.ops.scatter_mul(input_x, indices, updates)
>>> print(output)
[[2. 2. 2.]
 [4. 4. 4.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
    ↪ initialized.
>>> input_x = mindspore.Parameter(mindspore.tensor([[1.0, 1.0, 1.0],
...                                              [2.0, 2.0, 2.0]], mindspore.float32),
    ↪ name="x")
```

(下页继续)

(续上页)

```

>>> # for indices = [[0, 1], [1, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [1.0, 1.0, 1.0] = [1.0, 1.0, 1.0]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [3.0, 3.0, 3.0] = [6.0, 6.0, 6.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [6.0, 6.0, 6.0] * [7.0, 7.0, 7.0] = [42.0, 42.0, 42.0]
>>> # input_x[1] = [42.0, 42.0, 42.0] * [9.0, 9.0, 9.0] = [378.0, 378.0, 378.0]
>>> indices = mindspore.tensor([[0, 1], [1, 1]], mindspore.int32)
>>> updates = mindspore.tensor([[[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]], mindspore.float32)
>>> output = mindspore.ops.scatter_mul(input_x, indices, updates)
>>> print(output)
[[ 1.  1.  1.]
 [378. 378. 378.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = mindspore.Parameter(mindspore.tensor([[1.0, 1.0, 1.0],
...                                                  [2.0, 2.0, 2.0]], mindspore.float32),
↳ name="x")
>>> # for indices = [[1, 0], [1, 1]]
>>> # step 1: [1, 0]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [3.0, 3.0, 3.0] = [3.0, 3.0, 3.0]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [1.0, 1.0, 1.0] = [2.0, 2.0, 2.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [7.0, 7.0, 7.0] = [14.0, 14.0, 14.0]
>>> # input_x[1] = [14.0, 14.0, 14.0] * [9.0, 9.0, 9.0] = [126.0, 126.0, 126.0]
>>> indices = mindspore.tensor([[1, 0], [1, 1]], mindspore.int32)
>>> updates = mindspore.tensor([[[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]], mindspore.float32)
>>> output = mindspore.ops.scatter_mul(input_x, indices, updates)
>>> print(output)
[[ 3.  3.  3.]
 [126. 126. 126.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = mindspore.Parameter(mindspore.tensor([[1.0, 1.0, 1.0],
...                                                  [2.0, 2.0, 2.0]], mindspore.float32),
↳ name="x")
>>> # for indices = [[0, 1], [0, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [1.0, 1.0, 1.0] = [1.0, 1.0, 1.0]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [3.0, 3.0, 3.0] = [6.0, 6.0, 6.0]
>>> # step 2: [0, 1]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [7.0, 7.0, 7.0] = [7.0, 7.0, 7.0]
>>> # input_x[1] = [6.0, 6.0, 6.0] * [9.0, 9.0, 9.0] = [54.0, 54.0, 54.0]
>>> indices = mindspore.tensor([[0, 1], [0, 1]], mindspore.int32)
>>> updates = mindspore.tensor([[[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]], mindspore.float32)
>>> output = mindspore.ops.scatter_mul(input_x, indices, updates)
>>> print(output)
[[ 7.  7.  7.]
 [54. 54. 54.]]

```

2.4.9 mindspore.ops.scatter_nd_add

`mindspore.ops.scatter_nd_add(input_x, indices, updates, use_locking=False)`

根据指定索引和更新值对 `input_x` 进行稀疏加法更新。

$$\text{input\_x}[\text{indices}[i, \dots, j]] += \text{updates}[i, \dots, j]$$

说明:

- 支持隐式类型转换、类型提升。
- `indices` 的维度至少为 2，并且 `indices.shape[-1] <= len(indices.shape)`。
- `updates` 的 shape 为 `indices.shape[:-1] + input_x.shape[indices.shape[-1]:]`。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。
- **use_locking** (bool, 可选) - 是否启用锁保护。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([1, 2, 3, 4, 5, 6, 7, 8],
...                                              mindspore.float32), name="x")
>>> indices = mindspore.tensor([[2], [4], [1], [7]], mindspore.int32)
>>> updates = mindspore.tensor([6, 7, 8, 9], mindspore.float32)
>>> output = mindspore.ops.scatter_nd_add(input_x, indices, updates, False)
>>> print(output)
[ 1. 10.  9.  4. 12.  6.  7. 17.]
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.zeros((4, 4, 4)), mindspore.
↪int32))
>>> indices = mindspore.tensor([[0], [2]], mindspore.int32)
>>> updates = mindspore.tensor([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]],
↪mindspore.int32)
>>> output = mindspore.ops.scatter_nd_add(input_x, indices, updates, False)
>>> print(output)
[[[1 1 1 1]
  [2 2 2 2]
  [3 3 3 3]
  [4 4 4 4]]
 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]
```

(下页继续)

(续上页)

```
[0 0 0 0]]
[[5 5 5 5]
 [6 6 6 6]
 [7 7 7 7]
 [8 8 8 8]]
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]]
```

2.4.10 mindspore.ops.scatter_nd_div

`mindspore.ops.scatter_nd_div(input_x, indices, updates, use_locking=False)`

根据指定索引和更新值对 `input_x` 进行稀疏除法更新。

$$\text{input\_x}[\text{indices}[i, \dots, j]] \neq \text{updates}[i, \dots, j]$$

说明:

- 支持隐式类型转换、类型提升。
- `indices` 的维度至少为 2, 并且 `indices.shape[-1] <= len(indices.shape)`。
- `updates` 的 shape 为 `indices.shape[:-1] + input_x.shape[indices.shape[-1]:]`。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。
- **use_locking** (bool) - 是否启用锁保护。默认 False。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([1, 2, 3, 4, 5, 6, 7, 8],
...                                                mindspore.float32), name="x")
>>> indices = mindspore.tensor([[2], [4], [1], [7]], mindspore.int32)
>>> updates = mindspore.tensor([6, 7, 8, 9], mindspore.float32)
>>> output = mindspore.ops.scatter_nd_div(input_x, indices, updates, False)
>>> print(output)
[1.          0.25          0.5          4.          0.71428573  6.
 7.          0.88888889 ]
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.ones((4, 4, 4)), mindspore.
↪float32))
```

(下页继续)

(续上页)

```

>>> indices = mindspore.tensor([[0], [2]], mindspore.int32)
>>> updates = mindspore.tensor([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]],
↳mindspore.float32)
>>> output = mindspore.ops.scatter_nd_div(input_x, indices, updates, False)
>>> print(output)
[[[1.          1.          1.          1.          ]
  [0.5         0.5         0.5         0.5         ]
  [0.33333334 0.33333334 0.33333334 0.33333334]
  [0.25        0.25        0.25        0.25        ]]
 [[1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]]
 [[0.2         0.2         0.2         0.2         ]
  [0.16666667 0.16666667 0.16666667 0.16666667]
  [0.14285715 0.14285715 0.14285715 0.14285715]
  [0.125        0.125        0.125        0.125        ]]
 [[1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]]]

```

2.4.11 mindspore.ops.scatter_nd_max

`mindspore.ops.scatter_nd_max(input_x, indices, updates, use_locking=False)`

根据指定索引和更新值对 `input_x` 进行稀疏最大值更新。

$$\text{input\_x}[\text{indices}[i, \dots, j]] = \max(\text{input\_x}[\text{indices}[i, \dots, j]], \text{updates}[i, \dots, j])$$

说明:

- 支持隐式类型转换、类型提升。
- `indices` 的维度至少为 2，并且 `indices.shape[-1] <= len(indices.shape)`。
- `updates` 的 `shape` 为 `indices.shape[:-1] + input_x.shape[indices.shape[-1]:]`。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。
- **use_locking** (bool) - 是否启用锁保护。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([1, 2, 3, 4, 5, 6, 7, 8],
...                                              mindspore.float32), name="x")
>>> indices = mindspore.tensor([[2], [4], [1], [7]], mindspore.int32)
>>> updates = mindspore.tensor([6, 7, 8, 9], mindspore.float32)
>>> output = mindspore.ops.scatter_nd_max(input_x, indices, updates, False)
>>> print(output)
[1. 8. 6. 4. 7. 6. 7. 9.]
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.ones((4, 4, 4)), mindspore.
->int32))
>>> indices = mindspore.tensor([[0], [2]], mindspore.int32)
>>> updates = mindspore.tensor([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]],
->mindspore.int32)
>>> output = mindspore.ops.scatter_nd_max(input_x, indices, updates, False)
>>> print(output)
[[[1 1 1 1]
  [2 2 2 2]
  [3 3 3 3]
  [4 4 4 4]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]
 [[5 5 5 5]
  [6 6 6 6]
  [7 7 7 7]
  [8 8 8 8]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
```

2.4.12 mindspore.ops.scatter_nd_min

`mindspore.ops.scatter_nd_min(input_x, indices, updates, use_locking=False)`

根据指定索引和更新值对 `input_x` 进行稀疏最小值更新。

$$\text{input\_x}[\text{indices}[i, \dots, j]] = \min(\text{input\_x}[\text{indices}[i, \dots, j]], \text{updates}[i, \dots, j])$$

说明:

- 支持隐式类型转换、类型提升。
- `indices` 的维度至少为 2, 并且 `indices.shape[-1] <= len(indices.shape)`。
- `updates` 的 shape 为 `indices.shape[:-1] + input_x.shape[indices.shape[-1]:]`。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。

- **updates** (Tensor) - 更新值。
- **use_locking** (bool) - 是否启用锁保护。默认 `False`。

返回：

Tensor

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.ones(8) * 10, mindspore.
↳float32), name="x")
>>> indices = mindspore.tensor([[2], [4], [1], [7]], mindspore.int32)
>>> updates = mindspore.tensor([6, 7, 8, 9], mindspore.float32)
>>> output = mindspore.ops.scatter_nd_min(input_x, indices, updates, False)
>>> print(output)
[10.  8.  6. 10.  7. 10. 10.  9.]
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.ones((4, 4, 4)) * 10,
↳mindspore.int32))
>>> indices = mindspore.tensor([[0], [2]], mindspore.int32)
>>> updates = mindspore.tensor([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                               [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]],
↳mindspore.int32)
>>> output = mindspore.ops.scatter_nd_min(input_x, indices, updates, False)
>>> print(output)
[[[ 1  1  1  1]
 [ 2  2  2  2]
 [ 3  3  3  3]
 [ 4  4  4  4]]
 [[10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]]
 [[ 5  5  5  5]
 [ 6  6  6  6]
 [ 7  7  7  7]
 [ 8  8  8  8]]
 [[10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]
 [10 10 10 10]]]
```

2.4.13 mindspore.ops.scatter_nd_mul

`mindspore.ops.scatter_nd_mul` (*input_x, indices, updates, use_locking=False*)

根据指定索引和更新值对 *input_x* 进行稀疏乘法更新。

$$\text{input\_x}[\text{indices}[i, \dots, j]] \text{ *= updates}[i, \dots, j]$$

说明：

- 支持隐式类型转换、类型提升。

- *indices* 的维度至少为 2, 并且 *indices.shape[-1] <= len(indices.shape)*。
- *updates* 的 shape 为 *indices.shape[:-1] + input_x.shape[indices.shape[-1]:]*。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。
- **use_locking** (bool) - 是否启用锁保护。默认 False。

返回:

Tensor

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([1, 2, 3, 4, 5, 6, 7, 8],
...                                              mindspore.float32), name="x")
>>> indices = mindspore.tensor([[2], [4], [1], [7]], mindspore.int32)
>>> updates = mindspore.tensor([6, 7, 8, 9], mindspore.float32)
>>> output = mindspore.ops.scatter_nd_mul(input_x, indices, updates)
>>> print(output)
[ 1. 16. 18.  4. 35.  6.  7. 72.]
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.ones((4, 4, 4)), mindspore.
↪int32))
>>> indices = mindspore.tensor([[0], [2]], mindspore.int32)
>>> updates = mindspore.tensor([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]], ↪
↪mindspore.int32)
>>> output = mindspore.ops.scatter_nd_mul(input_x, indices, updates)
>>> print(output)
[[[1 1 1 1]
  [2 2 2 2]
  [3 3 3 3]
  [4 4 4 4]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]
 [[5 5 5 5]
  [6 6 6 6]
  [7 7 7 7]
  [8 8 8 8]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
```

2.4.14 mindspore.ops.scatter_nd_sub

`mindspore.ops.scatter_nd_sub(input_x, indices, updates, use_locking=False)`

根据指定索引和更新值对 `input_x` 进行稀疏减法更新。

$$\text{input\_x}[\text{indices}[i, \dots, j]] -= \text{updates}[i, \dots, j]$$

说明:

- 支持隐式类型转换、类型提升。
- `indices` 的维度至少为 2，并且 `indices.shape[-1] <= len(indices.shape)`。
- `updates` 的 shape 为 `indices.shape[:-1] + input_x.shape[indices.shape[-1]:]`。

参数:

- **input_x** (Union[Parameter, Tensor]) - 输入的 parameter 或 tensor。
- **indices** (Tensor) - 指定索引。
- **updates** (Tensor) - 更新值。
- **use_locking** (bool) - 是否启用锁保护。默认 `False`。

返回:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input_x = mindspore.Parameter(mindspore.tensor([1, 2, 3, 4, 5, 6, 7, 8],
...                                                mindspore.float32), name="x")
>>> indices = mindspore.tensor([[2], [4], [1], [7]], mindspore.int32)
>>> updates = mindspore.tensor([6, 7, 8, 9], mindspore.float32)
>>> output = mindspore.ops.scatter_nd_sub(input_x, indices, updates, False)
>>> print(output)
[ 1. -6. -3.  4. -2.  6.  7. -1.]
>>> input_x = mindspore.Parameter(mindspore.tensor(mindspore.ops.zeros((4, 4, 4)), mindspore.
->int32))
>>> indices = mindspore.tensor([[0], [2]], mindspore.int32)
>>> updates = mindspore.tensor([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]],
->mindspore.int32)
>>> output = mindspore.ops.scatter_nd_sub(input_x, indices, updates, False)
>>> print(output)
[[[-1 -1 -1 -1]
  [-2 -2 -2 -2]
  [-3 -3 -3 -3]
  [-4 -4 -4 -4]]
 [[ 0  0  0  0]
  [ 0  0  0  0]
  [ 0  0  0  0]]]
```

(下页继续)

(续上页)

```
[ 0  0  0  0]
[-5 -5 -5 -5]
[-6 -6 -6 -6]
[-7 -7 -7 -7]
[-8 -8 -8 -8]
[ 0  0  0  0]
[ 0  0  0  0]
[ 0  0  0  0]
[ 0  0  0  0]]]
```

2.4.15 mindspore.ops.scatter_update

`mindspore.ops.scatter_update(input_x, indices, updates)`

根据指定输入索引和更新值更新输入 `tensor` 的值。

说明:

- 支持隐式类型转换、类型提升。
- 因 `Parameter` 对象不支持类型转换，当 `input_x` 为低精度数据类型时，会抛出异常。
- 参数 `updates` 的 `shape` 为 `indices.shape + input_x.shape[1:]`。

若 `indices` 的 `shape` 为 $(i, \dots, j)$ ，则

$$\text{input\_x}[\text{indices}[i, \dots, j], :] = \text{updates}[i, \dots, j, :]$$

参数:

- **input_x** (`Union[Parameter, Tensor]`) - 输入的 `parameter` 或 `tensor`。
- **indices** (`Tensor`) - 更新操作的索引。如果索引中存在重复项，则更新的顺序无法得知。
- **updates** (`Tensor`) - 更新值。

返回:

`Tensor`

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> np_x = mindspore.tensor([[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]], mindspore.float32)
>>> input_x = mindspore.Parameter(np_x, name="x")
>>> indices = mindspore.tensor([0, 1], mindspore.int32)
>>> np_updates = mindspore.tensor([[2.0, 1.2, 1.0], [3.0, 1.2, 1.0]])
>>> updates = mindspore.tensor(np_updates, mindspore.float32)
>>> output = mindspore.ops.scatter_update(input_x, indices, updates)
>>> print(output)
[[2. 1.2 1.]
 [3. 1.2 1.]]
```

2.5 微分函数

接口名	概述	支持平台
<code>mindspore.ops.derivative</code>	计算函数或网络输出对输入的高阶微分。	Ascend GPU CPU
<code>mindspore.ops.jet</code>	计算函数或网络输出对输入的高阶微分。	Ascend GPU CPU
<code>mindspore.ops.stop_gradient</code>	用于消除某个值对梯度的影响，例如截断来自于函数输出的梯度传播。	Ascend GPU CPU

2.5.1 mindspore.ops.derivative

`mindspore.ops.derivative` (*fn*, *primals*, *order*)
计算函数或网络输出对输入的高阶微分。给定待求导函数的原始输入和求导的阶数 *n*，将返回函数输出对输入的第 *n* 阶导数。输入的初始 1 阶导数在内部默认设置为 1，其他阶设置为 0。

说明:

- 若 *primals* 是 int 型的 tensor，会被转化成 float32 格式进行计算。

参数:

- fn** (Union[Cell, function]) - 待求导的函数或网络。
- primals** (Union[Tensor, tuple[Tensor]]) - *fn* 的输入。
- order** (int) - 求导的阶数。

返回:

tuple(out_primals, out_series)

- out_primals** (Union[Tensor, list[Tensor]]) - *fn(primals)* 的结果。
- out_series** (Union[Tensor, list[Tensor]]) - *fn* 输出对输入的第 *n* 阶导数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import nn
>>> mindspore.set_context(mode=mindspore.GRAPH_MODE)
>>> class Net(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.sin = mindspore.ops.Sin()
...         self.exp = mindspore.ops.Exp()
...     def construct(self, x):
...         out1 = self.sin(x)
...         out2 = self.exp(out1)
...         return out2
>>>
>>> primals = mindspore.tensor([[1, 2], [3, 4]], mindspore.float32)
>>> order = 3
```

(下页继续)

(续上页)

```
>>> net = Net()
>>> out_primals, out_series = mindspore.ops.derivative(net, primals, order)
>>> print(out_primals, out_series)
[[2.319777  2.4825778]
 [1.1515628 0.4691642]] [[-4.0515366   3.6724353 ]
 [ 0.5053504  -0.52061415]]
```

2.5.2 mindspore.ops.jet

`mindspore.ops.jet` (*fn, primals, series*)

计算函数或网络输出对输入的高阶微分。给定待求导函数的原始输入和自定义的 1 到 *n* 阶导数，将返回函数输出对输入的第 1 到 *n* 阶导数。一般情况，建议输入的 1 阶导数值为全 1，更高阶的导数值为全 0，这与输入对本身的导数情况是一致的。

说明：

- 若 *primals* 是 `int` 型的 `tensor`，会被转化成 `float32` 格式进行计算。

参数：

- **fn** (`Union[Cell, function]`) - 待求导的函数或网络。
- **primals** (`Union[Tensor, tuple[Tensor]]`) - *fn* 的输入。
- **series** (`Union[Tensor, tuple[Tensor]]`) - 输入的原始第 1 到第 *n* 阶导数。`tensor` 第零维度索引 *i* 对应输出对输入的第 *i+1* 阶导数。

返回：

`tuple(out_primals, out_series)`

- **out_primals** (`Union[Tensor, list[Tensor]]`) - *fn(primals)* 的结果。
- **out_series** (`Union[Tensor, list[Tensor]]`) - *fn* 输出对输入的第 1 到 *n* 阶导数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> mindspore.set_context(mode=mindspore.GRAPH_MODE)
>>> class Net(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.sin = mindspore.ops.Sin()
...         self.exp = mindspore.ops.Exp()
...     def construct(self, x):
...         out1 = self.sin(x)
...         out2 = self.exp(out1)
...         return out2
>>> primals = mindspore.tensor([[1, 2], [3, 4]], mindspore.float32)
>>> series = mindspore.tensor([[[1, 1], [1, 1]], [[0, 0], [0, 0]], [[0, 0], [0, 0]]],
↵mindspore.float32)
```

(下页继续)

(续上页)

```
>>> net = Net()
>>> out_primals, out_series = mindspore.ops.jet(net, primals, series)
>>> print(out_primals, out_series)
[[2.319777  2.4825778]
 [1.1515628 0.4691642]] [[[ 1.2533808 -1.0331168 ]
 [-1.1400385 -0.3066662 ]
 [-1.2748207 -1.8274734 ]
 [ 0.966121  0.55551505]]
 [-4.0515366  3.6724353 ]
 [ 0.5053504 -0.52061415]]]
```

2.5.3 mindspore.ops.stop_gradient

`mindspore.ops.stop_gradient (value)`

用于消除某个值对梯度的影响，例如截断来自于函数输出的梯度传播。更多细节请参考 [Stop Gradient](#)。

参数：

- **value** (Any) - 需要被消除梯度影响的值。

返回：

一个与 *value* 相同的值。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> def f1(x):
...     return x ** 2
>>> x = 3.0
>>> f1(x)
9.0
>>> mindspore.ops.grad(f1)(mindspore.tensor(x))
Tensor(shape=[], dtype=Float32, value= 6)
>>>
>>> # The same function with stop_gradient, return a zero gradient because x is effectively_
    ↳ treated as a constant.
>>> def f2(x):
...     return mindspore.ops.stop_gradient(x) ** 2
>>> f2(x)
9.0
>>> mindspore.ops.grad(f2)(mindspore.tensor(x))
Tensor(shape=[], dtype=Float32, value= 0)
```

2.6 调试函数

接口名	概述	支持平台
<code>mindspore.ops.print_</code>	将输入数据进行打印输出。	Ascend GPU CPU
<code>mindspore.ops.tensor_dump</code>	将 <code>tensor</code> 保存为 <code>numpy</code> 格式的文件。	Ascend

2.6.1 mindspore.ops.print_

`mindspore.ops.print_(*input_x)`

将输入数据进行打印输出。

默认打印在屏幕上。也可以通过 `context` 设置 `print_file_path` 参数保存在指定文件中。通过函数 `mindspore.parse_print()` 可以重新加载数据。获取更多信息, 请查看 `mindspore.set_context()` 和 `mindspore.parse_print()`。

在 Ascend 平台上的 Graph 模式下, 可以通过设置环境变量 `MS_DUMP_SLICE_SIZE` 和 `MS_DUMP_WAIT_TIME` 解决在输出大 `tesnor` 或输出 `tensor` 比较密集场景下算子执行失败的问题。

说明: 在 PyNative 模式下, 请使用 Python `print` 函数。在 Ascend 平台上的 Graph 模式下, `bool`、`int`、`float`、`tuple` 以及 `list` 类型数据将被转换为 `tensor` 进行打印, `str` 保持不变。

参数:

- `input_x` (Union[`Tensor`, `bool`, `int`, `float`, `str`, `tuple`, `list`]) - `print_` 的输入。支持多个输入, 用', '分隔。

返回:

无效返回值, 应忽略。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> x = mindspore.tensor(mindspore.ops.ones([2, 1], mindspore.int32))
>>> y = mindspore.tensor(mindspore.ops.ones([2, 2], mindspore.int32))
>>> result = mindspore.ops.print_('Print Tensor x and Tensor y:', x, y)
Print Tensor x and Tensor y:
Tensor(shape=[2, 1], dtype=Int32, value=
[[1],
 [1]])
Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 1],
 [1, 1]])
```


2.6.2 mindspore.ops.tensordump

`mindspore.ops.tensordump(file_name, tensor, mode='out')`

将 `tensor` 保存为 `numpy` 格式的文件。

警告:

- 参数 `mode` 将不再支持参数值为 `'all'`。

在并行的场景下，该算子会保存不同计算卡上的数据分片。

在昇腾平台的静态图并行模式下，代码算子 `A -> 算子 B` 可能会被编译为算子 `A -> 重排算子 -> 算子 B`。

注：重排算子指在静态图并行场景中，由于设备间通信，算子切分策略而引入的算子。

对于算子 `A -> 算子 B` 的情况，算子 `A` 计算结果的输出等于算子 `B` 的输入。

然而对于算子 `A -> 重排算子 -> 算子 B` 这样的情况，由于重排算子的插入，导致算子 `A` 的输出不再等价于算子 `B` 的输入。

假设一种情况是算子 `A` 的计算结果既作为算子 `B` 的输入，也作为 `tensordump` 算子的输入。则在该情况下，通过设置 `mode` 参数可以实现不同的保存数据的需求：

- 如果 `mode` 参数设置为 `'out'`，保存的数据仅包含算子 `A` 的输出分片。
- 如果 `mode` 参数设置为 `'in'`，保存的数据将仅包含算子 `B` 的输入分片。

当 `mode` 参数被配置为 `'in'` 时，生成的输入分片所对应的 `numpy` 文件命名格式为：`fileName_dumpMode_dtype_id.npy`。

当 `mode` 参数被配置为 `'out'` 时，生成的输出分片所对应的 `numpy` 文件命名格式为：`fileName_dtype_id.npy`。

- `fileName`：参数 `file_name` 的值（若该参数传入时是一个使用者指定的路径，则 `fileName` 值为路径的最后一级）。
- `dumpMode`：参数 `mode` 的值。
- `dtype`：原始的数据类型。`bfloat16` 类型数据保存在 `.npy` 文件中会被转换成 `float32` 类型。
- `id`：一个自增的 ID。

说明:

- 在 Ascend 平台上的 `graph mode` 下，可以通过设置环境变量 `MS_DUMP_SLICE_SIZE` 和 `MS_DUMP_WAIT_TIME` 解决在输出大 `tesnor` 或输出 `tensor` 比较密集场景下算子执行失败的问题。
- 当前该算子不支持在控制流中使用。
- 如果当前的并行模式为 `STAND_ALONE`，参数 `mode` 只能设置为 `'out'`。
- 此函数用于调试。

参数:

- **file_name** (str) - `numpy` 文件的保存路径。
- **tensor** (Tensor) - 输入的张量。
- **mode** (str, 可选) - 控制 `tensordump` 行为模式的参数，可选的值为 `['out', 'in']` 中的一个，默认 `out`。

支持平台:

Ascend

样例:

说明: 使用 `msrun` 命令运行下面的例子: `msrun --worker_num=2 --local_worker_num=2 --master_port=11450 --log_dir=msrun_log --join=True --cluster_time_out=300 tensordump_example.py`

```
>>> import os
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, context
>>> from mindspore.communication import init, get_rank
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> from mindspore.nn.utils import no_init_parameters
>>> init()
>>> rank_id = get_rank()
>>> dump_path = f'rank_{rank_id}_mul1_mul2.npy'
>>> class Net(nn.Cell):
...     def __init__(self, strategy1, strategy2):
...         super(Net, self).__init__()
...         self.matmul1 = mindspore.ops.MatMul().shard(strategy1)
...         self.matmul2 = mindspore.ops.MatMul().shard(strategy2)
...
...     def construct(self, x, y, b):
...         out1 = self.matmul1(x, y)
...         mindspore.ops.tensordump(dump_path, out1, 'out')
...         out2 = self.matmul2(out1, b)
...         return out2
...
>>> mindspore.set_context(mode=mindspore.GRAPH_MODE)
>>> os.environ["MS_DEV_SAVE_GRAPHS"] = "2"
>>> strategy1 = ((1, 2), (2, 1))
>>> strategy2 = ((1, 2), (2, 1))
>>> with no_init_parameters():
>>>     net = Net(strategy1, strategy2)
>>> x = mindspore.tensor(0.1 * mindspore.ops.randn(64, 64), mindspore.float32)
>>> y = mindspore.tensor(0.1 * mindspore.ops.randn(64, 64), mindspore.float32)
>>> b = mindspore.tensor(0.1 * mindspore.ops.randn(64, 64), mindspore.float32)
>>> parallel_net = Autoparallel(net, parallel_mode="semi_auto")
>>> parallel_net.dataset_strategy(config="full_batch")
>>> out = parallel_net(x, y, b)
>>> print(f"out shape is: {out.shape}")
>>> # out shape is (64, 64)
>>> matmul1_output_slice = np.load(f'rank_{rank_id}_mul1_mul2_float32_0.npy') # load
↳matmul1's output slice
>>> print(f"matmul1_output_slice is loaded, shape is: {matmul1_output_slice.shape}")
>>> # matmul1_output_slice is loaded, shape is: (64, 64)
```

2.7 稀疏函数

警告：这些是实验性 API，后续可能修改或删除。

接口名	概述	支持平台
<code>mindspore.ops.dense_to_sparse_coo</code>	将常规 Tensor 转换为稀疏化的 COOTensor。	GPU
<code>mindspore.ops.dense_to_sparse_csr</code>	将常规 Tensor 转为稀疏化的 CSRTensor。	GPU
<code>mindspore.ops.csr_to_coo</code>	将一个 CSRTensor 转化成一个 COOTensor。	Ascend GPU CPU

2.7.1 mindspore.ops.dense_to_sparse_coo

`mindspore.ops.dense_to_sparse_coo` (*tensor*: Tensor)
将常规 Tensor 转换为稀疏化的 COOTensor。

说明：目前只支持二维 Tensor。

参数：

- **tensor** (Tensor) - 一个稠密 Tensor，必须是二维。

返回：

返回一个二维的 COOTensor，是原稠密 Tensor 的稀疏化表示。分为：

- **indices** (Tensor) - 二维整数张量，表示稀疏张量中 *values* 所处的位置索引。
- **values** (Tensor) - 一维张量，用来给 *indices* 中的每个元素提供数值。
- **shape** (tuple(int)) - 整数元组，用来指定稀疏矩阵的稠密形状。

异常：

- **TypeError** - *tensor* 不是 Tensor。
- **ValueError** - *tensor* 不是二维 Tensor。

支持平台：

GPU

样例：

```
>>> from mindspore import Tensor, ops
>>> import mindspore as ms
>>> x = Tensor([[1, 0], [-5, 0]], ms.float32)
>>> output = ops.dense_to_sparse_coo(x)
>>> print(output.indices)
[[0 0]
 [1 0]]
>>> print(output.values)
[ 1. -5.]
>>> print(output.shape)
(2, 2)
```

2.7.2 mindspore.ops.dense_to_sparse_csr

`mindspore.ops.dense_to_sparse_csr` (*tensor*: [Tensor](#))

将常规 `Tensor` 转为稀疏化的 `CSRTensor`。

说明：现在只支持二维 `Tensor`。

参数：

- **tensor** (`Tensor`) - 一个稠密 `Tensor`，必须是二维。

返回：

返回一个二维的 `CSRTensor`，是原稠密 `Tensor` 的稀疏化表示。其中数据分别为：

- **indptr** (`Tensor`) - 一维整数张量，表示每行非零元素的在 *values* 中存储的起止位置。
- **indices** (`Tensor`) - 一维整数张量，表示每个元素的列索引值。
- **values** (`Tensor`) - 一维张量，用来表示索引对应的数值。
- **shape** (`tuple(int)`) - 整数元组，用来指定稀疏矩阵的稠密形状。

异常：

- **TypeError** - *tensor* 不是 `Tensor`。
- **ValueError** - *tensor* 不是二维 `Tensor`。

支持平台：

GPU

样例：

```
>>> from mindspore import Tensor, ops
>>> import mindspore as ms
>>> x = Tensor([[1, 0], [-5, 0]], ms.float32)
>>> output = ops.dense_to_sparse_csr(x)
>>> print(output.indptr)
[0 1 2]
>>> print(output.indices)
[0 0]
>>> print(output.shape)
(2, 2)
```

2.7.3 mindspore.ops.csr_to_coo

`mindspore.ops.csr_to_coo` (*tensor*: [CSRTensor](#))

将一个 `CSRTensor` 转化成 `COOTensor`。

说明：现在只支持二维 `CSRTensor`。

参数：

- **tensor** (`CSRTensor`) - 一个 CSR 矩阵，必须是二维。

返回：

返回一个二维的 COOTensor，是原 COOTensor 的 CSR 格式表示。

异常：

- **TypeError** - *tensor* 不是 CSRTensor。
- **ValueError** - *tensor* 不是二维 CSRTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, CSRTensor
>>> indptr = Tensor([0, 1, 2]).astype("int32")
>>> indices = Tensor([0, 1]).astype("int32")
>>> values = Tensor([2, 1]).astype("float32")
>>> shape = (2, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_to_coo(x)
>>> print(output.indices)
[[0 0]
 [1 1]]
```

2.7.4 COO 函数

警告：这些是实验性 API，后续可能修改或删除。

接口名	概述	支持平台
<code>mindspore.ops.coo_abs</code>	逐元素计算输入 COOTensor 的绝对值。	Ascend GPU CPU
<code>mindspore.ops.coo_acos</code>	逐元素计算输入 COOTensor 的反余弦。	Ascend GPU CPU
<code>mindspore.ops.coo_acosh</code>	逐元素计算输入 COOTensor 的反双曲余弦。	Ascend GPU CPU
<code>mindspore.ops.coo_add</code>	两个 COOTensor 相加，根据相加的结果与 <i>thresh</i> 返回新的 COOTensor。	GPU CPU
<code>mindspore.ops.coo_asin</code>	逐元素计算输入 COOTensor 的反正弦。	Ascend GPU CPU
<code>mindspore.ops.coo_asinh</code>	计算 COOTensor 输入元素的反双曲正弦。	Ascend GPU CPU
<code>mindspore.ops.coo_atan</code>	逐元素计算输入 COOTensor 的反正切值。	Ascend GPU CPU
<code>mindspore.ops.coo_atanh</code>	逐元素计算输入 COOTensor 的反双曲正切值。	Ascend GPU CPU
<code>mindspore.ops.coo_ceil</code>	COOTensor 向上取整函数。	Ascend GPU CPU
<code>mindspore.ops.coo_concat</code>	根据指定的轴 <i>concat_dim</i> 对输入的 COO Tensor (<i>sp_input</i>) 进行合并操作。	CPU
<code>mindspore.ops.coo_cos</code>	逐元素计算 COOTensor 输入的余弦。	Ascend GPU CPU
<code>mindspore.ops.coo_cosh</code>	逐元素计算 COOTensor <i>x</i> 的双曲余弦值。	Ascend GPU CPU
<code>mindspore.ops.coo_exp</code>	逐元素计算 COOTensor <i>x</i> 的指数。	Ascend GPU CPU
<code>mindspore.ops.coo_expm1</code>	逐元素计算输入 COOTensor 的指数，然后减去 1。	Ascend GPU CPU
<code>mindspore.ops.coo_floor</code>	COOTensor 逐元素向下取整函数。	Ascend GPU CPU
<code>mindspore.ops.coo_inv</code>	逐元素计算输入 COOTensor 的倒数。	Ascend GPU CPU

下页继续

表 22 – 续上页

<code>mindspore.ops.coo_isfinite</code>	判断 COOTensor 输入数据每个位置上的值是否是有限数。	Ascend GPU CPU
<code>mindspore.ops.coo_isinf</code>	确定输入 COOTensor 每个位置上的元素是否为正负无穷大。	GPU CPU
<code>mindspore.ops.coo_isnan</code>	判断 COOTensor 输入数据每个位置上的值是否是 Nan。	Ascend GPU CPU
<code>mindspore.ops.coo_log</code>	逐元素返回 COOTensor 的自然对数。	Ascend GPU CPU
<code>mindspore.ops.coo_log1p</code>	对输入 COOTensor 逐元素加一后计算自然对数。	Ascend GPU CPU
<code>mindspore.ops.coo_neg</code>	计算输入 COOTensor 的相反数并返回。	Ascend GPU CPU
<code>mindspore.ops.coo_relu</code>	对输入的 COOTensor 逐元素计算其应用 ReLU 激活函数后的值。	Ascend GPU CPU
<code>mindspore.ops.coo_relu6</code>	对输入的 COOTensor 逐元素计算其应用 ReLU 激活函数后的值，上限为 6。	Ascend GPU CPU
<code>mindspore.ops.coo_round</code>	对 COOTensor 输入数据进行四舍五入到最近的整数数值。	Ascend GPU CPU
<code>mindspore.ops.coo_sigmoid</code>	Sigmoid 激活函数，对 COOTensor 逐元素计算 Sigmoid 激活函数。	Ascend GPU CPU
<code>mindspore.ops.coo_sin</code>	逐元素计算输入 COOTensor 的正弦。	Ascend GPU CPU
<code>mindspore.ops.coo_sinh</code>	逐元素计算输入 COOTensor 的双曲正弦。	Ascend GPU CPU
<code>mindspore.ops.coo_softsign</code>	COOTensor Softsign 激活函数。	Ascend GPU CPU
<code>mindspore.ops.coo_sqrt</code>	逐元素计算 COOTensor 的平方根。	Ascend GPU CPU
<code>mindspore.ops.coo_square</code>	逐元素返回 COOTensor 的平方。	Ascend GPU CPU
<code>mindspore.ops.coo_tan</code>	计算 COOTensor 输入元素的正切值。	Ascend GPU CPU
<code>mindspore.ops.coo_tanh</code>	按元素计算 COOTensor 输入元素的双曲正切。	Ascend GPU CPU

mindspore.ops.coo_abs

mindspore.ops.coo_abs (x: COOTensor)
逐元素计算输入 COOTensor 的绝对值。

$$out_i = |x_i|$$

参数：

- **x** (COOTensor) - 输入 COOTensor。

返回：

COOTensor，shape 与 x 相同。

异常：

- **TypeError** - 如果 x 不是 COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
```

(下页继续)

(续上页)

```
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_abs(x)
>>> print(output.values)
[1. 2.]
```

mindspore.ops.coo_acos

mindspore.ops.coo_acos(x: COOTensor)

逐元素计算输入 COOTensor 的反余弦。

$$out_i = \cos^{-1}(x_i)$$

参数:

- **x** (COOTensor) - 输入 COOTensor。

返回:

COOTensor, shape 和数据类型与 *x* 相同。

异常:

- **TypeError** - 如果 *x* 不是 COOTensor。
- **TypeError** - 如果 *x* 的数据类型不是 float16、float32 或 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_acos(x)
>>> print(output.values)
[3.1415927      nan]
```

mindspore.ops.coo_acosh

mindspore.ops.coo_acosh(x: COOTensor)

逐元素计算输入 COOTensor 的反双曲余弦。

$$y_i = \cosh^{-1}(x_i)$$

警告: 给定一个输入 COOTensor *x*, 该函数计算每个元素的反双曲余弦。输入范围为 [1, inf]。

参数:

- **x** (COOTensor) - 需要计算反双曲余弦函数的输入 COOTensor。

返回：

COOTensor，数据类型与 *x* 相同。

异常：

- **TypeError** - *x* 不是 COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_acosh(x)
>>> print(output.values)
[      nan 1.316958]
```

mindspore.ops.coo_add

mindspore.ops.coo_add (*x1*: COOTensor, *x2*: COOTensor, *thresh*: Tensor)

两个 COOTensor 相加，根据相加的结果与 *thresh* 返回新的 COOTensor。

参数：

- **x1** (COOTensor) - 一个操作数，与当前操作数相加。
- **x2** (COOTensor) - 另一个操作数，与当前操作数相加。
- **thresh** (Tensor) - 零维 Tensor，其值作为阈值，用来决定相加后的结果 Tensor 中的哪些 value/index 对在最终返回值中出现。如果结果中 value 的数据类型为实数，则 *thresh* 的数据类型应该与它的数据类型一致。如果结果中 value 小于 *thresh*，它将会被丢掉。

返回：

COOTensor，为两 COOTensor 相加后的结果。

异常：

- **ValueError** - 如果操作数 (*x1*/*x2*) 的 indices 的维度不等于 2。
- **ValueError** - 如果操作数 (*x1*/*x2*) 的 values 的维度不等于 1。
- **ValueError** - 如果操作数 (*x1*/*x2*) 的 shape 的维度不等于 1。
- **ValueError** - 如果 *thresh* 的维度不等于 0。
- **TypeError** - 如果操作数 (*x1*/*x2*) 的 indices 的数据类型不为 int64。
- **TypeError** - 如果操作数 (*x1*/*x2*) 的 shape 的数据类型不为 int64。
- **ValueError** - 如果操作数 (*x1*/*x2*) 的 indices 的长度不等于它的 values 的长度。
- **TypeError** - 如果操作数 (*x1*/*x2*) 的 values 的数据类型不为 (int8/int16/int32/int64/float32/float64/complex64/complex128) 中的任何一个。

- **TypeError** - 如果 thresh 的数据类型不为 (int8/int16/int32/int64/float32/float64) 中的任何一个。
- **TypeError** - 如果操作数 x1 的 indices 数据类型不等于 x2 的 indices 数据类型。
- **TypeError** - 如果操作数 x1 的 values 数据类型不等于 x2 的 values 数据类型。
- **TypeError** - 如果操作数 x1 的 shape 数据类型不等于 x2 的 shape 数据类型。
- **TypeError** - 如果操作数 (x1/x2) 的 values 的数据类型与 thresh 数据类型不匹配。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import Tensor, COOTensor
>>> from mindspore import dtype as mstype
>>> from mindspore import context
>>> from mindspore import ops
>>> indic0 = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values0 = Tensor([1, 2], dtype=mstype.int32)
>>> shape0 = (3, 4)
>>> input0 = COOTensor(indic0, values0, shape0)
>>> indic1 = Tensor([[0, 0], [1, 1]], dtype=mstype.int64)
>>> values1 = Tensor([3, 4], dtype=mstype.int32)
>>> shape1 = (3, 4)
>>> input1 = COOTensor(indic1, values1, shape1)
>>> thres = Tensor(0, dtype=mstype.int32)
>>> out = ops.coo_add(input0, input1, thres)
>>> print(out)
COOTensor(shape=[3, 4], dtype=Int32, indices=Tensor(shape=[4, 2], dtype=Int64, value=
[[0 0]
 [0 1]
 [1 1]
 [1 2]]), values=Tensor(shape=[4], dtype=Int32, value=[3 1 4 2]))
```

mindspore.ops.coo_asin

mindspore.ops.coo_asin (x: COOTensor)

逐元素计算输入 COOTensor 的反正弦。

$$out_i = \sin^{-1}(x_i)$$

参数:

- **x** (COOTensor) - 输入的 COOTensor。COOTensor 的 shape 为 (N,*), 其中 * 表示任意数量的附加维度。数据类型应该是以下类型之一: float16、float32、float64、complex64、complex128。

返回:

COOTensor, 数据类型和 shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 COOTensor。
- **TypeError** - 如果 x 的数据类型不是 float16、float32、float64、complex64、complex128。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_asinh(x)
>>> print(output.values)
[-1.5707964      nan]
```

mindspore.ops.coo_asinh

mindspore.ops.coo_asinh (x: COOTensor)

计算 COOTensor 输入元素的反双曲正弦。

$$out_i = \sinh^{-1}(input_i)$$

参数:

- **x** (COOTensor) - 需要计算反双曲正弦函数的输入。

返回:

COOTensor, 数据类型和 shape 与 *x* 相同。

异常:

- **TypeError** - 如果 *x* 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_asinh(x)
>>> print(output.values)
[-0.8813736  1.4436355]
```

mindspore.ops.coo_atan

mindspore.ops.coo_atan (x: COOTensor)

逐元素计算输入 COOTensor 的反正切值。

$$out_i = \tan^{-1}(x_i)$$

参数:

- **x** (COOTensor) - COOTensor 的输入, 数据类型支持: float16、float32。

返回:

COOTensor 的数据类型与输入相同。

异常：

- **TypeError** - 如果 x 不是 COOTensor。
- **TypeError** - 如果 x 的数据类型不是 float16 或 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_atanh(x)
>>> print(output.values)
[-0.7853982  1.1071488]
```

mindspore.ops.coo_atanh

mindspore.ops.coo_atanh(x : COOTensor)

逐元素计算输入 COOTensor 的反双曲正切值。

$$out_i = \tanh^{-1}(x_i)$$

警告：这是一个实验性 API，后续可能会修改或删除。

参数：

- **x** (COOTensor) - 输入 COOTensor。数据类型支持：float16、float32。

返回：

COOTensor 的数据类型与输入相同。

异常：

- **TypeError** - 如果 x 不是 COOTensor。
- **TypeError** - 如果 x 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
```

(下页继续)

(续上页)

```
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_atanh(x)
>>> print(output.values)
[-inf  nan]
```

mindspore.ops.coo_ceil

mindspore.ops.coo_ceil(*x*: COOTensor)

COOTensor 向上取整函数。

$$out_i = \lceil x_i \rceil = \lfloor x_i \rfloor + 1$$

参数:

- **x** (COOTensor) - Ceil 的输入。其数据类型为 float16 或 float32。

返回:

COOTensor, shape 与 *x* 相同。

异常:

- **TypeError** - *x* 的不是 COOTensor。
- **TypeError** - *x* 的数据类型既不是 float16 也不是 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_ceil(x)
>>> print(output.values)
[-1.  2.]
```

mindspore.ops.coo_concat

mindspore.ops.coo_concat(*sp_input*, *concat_dim*=0)

根据指定的轴 *concat_dim* 对输入的 COO Tensor (*sp_input*) 进行合并操作。

警告: 这是一个实验性 API, 后续可能修改或删除。目前只支持 CPU。

参数:

- **sp_input** (Union[list(COOTensor), tuple(COOTensor)]) - 输入的需要 concat 合并的稀疏张量。
- **concat_dim** (scalar) - 指定需要合并的轴序号, 它的取值必须是在 [-rank, rank) 之内, 其中 rank 为 *sp_input* 中 COOTensor 的 shape 的维度值, 默认 0。

返回：

COOTensor，按 concat_dim 轴合并后的 COOTensor。这个 COOTensor 的稠密 shape 值为：非 concat_dim 轴 shape 与输入一致，concat_dim 轴 shape 是所有输入对应轴 shape 的累加。

异常：

- **ValueError** - 如果只有一个 COOTensor 输入，报错。
- **ValueError** - 如果输入的 COOTensor 的 shape 维度大于 3，COOTensor 的构造会报错。目前 COOTensor 的 shape 维度只能为 2。

支持平台：

CPU

样例：

```
>>> from mindspore import Tensor, ops, COOTensor
>>> from mindspore import dtype as mstype
>>> indices0 = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values0 = Tensor([1, 2], dtype=mstype.int32)
>>> shape0 = (3, 4)
>>> input0 = COOTensor(indices0, values0, shape0)
>>> indices1 = Tensor([[0, 0], [1, 1]], dtype=mstype.int64)
>>> values1 = Tensor([3, 4], dtype=mstype.int32)
>>> shape1 = (3, 4)
>>> input1 = COOTensor(indices1, values1, shape1)
>>> concat_dim = 1
>>> out = ops.coo_concat((input0, input1), concat_dim)
>>> print(out)
COOTensor(shape=[3, 8], dtype=Int32, indices=Tensor(shape=[4, 2], dtype=Int64, value=
[[0 1]
 [0 4]
 [1 2]
 [1 5]]), values=Tensor(shape=[4], dtype=Int32, value=[1 3 2 4]))
```

mindspore.ops.coo_cos

mindspore.ops.coo_cos (x: COOTensor)

逐元素计算 COOTensor 输入的余弦。

$$out_i = \cos(x_i)$$

警告： 目前支持 float16、float32 数据类型。如果使用 float64，可能会存在精度丢失的问题。

参数：

- **x** (COOTensor) - COOTensor 的输入。

返回：

COOTensor，shape 与 x 相同。

异常：

- **TypeError** - x 不是 COOTensor。

- **TypeError** - x 的数据类型不是 float16、float32、float64、complex64 或 complex128。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_cos(x)
>>> print(output.values)
[ 0.5403023 -0.41614684]
```

mindspore.ops.coo_cosh

mindspore.ops.coo_cosh (x : COOTensor)

逐元素计算 COOTensor x 的双曲余弦值。

$$out_i = \cosh(x_i)$$

参数：

- **x** (COOTensor) - coo_cosh 的输入，任意维度的 COOTensor，其数据类型为 float16、float32、float64、complex64、complex128。

返回：

COOTensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - x 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - x 不是 COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_cosh(x)
>>> print(output.values)
[1.5430807 3.7621956]
```

mindspore.ops.coo_exp

mindspore.ops.coo_exp (x: COOTensor)

逐元素计算 COOTensor x 的指数。

$$out_i = e^{x_i}$$

参数:

- **x** (COOTensor) - 指数函数的输入 COOTensor。

返回:

COOTensor, 具有与 x 相同的数据类型和 shape。

异常:

- **TypeError** - x 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_exp(x)
>>> print(output.values)
[0.36787948 7.3890557 ]
```

mindspore.ops.coo_expm1

mindspore.ops.coo_expm1 (x: COOTensor)

逐元素计算输入 COOTensor 的指数, 然后减去 1。

$$out_i = e^{x_i} - 1$$

参数:

- **x** (COOTensor) - 数据类型为 float16 或 float32 的 COOTensor。

返回:

COOTensor, shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 COOTensor。
- **TypeError** - 如果 x 的数据类型既不是 float16 也不是 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_expml(x)
>>> print(output.values)
[-0.63212055  6.389056  ]
```

mindspore.ops.coo_floor

mindspore.ops.coo_floor(x: COOTensor)

COOTensor 逐元素向下取整函数。

$$out_i = \lfloor x_i \rfloor$$

参数:

- **x** (COOTensor) - Floor 的输入，任意维度的 COOTensor。其数据类型必须为 float16、float32、float64。

返回:

COOTensor, shape 与 *x* 相同。

异常:

- **TypeError** - *x* 的数据类型不是 COOTensor。
- **TypeError** - *x* 的数据类型不是 float16、float32、float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_floor(x)
>>> print(output.values)
[-1.  2.]
```

mindspore.ops.coo_inv

mindspore.ops.coo_inv(x: COOTensor)

逐元素计算输入 COOTensor 的倒数。

$$out_i = \frac{1}{x_i}$$

参数:

- **x** (COOTensor) - 任意维度的 COOTensor。其数据类型为 float16、float32 或 int32。

返回：

COOTensor, shape 和类型与输入相同。

异常：

- **TypeError** - x 不是 COOTensor。
- **TypeError** - x 的数据类型不为 float16、float32 或 int32。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_inv(x)
>>> print(output.values)
[-1.    0.5]
```

mindspore.ops.coo_isfinite

mindspore.ops.coo_isfinite (x: COOTensor)

判断 COOTensor 输入数据每个位置上的值是否是有限数。

$$out_i = \begin{cases} \text{if } x_i = \text{Finite}, & \text{True} \\ \text{if } x_i \neq \text{Finite}, & \text{False} \end{cases}$$

参数：

- **x** (COOTensor) - 任意维度的 COOTensor。

返回：

COOTensor, 输出的 shape 与输入相同, 数据类型为 bool。

异常：

- **TypeError** - x 不是 COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_isfinite(x)
>>> print(output.values)
[ True  True]
```

mindspore.ops.coo_isinf

`mindspore.ops.coo_isinf(x: COOTensor)`

确定输入 COOTensor 每个位置上的元素是否为正负无穷大。

$$out_i = \begin{cases} \text{if } x_i = \text{Inf}, & \text{True} \\ \text{if } x_i \neq \text{Inf}, & \text{False} \end{cases}$$

其中 *Inf* 表示正负无穷大。

参数:

- **x** (COOTensor) - 任意维度的 COOTensor。

返回:

COOTensor, shape 与输入相同, 数据类型为 bool。

异常:

- **TypeError** - 如果 *x* 不是 COOTensor。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_isinf(x)
>>> print(output.values)
[False False]
```

mindspore.ops.coo_isnan

`mindspore.ops.coo_isnan(x: COOTensor)`

判断 COOTensor 输入数据每个位置上的值是否是 Nan。

$$out_i = \begin{cases} \text{True}, & \text{if } x_i = \text{Nan} \\ \text{False}, & \text{if } x_i \neq \text{Nan} \end{cases}$$

其中 *Nan* 表示非数字。

参数:

- **x** (COOTensor) - 任意维度的 COOTensor。

返回:

COOTensor, 输出的 shape 与输入相同, 数据类型为 bool。

异常:

- **TypeError** - *x* 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_isnan(x)
>>> print(output.values)
[False False]
```

mindspore.ops.coo_log

mindspore.ops.coo_log (x: COOTensor)

逐元素返回 COOTensor 的自然对数。

$$y_i = \log_e(x_i)$$

警告: 如果算子 Log 的输入值在 (0, 0.01] 或 [0.95, 1.05] 范围内, 则输出精度可能会存在误差。

参数:

- **x** (COOTensor) - 任意维度的输入 COOTensor。该值必须大于 0。

返回:

COOTensor, 具有与 x 相同的 shape。

异常:

- **TypeError** - x 不是 COOTensor。
- **TypeError** - 在 GPU 和 CPU 平台上运行时, x 的数据类型不是 float16、float32 或 float64。
- **TypeError** - 在 Ascend 平台上运行时, x 的数据类型不是 float16 或 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_log(x)
>>> print(output.values)
[      nan 0.69314575]
```

mindspore.ops.coo_log1p

mindspore.ops.coo_log1p (x: COOTensor)

对输入 COOTensor 逐元素加一后计算自然对数。

$$out_i = \log_e(x_i + 1)$$

参数:

- **x** (COOTensor) - 输入 COOTensor。数据类型为 float16 或 float32。该值必须大于-1。

返回:

COOTensor, 与 x 的 shape 相同。

异常:

- **TypeError** - x 不是 COOTensor。
- **TypeError** - x 的数据类型非 float16 或 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_log1p(x)
>>> print(output.values)
[      -inf  1.0986123]
```

mindspore.ops.coo_neg

mindspore.ops.coo_neg (x: COOTensor)

计算输入 COOTensor 的相反数并返回。

$$out_i = -x_i$$

参数:

- **x** (COOTensor) - Neg 的输入，数据类型为数值型。

返回:

COOTensor, shape 和数据类型与输入 x 相同。

异常:

- **TypeError** - x 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_neg(x)
>>> print(output.values)
[ 1. -2.]
```

mindspore.ops.coo_relu

mindspore.ops.coo_relu(*x*: COOTensor)

对输入的 COOTensor 逐元素计算其应用 ReLU 激活函数后的值。

返回 $\max(x, 0)$ 的值。负值神经元将被设置为 0，正值神经元将保持不变。

$$ReLU(x) = (x)^+ = \max(0, x)$$

参数：

- **x** (COOTensor) - coo_relu 的输入，shape: (*N*, *), 其中 * 表示任意数量的附加维度，其数据类型为 `number`。

返回：

COOTensor，数据类型和 shape 与 *x* 相同。

异常：

- **TypeError** - *x* 的数据类型不是数值型。
- **TypeError** - *x* 不是 COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_relu(x)
>>> print(output.values)
[0. 2.]
```

mindspore.ops.coo_relu6

mindspore.ops.coo_relu6(x: COOTensor)

对输入的 COOTensor 逐元素计算其应用 ReLU 激活函数后的值, 上限为 6。

$$\text{ReLU6}(x) = \min(\max(0, x), 6)$$

返回 $\min(\max(0, x), 6)$ 元素的值。

参数:

- **x** (COOTensor) - coo_relu6 的输入, 数据类型为 float16 或 float32。

返回:

COOTensor, 数据类型和 shape 与 x 相同。

异常:

- **TypeError** - 如果 x 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 x 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_relu6(x)
>>> print(output.values)
[0. 2.]
```

mindspore.ops.coo_round

mindspore.ops.coo_round(x: COOTensor)

对 COOTensor 输入数据进行四舍五入到最接近的整数数值。

$$out_i \approx x_i$$

参数:

- **x** (COOTensor) - 输入 COOTensor。

返回:

COOTensor, shape 和数据类型与输入 x 相同。

异常:

- **TypeError** - x 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_round(x)
>>> print(output.values)
[-1.  2.]
```

mindspore.ops.coo_sigmoid

mindspore.ops.coo_sigmoid(x: COOTensor)

Sigmoid 激活函数，对 COOTensor 逐元素计算 Sigmoid 激活函数。Sigmoid 函数定义为：

$$\text{coo\_sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)}$$

其中， x_i 是 x 的一个元素。

参数：

- **x** (COOTensor) - 任意维度的 COOTensor，数据类型为 float16、float32、float64、complex64 或 complex128。

返回：

COOTensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - x 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - x 不是 COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_sigmoid(x)
>>> print(output.values)
[0.26894143 0.8807971 ]
```

mindspore.ops.coo_sin

mindspore.ops.coo_sin (x: COOTensor)

逐元素计算输入 COOTensor 的正弦。

$$out_i = \sin(x_i)$$

参数:

- **x** (COOTensor) - COOTensor 的输入。

返回:

COOTensor, shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 COOTensor。
- **TypeError** - 如果 x 的数据类型不是 float16、float32、float64、complex64 或 complex128。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_sin(x)
>>> print(output.values)
[-0.84147096  0.9092974 ]
```

mindspore.ops.coo_sinh

mindspore.ops.coo_sinh (x: COOTensor)

逐元素计算输入 COOTensor 的双曲正弦。

$$out_i = \sinh(x_i)$$

参数:

- **x** (COOTensor) - coo_sinh 的输入 COOTensor。

返回:

COOTensor, shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:


```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_sinh(x)
>>> print(output.values)
[-1.1752012  3.6268604]
```

mindspore.ops.coo_softsign

mindspore.ops.coo_softsign(x: COOTensor)

COOTensor Softsign 激活函数。

Softsign 函数定义为：

$$\text{SoftSign}(x) = \frac{x}{1 + |x|}$$

参数：

- **x** (COOTensor) - COOTensor 的输入。它的数据类型必须为 float16 或 float32。

返回：

COOTensor，数据类型和 shape 与 *x* 相同。

异常：

- **TypeError** - *x* 不是 COOTensor。
- **TypeError** - *x* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_softsign(x)
>>> print(output.values)
[-0.5          0.6666667]
```

mindspore.ops.coo_sqrt**mindspore.ops.coo_sqrt** (*x*: COOTensor)

逐元素计算 COOTensor 的平方根。

$$out_i = \sqrt{x_i}$$

参数:

- **x** (COOTensor) - 输入 COOTensor, 数据类型为 number.Number。

返回:

COOTensor, shape 与 *x* 相同。

异常:

- **TypeError** - 如果 *x* 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_sqrt(x)
>>> print(output.values)
[      nan 1.4142135]
```

mindspore.ops.coo_square**mindspore.ops.coo_square** (*x*: COOTensor)

逐元素返回 COOTensor 的平方。

$$out_i = (x_i)^2$$

参数:

- **x** (COOTensor) - 输入 COOTensor 的类型为数值类型。

返回:

COOTensor, 具有与当前 COOTensor 相同的数据类型和 shape。

异常:

- **TypeError** - *x* 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_square(x)
>>> print(output.values)
[1. 4.]
```

mindspore.ops.coo_tan

mindspore.ops.coo_tan(x: COOTensor)

计算 COOTensor 输入元素的正切值。

$$out_i = \tan(x_i)$$

参数:

- **x** (COOTensor) - Tan 的输入，任意维度的 COOTensor。

返回:

COOTensor，数据类型和 shape 与 *x* 相同。

异常:

- **TypeError** - *x* 不是 COOTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_tan(x)
>>> print(output.values)
[-1.5574077 -2.1850398]
```

mindspore.ops.coo_tanh

mindspore.ops.coo_tanh(x: COOTensor)

按元素计算 COOTensor 输入元素的双曲正切。Tanh 函数定义为:

$$\tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 COOTensor 的元素。

参数:

- **x** (COOTensor) - Tanh 的输入，任意维度的 COOTensor，其数据类型为 float16 或 float32。

返回：

COOTensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - x 的数据类型既不是 float16 也不是 float32。
- **TypeError** - x 不是 COOTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, COOTensor
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mstype.int64)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = COOTensor(indices, values, shape)
>>> output = ops.coo_tanh(x)
>>> print(output.values)
[-0.7615942  0.9640276]
```

2.7.5 CSR 函数

警告：这些是实验性 API，后续可能修改或删除。

接口名	概述	支持平台
<code>mindspore.ops.csr_abs</code>	逐元素计算输入 CSRTensor 的绝对值。	Ascend GPU CPU
<code>mindspore.ops.csr_acos</code>	逐元素计算输入 CSRTensor 的反余弦。	Ascend GPU CPU
<code>mindspore.ops.csr_acosh</code>	逐元素计算输入 CSRTensor 的反双曲余弦。	Ascend GPU CPU
<code>mindspore.ops.csr_add</code>	计算两个输入 CSRTensor 的线性组合。	GPU CPU
<code>mindspore.ops.csr_asin</code>	逐元素计算输入 CSRTensor 的正弦。	Ascend GPU CPU
<code>mindspore.ops.csr_asinh</code>	计算 CSRTensor 输入元素的反双曲正弦。	Ascend GPU CPU
<code>mindspore.ops.csr_atan</code>	逐元素计算输入 CSRTensor 的正切值。	Ascend GPU CPU
<code>mindspore.ops.csr_atanh</code>	逐元素计算输入 CSRTensor 的反双曲正切值。	Ascend GPU CPU
<code>mindspore.ops.csr_ceil</code>	CSRTensor 向上取整函数。	Ascend GPU CPU
<code>mindspore.ops.csr_cos</code>	逐元素计算 CSRTensor 输入的余弦。	Ascend GPU CPU
<code>mindspore.ops.csr_cosh</code>	逐元素计算 CSRTensor x 的双曲余弦值。	Ascend GPU CPU
<code>mindspore.ops.csr_exp</code>	逐元素计算 CSRTensor x 的指数。	Ascend GPU CPU
<code>mindspore.ops.csr_expm1</code>	逐元素计算输入 CSRTensor 的指数，然后减去 1。	Ascend GPU CPU
<code>mindspore.ops.csr_floor</code>	CSRTensor 逐元素向下取整函数。	Ascend GPU CPU
<code>mindspore.ops.csr_inv</code>	逐元素计算输入 CSRTensor 的倒数。	Ascend GPU CPU
<code>mindspore.ops.csr_isfinite</code>	判断 CSRTensor 输入数据每个位置上的值是否是有限数。	Ascend GPU CPU
<code>mindspore.ops.csr_isinf</code>	确定输入 CSRTensor 每个位置上的元素是否为无穷大或无穷小。	GPU CPU
<code>mindspore.ops.csr_isnan</code>	判断 CSRTensor 输入数据每个位置上的值是否是 NaN。	Ascend GPU CPU

下页继续

表 23 - 续上页

<code>mindspore.ops.csr_log</code>	逐元素返回 CSRTensor 的自然对数。	Ascend GPU CPU
<code>mindspore.ops.csr_log1p</code>	对输入 CSRTensor 逐元素加一后计算自然对数。	Ascend GPU CPU
<code>mindspore.ops.csr_mm</code>	返回稀疏矩阵 a 与稀疏矩阵或稠密矩阵 b 的矩阵乘法结果。	GPU
<code>mindspore.ops.csr_neg</code>	计算输入 CSRTensor 的相反数并返回。	Ascend GPU CPU
<code>mindspore.ops.csr_relu</code>	逐元素计算 CSRTensor 的 ReLU (Rectified Linear Unit) 激活值。	Ascend GPU CPU
<code>mindspore.ops.csr_relu6</code>	逐元素计算 CSRTensor 的 ReLU 值, 其上限为 6。	Ascend GPU CPU
<code>mindspore.ops.csr_round</code>	对 CSRTensor 输入数据进行四舍五入到最近的整数数值。	Ascend GPU CPU
<code>mindspore.ops.csr_sigmoid</code>	Sigmoid 激活函数, CSRTensor 逐元素计算 Sigmoid 激活函数。	Ascend GPU CPU
<code>mindspore.ops.csr_sin</code>	逐元素计算输入 CSRTensor 的正弦。	Ascend GPU CPU
<code>mindspore.ops.csr_sinh</code>	逐元素计算输入 CSRTensor 的双曲正弦。	Ascend GPU CPU
<code>mindspore.ops.csr_softmax</code>	计算 CSRTensorMatrix 的 softmax。	GPU CPU
<code>mindspore.ops.csr_softsign</code>	CSRTensor 的 Softsign 激活函数。	Ascend GPU CPU
<code>mindspore.ops.csr_sqrt</code>	逐元素返回当前 CSRTensor 的平方根。	Ascend GPU CPU
<code>mindspore.ops.csr_square</code>	逐元素返回 CSRTensor 的平方。	Ascend GPU CPU
<code>mindspore.ops.csr_tan</code>	逐元素计算 CSRTensor 的正切值。	Ascend GPU CPU
<code>mindspore.ops.csr_tanh</code>	逐元素计算 CSRTensor 输入元素的双曲正切。	Ascend GPU CPU

mindspore.ops.csr_abs

`mindspore.ops.csr_abs` (*x*: CSRTensor)

逐元素计算输入 CSRTensor 的绝对值。

$$out_i = |x_i|$$

参数:

- **x** (CSRTensor) - 输入 CSRTensor。

返回:

CSRTensor, shape 与 *x* 相同。

异常:

- **TypeError** - 如果 *x* 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
```

(下页继续)

(续上页)

```
>>> output = ops.csr_abs(x)
>>> print(output.values)
[1. 2.]
```

mindspore.ops.csr_acos

mindspore.ops.csr_acos (x: CSRTensor)

逐元素计算输入 CSRTensor 的反余弦。

$$out_i = \cos^{-1}(x_i)$$

参数:

- **x** (CSRTensor) - 输入 CSRTensor。

返回:

CSRTensor, shape 和数据类型与 *x* 相同。

异常:

- **TypeError** - 如果 *x* 不是 CSRTensor。
- **TypeError** - 如果 *x* 的数据类型不是 float16、float32 或 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_acos(x)
>>> print(output.values)
[3.1415927      nan]
```

mindspore.ops.csr_acosh

mindspore.ops.csr_acosh (x: CSRTensor)

逐元素计算输入 CSRTensor 的反双曲余弦。

$$out_i = \cosh^{-1}(input_i)$$

参数:

- **x** (CSRTensor) - 需要计算反双曲余弦函数的输入 CSRTensor, 即 $input_i$, 其每个元素的取值范围必须在 $[1, \inf]$ 。

返回:

CSRTensor, 数据类型与 *x* 相同。

异常:

- **TypeError** - 如果 x 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_acosh(x)
>>> print(output.values)
[      nan 1.316958]
```

mindspore.ops.csr_add

`mindspore.ops.csr_add(a: CSRTensor, b: CSRTensor, alpha: Tensor, beta: Tensor)`

计算两个输入 CSRTensor 的线性组合。

$$out = alpha * a + beta * b$$

说明: 用户需要确保输入的 CSRTensor 的合法性。否则, 算子将返回错误结果。例如, 当同一个位置有多个元素时, 算子可能会返回错误结果或执行失败。

参数:

- **a** (CSRTensor) - 稀疏的 CSRTensor。
- **b** (CSRTensor) - 稀疏的 CSRTensor。
- **alpha** (Tensor) - 稠密张量, shape 必须可以广播到 a 。
- **beta** (Tensor) - 稠密张量, shape 必须可以广播到 b 。

返回:

返回一个包含以下数据的 CSRTensor:

- **indptr** - 指示每行中非零值的起始点和结束点。
- **indices** - 输入中所有非零值的列位置。
- **values** - 非零值。
- **shape** - CSRTensor 的形状。

支持平台:

GPU CPU

样例:

```
>>> import mindspore.common.dtype as mstype
>>> from mindspore import Tensor, CSRTensor
>>> from mindspore import ops
>>> a_indptr = Tensor([0, 1, 2], dtype=mstype.int32)
>>> a_indices = Tensor([0, 1], dtype=mstype.int32)
>>> a_values = Tensor([1, 2], dtype=mstype.float32)
>>> shape = (2, 6)
>>> b_indptr = Tensor([0, 1, 2], dtype=mstype.int32)
>>> b_indices = Tensor([0, 1], dtype=mstype.int32)
>>> b_values = Tensor([1, 2], dtype=mstype.float32)
>>> alpha = Tensor(1, mstype.float32)
>>> beta = Tensor(1, mstype.float32)
>>> csra = CSRTensor(a_indptr, a_indices, a_values, shape)
>>> csrb = CSRTensor(b_indptr, b_indices, b_values, shape)
>>> out = ops.csr_add(csra, csrb, alpha, beta)
>>> print(out)
CSRTensor(shape=[2, 6], dtype=Float32, indptr=Tensor(shape=[3],
dtype=Int32, value=[0 1 2]), indices=Tensor(shape=[2], dtype=Int32,
value=[0 1]), values=Tensor(shape=[2], dtype=Float32, value=[ 2.
00000000e+00  4.00000000e+00]))
```

mindspore.ops.csr_asin

`mindspore.ops.csr_asin(x: CSRTensor)`

逐元素计算输入 CSRTensor 的反正弦。

$$out_i = \sin^{-1}(x_i)$$

参数：

- **x** (CSRTensor) - CSRTensor 的输入。数据类型应该是以下类型之一：float16、float32、float64。

返回：

CSRTensor，数据类型和 shape 与 *x* 相同。

异常：

- **TypeError** - *x* 不是 CSRTensor。
- **TypeError** - *x* 的数据类型不是 float16、float32、float64。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_asin(x)
>>> print(output.values)
[-1.5707964      nan]
```


mindspore.ops.csr_asinh

mindspore.ops.csr_asinh(x: CSRTensor)

计算 CSRTensor 输入元素的反双曲正弦。

$$out_i = \sinh^{-1}(input_i)$$

参数:

- **x** (CSRTensor) - 需要计算反双曲正弦函数的输入, 即 $input_i$ 。

返回:

CSRTensor, 数据类型和 shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_asinh(x)
>>> print(output.values)
[-0.8813736  1.4436355]
```

mindspore.ops.csr_atan

mindspore.ops.csr_atan(x: CSRTensor)

逐元素计算输入 CSRTensor 的反正切值。

$$out_i = \tan^{-1}(x_i)$$

参数:

- **x** (CSRTensor) - 数据类型支持: float16、float32。

返回:

CSRTensor, 数据类型与输入相同。

异常:

- **TypeError** - x 不是 CSRTensor。
- **TypeError** - x 的数据类型不是 float16 或 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_atanh(x)
>>> print(output.values)
[-0.7853982  1.1071488]
```

mindspore.ops.csr_atanh

mindspore.ops.csr_atanh(x: CSRTensor)

逐元素计算输入 CSRTensor 的反双曲正切值。

$$out_i = \tanh^{-1}(x_i)$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **x** (CSRTensor) - CSRTensor 的 shape 为 $(N,*)$ ，其中 * 表示任意数量的附加维度。数据类型支持：float16、float32。

返回：

CSRTensor，数据类型与输入相同。

异常：

- **TypeError** - x 不是 CSRTensor。
- **TypeError** - x 的数据类型不是 float16 或 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_atanh(x)
>>> print(output.values)
[-inf nan]
```

mindspore.ops.csr_ceil

mindspore.ops.csr_ceil (x: CSRTensor)

CSRTensor 向上取整函数。

$$out_i = \lceil x_i \rceil = \lfloor x_i \rfloor + 1$$

参数:

- **x** (CSRTensor) - Ceil 的输入。其数据类型为 float16 或 float32。

返回:

CSRTensor, shape 与 x 相同。

异常:

- **TypeError** - x 不是 CSRTensor。
- **TypeError** - x 的数据类型既不是 float16 也不是 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_ceil(x)
>>> print(output.values)
[-1.  2.]
```

mindspore.ops.csr_cos

mindspore.ops.csr_cos (x: CSRTensor)

逐元素计算 CSRTensor 输入的余弦。

$$out_i = \cos(x_i)$$

警告: 目前支持 float16、float32 数据类型。如果使用 float64, 可能会存在精度丢失的问题。

参数:

- **x** (CSRTensor) - CSRTensor 的输入。

返回:

CSRTensor, shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 CSRTensor。

- **TypeError** - 如果 x 的数据类型不是 float16、float32、float64、complex64 或 complex128。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_cos(x)
>>> print(output.values)
[ 0.5403023 -0.41614684]
```

mindspore.ops.csr_cosh

mindspore.ops.csr_cosh(x : CSRTensor)

逐元素计算 CSRTensor x 的双曲余弦值。

$$out_i = \cosh(x_i)$$

参数：

- **x** (CSRTensor) - csr_cosh 的输入，任意维度的 CSRTensor，其数据类型为 float16、float32、float64、complex64、complex128。

返回：

CSRTensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - x 的数据类型不是 float16、float32、float64、complex64、complex128。
- **TypeError** - x 不是 CSRTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_cosh(x)
>>> print(output.values)
[1.5430807 3.7621956]
```

mindspore.ops.csr_exp

`mindspore.ops.csr_exp` (x : CSRTensor)

逐元素计算 CSRTensor x 的指数。

$$out_i = e^{x_i}$$

参数:

- x (CSRTensor) - 指数函数的输入 CSRTensor。

返回:

CSRTensor, 具有与 x 相同的数据类型和 shape。

异常:

- **TypeError** - x 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_exp(x)
>>> print(output.values)
[0.36787948 7.3890557 ]
```

mindspore.ops.csr_expm1

`mindspore.ops.csr_expm1` (x : CSRTensor)

逐元素计算输入 CSRTensor 的指数, 然后减去 1。

$$out_i = e^{x_i} - 1$$

参数:

- x (CSRTensor) - 数据类型为 float16 或 float32 的 CSRTensor。

返回:

CSRTensor, shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 CSRTensor。
- **TypeError** - 如果 x 的数据类型既不是 float16 也不是 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_expml(x)
>>> print(output.values)
[-0.63212055  6.389056  ]
```

mindspore.ops.csr_floor

mindspore.ops.csr_floor(x: CSRTensor)

CSRTensor 逐元素向下取整函数。

$$out_i = \lfloor x_i \rfloor$$

参数:

- **x** (CSRTensor) - Floor 的输入, 任意维度的 CSRTensor。其数据类型必须为 float16、float32 或 float64。

返回:

CSRTensor, shape 与 *x* 相同。

异常:

- **TypeError** - *x* 的类型不是 CSRTensor。
- **TypeError** - *x* 的数据类型不是 float16、float32 或 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_floor(x)
>>> print(output.values)
[-1.  2.]
```

mindspore.ops.csr_inv

mindspore.ops.csr_inv(x: CSRTensor)

逐元素计算输入 CSRTensor 的倒数。

$$out_i = \frac{1}{x_i}$$

参数:

- **x** (CSRTensor) - 任意维度的 CSRTensor。其数据类型为 float16、float32 或 int32。

返回:

CSRTensor, shape 和类型与输入相同。

异常:

- **TypeError** - *x* 不是 CSRTensor。
- **TypeError** - *x* 的数据类型不为 float16、float32 或 int32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_inv(x)
>>> print(output.values)
[-1.    0.5]
```

mindspore.ops.csr_isfinite

mindspore.ops.csr_isfinite(x: CSRTensor)

判断 CSRTensor 输入数据每个位置上的值是否是有限数。

$$out_i = \begin{cases} \text{if } x_i = \text{Finite}, & \text{True} \\ \text{if } x_i \neq \text{Finite}, & \text{False} \end{cases}$$

参数:

- **x** (CSRTensor) - IsFinite 的输入, 任意维度的 CSRTensor。

返回:

CSRTensor, 输出的 shape 与输入相同, 数据类型为 bool。

异常:

- **TypeError** - *x* 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_isfinite(x)
>>> print(output.values)
[ True  True]
```

mindspore.ops.csr_isinf

mindspore.ops.csr_isinf (x: CSRTensor)

确定输入 CSRTensor 每个位置上的元素是否为无穷大或无穷小。

$$out_i = \begin{cases} \text{if } x_i = \text{Inf}, & \text{True} \\ \text{if } x_i \neq \text{Inf}, & \text{False} \end{cases}$$

其中 *Inf* 表示不是一个数字。

参数:

- **x** (CSRTensor) - IsInf 的输入。

返回:

CSRTensor, shape 与相同的输入, 数据的类型为 bool。

异常:

- **TypeError** - 如果 *x* 不是 CSRTensor。

支持平台:

GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_isinf(x)
>>> print(output.values)
[False False]
```


mindspore.ops.csr_isnan

`mindspore.ops.csr_isnan(x: CSRTensor)`

判断 CSRTensor 输入数据每个位置上的值是否是 NaN。

$$out_i = \begin{cases} True, & \text{if } x_i = \text{NaN} \\ False, & \text{if } x_i \neq \text{NaN} \end{cases}$$

其中 *NaN* 表示非数值 (Not a Number)。

参数:

- **x** (CSRTensor) - IsNaN 的输入, 任意维度的 CSRTensor。

返回:

CSRTensor, 输出的 shape 与输入相同, 数据类型为 bool。

异常:

- **TypeError** - *x* 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_isnan(x)
>>> print(output.values)
[False False]
```

mindspore.ops.csr_log

`mindspore.ops.csr_log(x: CSRTensor)`

逐元素返回 CSRTensor 的自然对数。

$$y_i = \log_e(x_i)$$

警告: 如果算子 Log 的输入值在 (0, 0.01] 或 [0.95, 1.05] 范围内, 则输出精度可能会存在误差。

参数:

- **x** (CSRTensor) - 任意维度的输入 CSRTensor。值小于 0 时, 返回 nan。

返回:

CSRTensor, 具有与 *x* 相同的 shape。

异常:

- **TypeError** - x 不是 CSRTensor。
- **TypeError** - 在 GPU 和 CPU 平台上运行时, x 的数据类型不是 float16、float32 或 float64。
- **TypeError** - 在 Ascend 平台上运行时, x 的数据类型不是 float16 或 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_log(x)
>>> print(output.values)
[      nan 0.69314575]
```

mindspore.ops.csr_log1p

mindspore.ops.csr_log1p(x : CSRTensor)

对输入 CSRTensor 逐元素加一后计算自然对数。

$$out_i = \log_e(x_i + 1)$$

参数:

- **x** (CSRTensor) - 输入 CSRTensor。数据类型为 float16 或 float32。该值必须大于-1。

返回:

CSRTensor, 与 x 的 shape 相同。

异常:

- **TypeError** - x 不是 CSRTensor。
- **TypeError** - x 的数据类型非 float16 或 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_log1p(x)
>>> print(output.values)
[      -inf 1.0986123]
```

mindspore.ops.csr_mm

`mindspore.ops.csr_mm(a: CSRTensor, b: CSRTensor, trans_a: bool = False, trans_b: bool = False, adjoint_a: bool = False, adjoint_b: bool = False)`

返回稀疏矩阵 **a** 与稀疏矩阵或稠密矩阵 **b** 的矩阵乘法结果。

说明：若右矩阵为 `Tensor`，则仅支持安装了 LLVM12.0.1 及以上版本的 CPU 后端或 GPU 后端。若右矩阵为 `CSRTensor`，则仅支持 GPU 后端。

参数：

- **a** (`CSRTensor`) - 稀疏的 `CSRTensor`。
- **b** (`CSRTensor`) - 稀疏的 `CSRTensor` 或稠密矩阵。
- **trans_a** (bool, 可选) - 是否对矩阵 **a** 进行转置。默认 `False`。
- **trans_b** (bool, 可选) - 是否对矩阵 **b** 进行转置。默认 `False`。
- **adjoint_a** (bool, 可选) - 是否对矩阵 **a** 进行共轭。默认 `False`。
- **adjoint_b** (bool, 可选) - 是否对矩阵 **b** 进行共轭。默认 `False`。

返回：

返回稀疏矩阵，类型为 `CSRTensor`。

支持平台：

GPU

样例：

```
>>> from mindspore import Tensor, CSRTensor
>>> from mindspore import dtype as mstype
>>> from mindspore import ops
>>> a_shape = (4, 5)
>>> a_indptr = Tensor([0, 1, 1, 3, 4], dtype=mstype.int32)
>>> a_indices = Tensor([0, 3, 4, 0], dtype=mstype.int32)
>>> a_values = Tensor([1.0, 5.0, -1.0, -2.0], dtype=mstype.float32)
>>> b_shape = (5, 3)
>>> b_indptr = Tensor([0, 1, 1, 3, 3, 3], dtype=mstype.int32)
>>> b_indices = Tensor([0, 0, 1], dtype=mstype.int32)
>>> b_values = Tensor([2.0, 7.0, 8.0], dtype=mstype.float32)
>>> a = CSRTensor(a_indptr, a_indices, a_values, a_shape)
>>> b = CSRTensor(b_indptr, b_indices, b_values, b_shape)
>>> c = ops.csr_mm(a, b)
>>> print(c.shape)
(4, 3)
>>> print(c.values)
[2. -4.]
>>> print(c.indptr)
[0 1 1 1 2]
>>> print(c.indices)
[0 0]
```

mindspore.ops.csr_neg

mindspore.ops.csr_neg (x: CSRTensor)

计算输入 CSRTensor 的相反数并返回。

$$out_i = -x_i$$

参数:

- **x** (CSRTensor) - Neg 的输入，其数据类型为数值型。

返回:

CSRTensor, shape 和类型与输入相同。

异常:

- **TypeError** - *x* 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_neg(x)
>>> print(output.values)
[ 1. -2.]
```

mindspore.ops.csr_relu

mindspore.ops.csr_relu (x: CSRTensor)

逐元素计算 CSRTensor 的 ReLU (Rectified Linear Unit) 激活值。

返回 $\max(x, 0)$ 的值，其中负值神经元将被设置为 0，而正值神经元将保持不变。

$$ReLU(x) = (x)^+ = \max(0, x)$$

参数:

- **x** (CSRTensor) - csr_relu 的输入。

返回:

CSRTensor, 数据类型和 shape 与 *x* 相同。

异常:

- **TypeError** - *x* 的数据类型不是数值型。
- **TypeError** - *x* 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_relu(x)
>>> print(output.values)
[0. 2.]
```

mindspore.ops.csr_relu6

`mindspore.ops.csr_relu6` (x : `CSRTensor`)

逐元素计算 `CSRTensor` 的 ReLU 值，其上限为 6。

$$\text{ReLU6}(x) = \min(\max(0, x), 6)$$

返回 $\min(\max(0, x), 6)$ 元素的值。

参数:

- **x** (`CSRTensor`) - `csr_relu6` 的输入，数据类型为 `float16` 或 `float32`。

返回:

`CSRTensor`，数据类型和 `shape` 与 x 相同。

异常:

- **TypeError** - 如果 x 的数据类型既不是 `float16` 也不是 `float32`。
- **TypeError** - 如果 x 不是 `CSRTensor`。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_relu6(x)
>>> print(output.values)
[0. 2.]
```

mindspore.ops.csr_round

mindspore.ops.csr_round(x: CSRTensor)

对 CSRTensor 输入数据进行四舍五入到最近的整数数值。

$$out_i \approx x_i$$

参数:

- **x** (CSRTensor) - 输入 CSRTensor。

返回:

CSRTensor, shape 和数据类型与输入 x 相同。

异常:

- **TypeError** - x 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_round(x)
>>> print(output.values)
[-1.  2.]
```

mindspore.ops.csr_sigmoid

mindspore.ops.csr_sigmoid(x: CSRTensor)

Sigmoid 激活函数, CSRTensor 逐元素计算 Sigmoid 激活函数。Sigmoid 函数定义为:

$$\text{csr\_sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)}$$

其中, x_i 是 x 的一个元素。

参数:

- **x** (CSRTensor) - 任意维度的 CSRTensor, 数据类型为 float16、float32、float64、complex64 或 complex128。

返回:

CSRTensor, 数据类型和 shape 与 x 的相同。

异常:

- **TypeError** - x 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - x 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_sigmoid(x)
>>> print(output.values)
[0.26894143 0.8807971 ]
```

mindspore.ops.csr_sin

mindspore.ops.csr_sin(*x*: CSRTensor)

逐元素计算输入 CSRTensor 的正弦。

$$out_i = \sin(x_i)$$

参数:

- **x** (CSRTensor) - CSRTensor 的输入。

返回:

CSRTensor, shape 与 *x* 相同。

异常:

- **TypeError** - 如果 *x* 不是 CSRTensor。
- **TypeError** - 如果 *x* 的数据类型不是 float16、float32 或者 float64、complex64、complex128。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_sin(x)
>>> print(output.values)
[-0.84147096 0.9092974 ]
```

mindspore.ops.csr_sinh

`mindspore.ops.csr_sinh(x: CSRTensor)`

逐元素计算输入 CSRTensor 的双曲正弦。

$$out_i = \sinh(x_i)$$

参数:

- **x** (CSRTensor) - `csr_sinh` 的输入 CSRTensor。

返回:

CSRTensor, shape 与 *x* 相同。

异常:

- **TypeError** - 如果 *x* 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_sinh(x)
>>> print(output.values)
[-1.1752012  3.6268604]
```

mindspore.ops.csr_softmax

`mindspore.ops.csr_softmax(logits: CSRTensor, dtype: mstype)`

计算 CSRTensorMatrix 的 softmax。

参数:

- **logits** (CSRTensor) - 输入稀疏的 CSRTensor。
- **dtype** (dtype) - 输入的数据类型。

返回:

- **CSRTensor** (CSRTensor) - 一个 `csr_tensor` 包含
 - **indptr** - 指示每行中非零值的起始点和结束点。
 - **indices** - 输入中所有非零值的列位置。
 - **values** - 稠密张量的非零值。
 - **shape** - `csr_tensor` 的 shape。

支持平台:

GPU CPU

样例:


```
>>> import mindspore as ms
>>> from mindspore import ops
>>> import mindspore.common.dtype as mstype
>>> from mindspore import Tensor, CSRTensor
>>> logits_indptr = Tensor([0, 4, 6], dtype=mstype.int32)
>>> logits_indices = Tensor([0, 2, 3, 4, 3, 4], dtype=mstype.int32)
>>> logits_values = Tensor([1, 2, 3, 4, 1, 2], dtype=mstype.float32)
>>> shape = (2, 6)
>>> logits = CSRTensor(logits_indptr, logits_indices, logits_values, shape)
>>> out = ops.csr_softmax(logits, dtype=mstype.float32)
>>> print(out)
CSRTensor(shape=[2, 6], dtype=Float32, indptr=Tensor(shape=[3], dtype=Int32, value=[0 4 6]),
            indices=Tensor(shape=[6], dtype=Int32, value=[0 2 3 4 3 4]),
            values=Tensor(shape=[6], dtype=Float32, value=[ 3.20586003e-02  8.71443152e-
↪02  2.36882806e-01
                        6.43914223e-01  2.68941432e-01  7.31058598e-01]))
```

mindspore.ops.csr_softsign

mindspore.ops.csr_softsign(x: CSRTensor)

CSRTensor 的 Softsign 激活函数。

Softsign 函数定义为：

$$\text{SoftSign}(x) = \frac{x}{1 + |x|}$$

参数：

- **x** (CSRTensor) - CSRTensor，它的数据类型必须为 float16 或 float32。

返回：

CSRTensor，数据类型和 shape 与 *x* 相同。

异常：

- **TypeError** - *x* 不是 CSRTensor。
- **TypeError** - *x* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_softsign(x)
>>> print(output.values)
[-0.5      0.6666667]
```

mindspore.ops.csr_sqrt

`mindspore.ops.csr_sqrt` (x : `CSRTensor`)

逐元素返回当前 `CSRTensor` 的平方根。

$$out_i = \sqrt{x_i}$$

参数:

- **x** (`CSRTensor`) - 输入 `CSRTensor`, 数据类型为 `number.Number`。

返回:

`CSRTensor`, 具有与 x 相同的 `shape`。

异常:

- **TypeError** - 如果 x 不是 `CSRTensor`。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_sqrt(x)
>>> print(output.values)
[      nan 1.4142135]
```

mindspore.ops.csr_square

`mindspore.ops.csr_square` (x : `CSRTensor`)

逐元素返回 `CSRTensor` 的平方。

$$out_i = (x_i)^2$$

参数:

- **x** (`CSRTensor`) - 输入 `CSRTensor`, 数据类型为数值类型。

返回:

`CSRTensor`, 具有与当前 `CSRTensor` 相同的数据类型和 `shape`。

异常:

- **TypeError** - x 不是 `CSRTensor`。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_square(x)
>>> print(output.values)
[1. 4.]
```

mindspore.ops.csr_tan

mindspore.ops.csr_tan(x: CSRTensor)

逐元素计算 CSRTensor 的正切值。

$$out_i = \tan(x_i)$$

参数:

- **x** (CSRTensor) - Tan 的输入，任意维度的 CSRTensor。

返回:

CSRTensor，数据类型和 shape 与 *x* 相同。

异常:

- **TypeError** - *x* 不是 CSRTensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_tan(x)
>>> print(output.values)
[-1.5574077 -2.1850398]
```

mindspore.ops.csr_tanh

mindspore.ops.csr_tanh(x: CSRTensor)

逐元素计算 CSRTensor 输入元素的双曲正切。Tanh 函数定义为:

$$\tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 CSRTensor 的元素。

参数:

- **x** (CSRTensor) - Tanh 的输入，任意维度的 CSRTensor，其数据类型为 float16 或 float32。

返回：

CSRTensor，数据类型和 shape 与 *x* 相同。

异常：

- **TypeError** - *x* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - *x* 不是 CSRTensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, CSRTensor
>>> indptr = Tensor([0, 1, 2, 2], dtype=mstype.int32)
>>> indices = Tensor([3, 0], dtype=mstype.int32)
>>> values = Tensor([-1, 2], dtype=mstype.float32)
>>> shape = (3, 4)
>>> x = CSRTensor(indptr, indices, values, shape)
>>> output = ops.csr_tanh(x)
>>> print(output.values)
[-0.7615942  0.9640276]
```

2.8 MC2 函数

警告：这些是实验性 API，后续可能修改或删除。

接口名	概述	支持平台
<code>mindspore.ops.all_gather_matmul</code>	TP 切分场景下，实现 allgather 和 matmul 的融合，融合算子内部实现通信和计算流水并行。	Ascend
<code>mindspore.ops.matmul_reduce_scatter</code>	TP 切分场景下，实现 matmul 和 reducescatter 的融合，融合算子内部实现通信和计算流水并行。	Ascend

2.8.1 mindspore.ops.all_gather_matmul

`mindspore.ops.all_gather_matmul(input, x2, group, world_size, *, bias=None, gather_index=0, gather_output=True, comm_turn=0, trans_input=False, trans_x2=False)`

TP 切分场景下，实现 allgather 和 matmul 的融合，融合算子内部实现通信和计算流水并行。

output = allgather(input)@x2
gather_out = allgather(input)

警告：这是一个实验性 API，后续可能修改或删除。

参数:

- **input** (Tensor) - matmul 的左矩阵, dtype 支持 float16、bfloat16, shape 支持二维, 数据格式支持 ND。
- **x2** (Tensor) - matmul 的右矩阵, dtype 需要和 input 一致, shape 支持二维, 数据格式支持 ND。
- **group** (str) - 通信组名称, 可以由 create_group 方法创建, 或者使用默认组 mindspore.communication.GlobalComm.WORLD_COMM_GROUP。
- **world_size** (int) - 通信组的总进程数, 要求与实际运行的卡数一致, 支持 2、4、8。

关键字参数:

- **bias** (Tensor, 可选) - 当前仅支持 None。默认 None。
- **gather_index** (int, 可选) - 表示 allgather 操作对象, 0 表示对 input 做 gather, 1 表示对 x2 做 gather。当前仅支持 0。默认 0。
- **gather_output** (bool, 可选) - 表示是否需要 gather 输出。默认 True。
- **comm_turn** (int, 可选) - 表示进程间通信切分粒度。当前仅支持 0。默认 0。
- **trans_input** (bool, 可选) - 表示 input 是否转置。当前仅支持 False。默认 False。
- **trans_x2** (bool, 可选) - 表示 x2 是否转置。默认 False。

返回:

- **output** (Tensor) - allgather 和 matmul 融合计算的结果。
- **gather_out** (Tensor) - allgather 的结果。如果 gather_output 为 False, gather_out 返回 shape 为 0 的 tensor。

说明:

- 使用该接口时, 请确保驱动固件包和 CANN 包都为配套的 8.0.RC2 版本或者配套的更高版本, 否则将会引发报错, 比如 BUS ERROR 等。
- input 的 shape 为 (m, k), x2 的 shape 为 (k, n), 要求 k 相等, 且 k 的取值范围为 [256, 65535)。output 的 shape 为 (m * world_size, n), gather_out 的 shape 为 (m * world_size, k)。
- 一个模型中的通算融合算子仅支持相同通信组。

异常:

- **TypeError** - 参数的类型不对。
- **RuntimeError** - input 或 x2 的 dtype 不是 float16 或 bfloat16。
- **RuntimeError** - input 和 x2 的 dtype 不一致。
- **RuntimeError** - input 或 x2 的 shape 不是二维。
- **RuntimeError** - input shape 和 x2 shape 的 k 不相等。
- **RuntimeError** - k 小于 256 或大于等于 65535。
- **RuntimeError** - bias 不是 None。
- **RuntimeError** - group 不存在。
- **RuntimeError** - world_size 与实际运行的卡数不一致。
- **RuntimeError** - world_size 不等于 2、4、8。
- **RuntimeError** - gather_index 不是 0。

- **RuntimeError** - trans_input 为 True。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import ops
>>> ms.communication.init()
>>> rank = ms.communication.get_rank()
>>> np.random.seed(rank)
>>> input = ms.Tensor(np.random.randn(128, 256).astype(np.float32), dtype=ms.float16)
>>> x2 = ms.Tensor(np.random.randn(256, 512).astype(np.float32), dtype=ms.float16)
>>> group = ms.communication.GlobalComm.WORLD_COMM_GROUP
>>> world_size = ms.communication.get_group_size()
>>> output, gather_out = ops.all_gather_matmul(
...     input,
...     x2,
...     group,
...     world_size,
...     bias=None,
...     gather_index=0,
...     gather_output=True,
...     comm_turn=0,
...     trans_input=False,
...     trans_x2=False,
... )
>>> print(output.shape)
(256, 512)
>>> print(gather_out.shape)
(256, 256)
```

2.8.2 mindspore.ops.matmul_reduce_scatter

`mindspore.ops.matmul_reduce_scatter` (*input*, *x2*, *group*, *world_size*, *, *reduce_op*='sum', *bias*=None, *comm_turn*=0, *trans_input*=False, *trans_x2*=False)

TP 切分场景下，实现 `matmul` 和 `reducescatter` 的融合，融合算子内部实现通信和计算流水并行。

output = `reducescatter`(*input*@*x2*)

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - `matmul` 的左矩阵，`dtype` 支持 `float16`、`bfloat16`，`shape` 支持二维，数据格式支持 ND。

- **x2** (Tensor) - matmul 的右矩阵, dtype 需要和 input 一致, shape 支持二维, 数据格式支持 ND。
- **group** (str) - 通信组名称, 可以由 create_group 方法创建, 或者使用默认组 mindspore.communication.GlobalComm.WORLD_COMM_GROUP。
- **world_size** (int) - 通信组的总进程数, 要求与实际运行的卡数一致, 支持 2、4、8。

关键字参数:

- **reduce_op** (str, 可选) - reduce 操作类型。当前仅支持 'sum'。默认 'sum'。
- **bias** (Tensor, 可选) - 当前仅支持 None。默认 None。
- **comm_turn** (int, 可选) - 表示进程间通信切分粒度。当前仅支持 0。默认 0。
- **trans_input** (bool, 可选) - 表示 input 是否转置。当前仅支持 False。默认 False。
- **trans_x2** (bool, 可选) - 表示 x2 是否转置。默认 False。

返回:

- **output** (Tensor) - matmul 和 reducescatter 融合计算的结果。

说明:

- 使用该接口时, 请确保驱动固件包和 CANN 包都为配套的 8.0.RC2 版本或者配套的更高版本, 否则将会引发报错, 比如 BUS ERROR 等。
- input 的 shape 为 (m, k), x2 的 shape 为 (k, n), 要求 k 相等, 且 k 的取值范围为 [256, 65535), 要求 m 是 world_size 的整数倍。output 的 shape 为 (m * world_size, n)。
- 一个模型中的通算融合算子仅支持相同通信组。

异常:

- **TypeError** - 参数的类型不对。
- **RuntimeError** - input 或 x2 的 dtype 不是 float16 或 bfloat16。
- **RuntimeError** - input 和 x2 的 dtype 不一致。
- **RuntimeError** - input 或 x2 的 shape 不是二维。
- **RuntimeError** - input shape 和 x2 shape 的 k 不相等。
- **RuntimeError** - k 小于 256 或大于等于 65535。
- **RuntimeError** - bias 不是 None。
- **RuntimeError** - group 不存在。
- **RuntimeError** - world_size 与实际运行的卡数不一致。
- **RuntimeError** - world_size 不等于 2、4、8。
- **RuntimeError** - reduce_op 不是 'sum'。
- **RuntimeError** - trans_input 为 True。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import mindspore as ms
>>> from mindspore import ops
>>> import numpy as np
>>> ms.communication.init()
>>> rank = ms.communication.get_rank()
>>> np.random.seed(rank)
>>> input = ms.Tensor(np.random.randn(1024, 256).astype(np.float32), dtype=ms.float16)
>>> x2 = ms.Tensor(np.random.randn(256, 512).astype(np.float32), dtype=ms.float16)
>>> group = ms.communication.GlobalComm.WORLD_COMM_GROUP
>>> world_size = ms.communication.get_group_size()
>>> reduce_op = ops.ReduceOp.SUM
>>> output = ops.matmul_reduce_scatter(
...     input,
...     x2,
...     group,
...     world_size,
...     reduce_op=reduce_op,
...     bias=None,
...     comm_turn=0,
...     trans_input=False,
...     trans_x2=False,
... )
>>> print(output.shape)
(512, 512)
```


神经网络 Cell。

用于构建神经网络中的预定义构建块或计算单元。

动态 shape 的支持情况详见 [nn 接口动态 shape 支持情况](#)。

MindSpore 中 *mindspore.nn* 接口与上一版本相比，新增、删除和支持平台的变化信息请参考 [mindspore.nn API 接口变更](#)。

3.1 基本构成单元

接口名	概述	支持平台
<i>mindspore.nn.Cell</i>	MindSpore 中神经网络的基本构成单元。	Ascend GPU CPU
<i>mindspore.nn.GraphCell</i>	运行从 MindIR 加载的计算图。	Ascend GPU CPU
<i>mindspore.nn.LossBase</i>	损失函数的基类。	Ascend GPU CPU
<i>mindspore.nn.Optimizer</i>	用于参数更新的优化器基类。	Ascend GPU CPU

3.1.1 mindspore.nn.Cell

class *mindspore.nn.Cell* (*auto_prefix=True, flags=None*)

MindSpore 中神经网络的基本构成单元。模型或神经网络层应当继承该基类。

mindspore.nn 中神经网络层也是 Cell 的子类，如 *mindspore.nn.Conv2d*、*mindspore.nn.ReLU* 等。Cell 在 GRAPH_MODE（静态图模式）下将编译为一张计算图，在 PYNATIVE_MODE（动态图模式）下作为神经网络的基础模块。

说明：Cell 默认情况下是推理模式。对于继承 Cell 的类，如果训练和推理具有不同结构，子类会默认执行推理分支。设置训练模式，请参考 *mindspore.nn.Cell.set_train*。

警告：在 Cell 的子类中不能定义名为'cast'的方法，不能定义名为'phase'和'cells'的属性，否则会报错。

参数：

- **auto_prefix** (bool, 可选) - 是否自动为 Cell 及其子 Cell 生成 NameSpace。该参数同时会影响 Cell 中权重参数的名称。如果设置为 True，则自动给权重参数的名称添加前缀，否则不添加前缀。通常情况下，骨干网络应设置为 True，否则会产生重名问题。用于训练骨干网络的优化器、`mindspore.nn.TrainOneStepCell` 等，应设置为 False，否则骨干网络的权重参数名会被误改。默认值：True。
- **flags** (dict, 可选) - Cell 的配置信息，目前用于绑定 Cell 和数据集。用户也可通过该参数自定义 Cell 属性。默认值：None。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> class MyCell(nn.Cell):
...     def __init__(self, forward_net):
...         super(MyCell, self).__init__(auto_prefix=False)
...         self.net = forward_net
...         self.relu = ops.ReLU()
...
...     def construct(self, x):
...         y = self.net(x)
...         return self.relu(y)
>>>
>>> inner_net = nn.Conv2d(120, 240, 4, has_bias=False, weight_init='normal')
>>> my_net = MyCell(inner_net)
>>> print(my_net.trainable_params())
... # If the 'auto_prefix' set to True or not set when call the '__init__' method of the
... # parent class,
... # the parameter's name will be 'net.weight'.
[Parameter (name=weight, shape=(240, 120, 4, 4), dtype=Float32, requires_grad=True)]
```

add_flags (**flags)

为 Cell 添加自定义属性。

在实例化 Cell 类时，如果入参 flags 不为空，会调用此方法。

参数：

- **flags** (dict) - Cell 的配置信息，目前用于绑定 Cell 和数据集。用户也可通过该参数自定义 Cell 属性。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.relu = nn.ReLU()
...
...     def construct(self, x):
```

(下页继续)

(续上页)

```

...         x = self.relu(x)
...         return x
>>> net = Net()
>>> net.add_flags(sink_mode=True)
>>> print(net.sink_mode)
True

```

add_flags_recursive(flags)**

如果 Cell 含有多个子 Cell，此方法会递归地给所有子 Cell 添加自定义属性。

参数：

- **flags** (dict) - Cell 的配置信息，目前用于绑定 Cell 和数据集。用户也可通过该参数自定义 Cell 属性。

样例：

```

>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.relu = nn.ReLU()
...
...     def construct(self, x):
...         x = self.relu(x)
...         return x
>>> net = Net()
>>> net.add_flags_recursive(sink_mode=True)
>>> print(net.sink_mode)
True

```

apply(fn)

递归地将 *fn* 应用于每个子 Cell（由 *.cells()* 返回）以及自身。通常用于初始化模型的参数。

参数：

- **fn** (function) - 被执行于每个 Cell 的 function。

返回：

Cell 类型，Cell 本身。

样例：

```

>>> import mindspore.nn as nn
>>> from mindspore.common.initializer import initializer, One
>>> net = nn.SequentialCell(nn.Dense(2, 2), nn.Dense(2, 2))
>>> def func(cell):
...     if isinstance(cell, nn.Dense):
...         cell.weight.set_data(initializer(One(), cell.weight.shape, cell.weight.
...dtype))
>>> net.apply(func)
SequentialCell(
  (0): Dense(input_channels=2, output_channels=2, has_bias=True)
  (1): Dense(input_channels=2, output_channels=2, has_bias=True)
)
>>> print(net[0].weight.asnumpy())

```

(下页继续)

(续上页)

```
[[1. 1.]
 [1. 1.]]
```

auto_cast_inputs (*inputs*)

在混合精度下，自动对输入进行类型转换。

参数：

- **inputs** (tuple) - construct 方法的输入。

返回：

Tuple 类型，经过类型转换后的输入。

property bprop_debug

在图模式下使用，用于标识是否使用自定义的反向传播函数。

buffers (*recurse: bool = True*)

返回 Cell 缓冲区的迭代器，只包含缓冲区本身。

参数：

- **recurse** (bool, 可选) - 如果为 True，则生成此 Cell 及其子 Cell 的缓冲区。否则，仅生成此 Cell 的缓冲区。默认 True。

返回：

Iterator[Tensor]，缓冲区的迭代器。

样例：

```
>>> import mindspore
...
...
>>> class NetB(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer_b", mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return x + self.buffer_b
...
...
>>> class NetA(mindspore.nn.Cell):
...     def __init__(self, net_b):
...         super().__init__()
...         self.net_b = net_b
...         self.register_buffer("buffer_a", mindspore.tensor([4, 5, 6]))
...
...     def construct(self, x):
...         return self.net_b(x) + self.buffer_a
...
...
>>> net_b = NetB()
>>> net_a = NetA(net_b)
>>>
>>> for buffer in net_a.buffers():
...     print(f'buffer is {buffer}')
buffer is [4 5 6]
buffer is [1 2 3]
```

cast_inputs (*inputs, dst_type*)

将输入转换为指定类型。

参数：

- **inputs** (tuple[Tensor]) - 输入。
- **dst_type** (mindspore.dtype) - 指定的数据类型。

返回：

tuple[Tensor] 类型，转换类型后的结果。

cast_param (*param*)

在 PyNative 模式下，根据自动混合精度的精度设置转换 Cell 中参数的类型。

该接口目前在自动混合精度场景下使用。

参数：

- **param** (Parameter) - 需要被转换类型的输入参数。

返回：

Parameter 类型，转换类型后的参数。

cells ()

返回当前 Cell 的子 Cell 的迭代器。

返回：

Iteration 类型，Cell 的子 Cell。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.dense = nn.Dense(2, 2)
...
...     def construct(self, x):
...         x = self.dense(x)
...         return x
>>> net = Net()
>>> print(net.cells())
odict_values([Dense(input_channels=2, output_channels=2, has_bias=True)])
```

cells_and_names (*cells=None, name_prefix=""*)

递归地获取当前 Cell 及输入 *cells* 的所有子 Cell 的迭代器，包括 Cell 的名称及其本身。

参数：

- **cells** (str) - 需要进行迭代的 Cell。默认值：None。
- **name_prefix** (str) - 作用域。默认值：''。

返回：

Iteration 类型，当前 Cell 及输入 *cells* 的所有子 Cell 和相对应的名称。

样例：

```

>>> from mindspore import nn
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv = nn.Conv2d(3, 64, 3)
...     def construct(self, x):
...         out = self.conv(x)
...         return out
>>> names = []
>>> n = Net()
>>> for m in n.cells_and_names():
...     if m[0]:
...         names.append(m[0])

```

check_names()

检查 Cell 中的网络参数名称是否重复。

compile(*args, **kwargs)

编译 Cell 为计算图，输入需与 construct 中定义的输入一致。

参数：

- **args** (tuple) - Cell 的输入。
- **kwargs** (dict) - Cell 的输入。

compile_and_run(*args, **kwargs)

编译并运行 Cell，输入需与 construct 中定义的输入一致。

说明：不推荐使用该函数，建议直接调用 Cell 实例。

参数：

- **args** (tuple) - Cell 的输入。
- **kwargs** (dict) - Cell 的输入。

返回：

Object 类型，执行的结果。

construct(*args, **kwargs)

定义要执行的计算逻辑。所有子类都必须重写此方法。

说明：当前不支持 inputs 同时输入 tuple 类型和非 tuple 类型。

参数：

- **args** (tuple) - 可变参数列表，默认值：()。
- **kwargs** (dict) - 可变的关键字参数的字典，默认值：{ }。

返回：

Tensor 类型，返回计算结果。

extend_repr()

在原有描述基础上扩展 Cell 的描述。

若需要在 print 时输出个性化的扩展信息, 请在您的网络中重新实现此方法。

flatten_weights (*fusion_size=0*)

重置权重参数 (即可训练参数) 使用的数据内存, 让这些参数按数据类型分组使用连续内存块。

说明: 默认情况下, 具有相同数据类型的参数会使用同一个连续内存块。但对于某些具有大量参数的模型, 将一个大的连续内存块分为多个小一点的内存块, 有可能提升性能, 对于这种情况, 可以通过 *fusion_size* 参数来限制最大连续内存块的大小。

参数:

- **fusion_size** (int) - 最大连续内存块的大小 (以字节为单位), 0 表示不限制大小。默认值: 0。

generate_scope()

为网络中的每个 Cell 对象生成 NameSpace。

get_buffer (*target: str*)

返回给定 *target* 的缓冲区, 如果不存在则抛出错误。

请参阅 *get_sub_cell* 的文档, 了解有关此方法功能的更详细说明以及如何正确指定 *target*。

参数:

- **target** (str) - 要查找的缓冲区的完全限定字符串名称。(请参阅 *get_sub_cell* 了解如何指定完全限定字符串。)

返回:

Tensor

样例:

```
>>> import mindspore
...
...
>>> class NetC(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer_c", mindspore.tensor([0, 0, 0]))
...
...     def construct(self, x):
...         return x + self.buffer_c
...
...
>>> class NetB(mindspore.nn.Cell):
...     def __init__(self, net_c):
...         super().__init__()
...         self.net_c = net_c
...         self.register_buffer("buffer_b", mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return self.net_c(x) + self.buffer_b
...
...
>>> class NetA(mindspore.nn.Cell):
```

(下页继续)

(续上页)

```

...     def __init__(self, net_b):
...         super().__init__()
...         self.net_b = net_b
...         self.register_buffer("buffer_a", mindspore.tensor([4, 5, 6]))
...
...     def construct(self, x):
...         return self.net_b(x) + self.buffer_a
...
...
>>> net_c = NetC()
>>> net_b = NetB(net_c)
>>> net_a = NetA(net_b)
>>> buffer_c = net_a.get_buffer("net_b.net_c.buffer_c")
>>> print(f'buffer_c is {buffer_c}')
buffer_c is [0 0 0]

```

get_extra_state()

返回要包含在 Cell 的 `state_dict` 中的任何额外状态。

当构建 Cell 的 `state_dict()` 时，将调用此函数。如果您需要存储额外状态，实现此方法，并为您的 Cell 实现相应的 `set_extra_state()`。

请注意，额外状态应为可序列化对象（`pickleable`），以确保 `state_dict` 的序列化可用性。仅对 `tensor` 的序列化提供向后兼容性保证；对于其他对象，如果其序列化的 `pickled` 形式发生变化，可能会导致向后兼容性问题。

返回：

object，要存储在 Cell 的 `state_dict` 中的额外状态。

get_flags()

获取该 Cell 的自定义属性，自定义属性通过 `add_flags` 方法添加。

样例：

```

>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.relu = nn.ReLU()
...
...     def construct(self, x):
...         x = self.relu(x)
...         return x
>>> net = Net()
>>> net.add_flags(sink_mode=True)
>>> print(net.get_flags())
{'sink_mode': True}

```

get_func_graph_proto()

返回图的二进制原型。

get_inputs()

返回编译计算图所设置的输入。

返回：

Tuple 类型，编译计算图所设置的输入。

警告：这是一个实验性 API，后续可能修改或删除。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, Tensor
>>>
>>> class ReluNet(nn.Cell):
...     def __init__(self):
...         super(ReluNet, self).__init__()
...         self.relu = nn.ReLU()
...     def construct(self, x):
...         return self.relu(x)
>>>
>>> net = ReluNet()
>>> input_dyn = Tensor(shape=[3, None], dtype=ms.float32)
>>> net.set_inputs(input_dyn)
>>> get_inputs = net.get_inputs()
>>> print(get_inputs)
(Tensor(shape=[3, -1], dtype=Float32, value= ),)
```

get_parameters (expand=True)

返回 Cell 中 parameter 的迭代器。

获取 Cell 的参数。如果 *expand* 为 true，获取此 cell 和所有 subcells 的参数。关于 subcell，请看下面的示例。

参数：

- **expand** (bool) - 如果为 True，则递归地获取当前 Cell 和所有子 Cell 的 parameter。否则，只生成当前 Cell 的 subcell 的 parameter。默认值：True。

返回：

Iteration 类型，Cell 的 parameter。

样例：

```
>>> import mindspore as ms
>>> from mindspore import nn, ops, Tensor
>>> import numpy as np
>>> class TestNet(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.my_w1 = ms.Parameter(Tensor(np.ones([4, 4]), ms.float32))
...         self.my_w2 = ms.Parameter(Tensor(np.ones([16]), ms.float32))
...     def construct(self, x):
...         x += self.my_w1
...         x = ops.reshape(x, (16,)) - self.my_w2
...         return x
>>> class TestNet2(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.my_t1 = ms.Parameter(Tensor(np.ones([4, 4]), ms.float32))
...         # self.subcell is a subcell of TestNet2, when using expand=True, the_
        ↪ parameters of TestNet will
...         # also be gathered.
...         self.subcell = TestNet()
```

(下页继续)

(续上页)

```

...     def construct(self, x):
...         x += self.my_w1
...         x = ops.reshape(x, (16,)) - self.my_w2
...         return x
>>> net = TestNet2()
>>> print([p for p in net.get_parameters(expand=True)])
[Parameter (name=my_t1, shape=(4, 4), dtype=Float32, requires_grad=True), Parameter
↳ (name=subcell.my_w1,
shape=(4, 4), dtype=Float32, requires_grad=True), Parameter (name=subcell.my_w2,
↳ shape=(16,), dtype=Float32,
requires_grad=True)]

```

get_scope()

返回 Cell 的作用域。

返回:

String 类型, 网络的作用域。

get_sub_cell(target: str)

返回给定 *target* 的子 Cell, 如果不存在则抛出错误。

例如, 假设你有一个 *nn.Cell A*, 如下所示:

```

A(
  (net_b): NetB(
    (net_c): NetC(
      (conv): Conv2d(16, 33, kernel_size=(3, 3), stride=(2, 2))
    )
    (dense): Dense(in_features=100, out_features=200, bias=True)
  )
)

```

(该图显示了 *nn.Cell A*。A 有一个嵌套的子 Cell *net_b*, 而后者本身又有两个子 Cell *net_c* 和 *dense*。 *net_c* 则有一个子 Cell *conv*。)

要检查是否拥有子 Cell *dense*, 我们将调用 `get_sub_cell("net_b.dense")`。要检查是否拥有子 Cell *conv*, 我们将调用 `get_sub_cell("net_b.net_c.conv")`。

`get_sub_cell` 的运行时间受 *target* 中 Cell 嵌套程度的限制。使用 `name_cells` 的查询可获得相同的结果, 但传递的 Cell 的数量级为 $O(N)$ 。因此, 为了简单检查是否存在某些子 Cell, 应始终使用 `get_sub_cell`。

参数:

- **target** (str) - 要查找的子 Cell 的完全限定字符串名称。(请参阅上述示例以了解如何指定完全限定字符串。)

返回:

Cell

样例:

```

>>> import mindspore
...
...
>>> class NetC(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer_c", mindspore.tensor([0, 0, 0]))

```

(下页继续)

(续上页)

```

...     self.dense_c = mindspore.nn.Dense(5, 3)
...
...     def construct(self, x):
...         return self.dense_c(x) + self.buffer_c
...
...
>>> class NetB(mindspore.nn.Cell):
...     def __init__(self, net_c):
...         super().__init__()
...         self.net_c = net_c
...         self.register_buffer("buffer_b", mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return self.net_c(x) + self.buffer_b
...
...
>>> class NetA(mindspore.nn.Cell):
...     def __init__(self, net_b):
...         super().__init__()
...         self.net_b = net_b
...         self.register_buffer("buffer_a", mindspore.tensor([4, 5, 6]))
...
...     def construct(self, x):
...         return self.net_b(x) + self.buffer_a
...
...
>>> net_c = NetC()
>>> net_b = NetB(net_c)
>>> net_a = NetA(net_b)
>>> net_c = net_a.get_sub_cell("net_b.net_c")
>>> print(f'net_c is {net_c}')
net_c is NetC(
  (dense_c): Dense(input_channels=5, output_channels=3, has_bias=True)
)

```

infer_param_pipeline_stage()

推导 Cell 中当前 *pipeline_stage* 的参数。

说明:

- 这个接口在 2.3 版本废弃，并且会在未来版本移除。

返回:

属于当前 *pipeline_stage* 的参数。

异常:

- **RuntimeError** - 如果参数不属于任何 stage。

init_parameters_data (auto_parallel_mode=False)

初始化并替换 Cell 中所有的 parameter 的值。

说明: 在调用 *init_parameters_data* 后, *trainable_params()* 或其他相似的接口可能返回不同的参数对象, 不建议保

存这些结果。

参数:

- **auto_parallel_mode** (bool) - 是否在自动并行模式下执行。默认值: `False`。

返回:

`Dict[Parameter, Parameter]`, 返回一个原始参数和替换参数的字典。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.dense = nn.Dense(2, 2)
...
...     def construct(self, x):
...         x = self.dense(x)
...         return x
>>> net = Net()
>>> print(net.init_parameters_data())
{Parameter (name=dense.weight, shape=(2,2), dtype=Float32, requires_grad=True):
Parameter (name=dense.weight, shape=(2,2), dtype=Float32, requires_grad=True),
Parameter (name=dense.bias, shape=(2,), dtype=Float32, requires_grad=True):
Parameter (name=dense.bias, shape=(2,), dtype=Float32, requires_grad=True)}
```

insert_child_to_cell (*child_name*, *child_cell*)

将一个给定名称的子 Cell 添加到当前 Cell。

参数:

- **child_name** (str) - 子 Cell 名称。
- **child_cell** (Cell) - 要插入的子 Cell。

异常:

- **KeyError** - 如果子 Cell 的名称不正确或其他子 Cell 名称重复。
- **TypeError** - 如果 *child_name* 的类型不为 str 类型。
- **TypeError** - 如果子 Cell 的类型不正确。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> net1 = nn.ReLU()
>>> net2 = nn.Dense(2, 2)
>>> net1.insert_child_to_cell("child", net2)
>>> print(net1)
ReLU(
  (child): Dense(input_channels=2, output_channels=2, has_bias=True)
)
```

insert_param_to_cell (*param_name*, *param*, *check_name_contain_dot=True*)

向当前 Cell 添加参数。

将指定名称的参数添加到 Cell 中。目前在 `mindspore.nn.Cell.__setattr__` 中使用。

参数：

- **param_name** (str) - 参数名称。
- **param** (Parameter) - 要插入到 Cell 的参数。
- **check_name_contain_dot** (bool) - 是否对 *param_name* 中的“.”进行检查。默认值：True。

异常：

- **KeyError** - 如果参数名称为空或包含“.”。
- **TypeError** - 如果参数的类型不是 Parameter。

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, Parameter
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.relu = nn.ReLU()
...
...     def construct(self, x):
...         x = self.relu(x)
...         return x
>>> net = Net()
>>> net.insert_param_to_cell("bias", Parameter(Tensor([1, 2, 3])))
>>> print(net.bias)
Parameter(name=bias, shape=(3,), dtype=Int64, requires_grad=True)
```

load_state_dict (*state_dict: Mapping[str, Any]*, *strict: bool = True*)

将 *state_dict* 中的参数和缓冲区复制到当前 Cell 及其子 Cell 中。

如果 *strict* 设置为 True，则 *state_dict* 的键必须与该 Cell 的 `mindspore.nn.Cell.state_dict()` 方法返回的键完全匹配。

参数：

- **state_dict** (dict) - 包含参数和持久缓冲区的字典。
- **strict** (bool, 可选) - 是否严格要求输入 *state_dict* 中的键必须与当前 Cell 的 `mindspore.nn.Cell.state_dict()` 方法返回的键匹配。默认 True。

返回：

一个包含 *missing_keys* 和 *unexpected_keys* 字段的 `namedtuple`，

- *missing_keys* 是一个包含字符串的列表，表示当前 Cell 需要但在 *state_dict* 中缺失的键。
- *unexpected_keys* 是一个包含字符串的列表，表示 *state_dict* 中存在但当前 Cell 不需要的键。

说明： 如果 *strict* 为 True 且某个参数或缓冲区被注册为 None，但其对应的键在 *state_dict* 中存在，则 `mindspore.nn.Cell.load_state_dict()` 将会抛出 `RuntimeError`。

样例：

```

>>> import mindspore
>>> import os
>>> class Model(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer_a", mindspore.tensor([4, 5, 6]))
...         self.param_a = mindspore.Parameter(mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return x + self.buffer_a + self.param_a
...
>>> model = Model()
>>> print(model.state_dict())
>>> mindspore.save_checkpoint(model.state_dict(), './model_state_dict_ckpt')
>>> new_model = Model()
>>> new_model.load_state_dict(mindspore.load_checkpoint('./model_state_dict_ckpt'))
>>> print(new_model.state_dict())
>>> os.remove('./model_state_dict_ckpt')
OrderedDict([('param_a', Parameter (name=param_a, shape=(3,), dtype=Int64, requires_
↳grad=True)), \
('buffer_a', Tensor(shape=[3], dtype=Int64, value= [4, 5, 6]))])
OrderedDict([('param_a', Parameter (name=param_a, shape=(3,), dtype=Int64, requires_
↳grad=True)), \
('buffer_a', Tensor(shape=[3], dtype=Int64, value= [4, 5, 6]))])

```

name_cells()

递归地获取一个 Cell 中所有子 Cell 的迭代器。

包括 Cell 名称和 Cell 本身。

返回：

Dict[String, Cell], Cell 中的所有子 Cell 及其名称。

样例：

```

>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.dense = nn.Dense(2, 2)
...
...     def construct(self, x):
...         x = self.dense(x)
...         return x
>>> net = Net()
>>> print(net.name_cells())
OrderedDict([('dense', Dense(input_channels=2, output_channels=2, has_bias=True))])

```

named_buffers (prefix: str = "", recurse: bool = True, remove_duplicate: bool = True)

返回 Cell 中缓冲区的迭代器，包含缓冲区的名称以及缓冲区本身。

参数：

- **prefix** (str, 可选) - 添加到所有缓冲区名称前面的前缀。默认 ""。

- **recurse** (bool, 可选) - 如果为 True，则生成此 Cell 和所有子 Cell 的缓冲区。否则，仅生成此 Cell 的缓冲区。默认 True。
- **remove_duplicate** (bool, 可选) - 是否删除结果中的重复缓冲区。默认 True。

返回:

Iterator[Tuple[str, Tensor]], 包含名称和缓冲区的元组的迭代器。

样例:

```
>>> import mindspore
...
...
>>> class NetB(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer_b", mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return x + self.buffer_b
...
...
>>> class NetA(mindspore.nn.Cell):
...     def __init__(self, net_b):
...         super().__init__()
...         self.net_b = net_b
...         self.register_buffer("buffer_a", mindspore.tensor([4, 5, 6]))
...
...     def construct(self, x):
...         return self.net_b(x) + self.buffer_a
...
...
>>> net_b = NetB()
>>> net_a = NetA(net_b)
>>>
>>> for name, buffer in net_a.named_buffers():
...     print(f'buffer name is {name}, buffer is {buffer}')
buffer name is buffer_a, buffer is [4 5 6]
buffer name is net_b.buffer_b, buffer is [1 2 3]
```

offload (*backward_prefetch='Auto'*)

设置 Cell 激活值卸载，设置后该 Cell 中所有的 Primitive 类会被使能激活值卸载标签。若激活值需要在反向阶段被用于计算梯度，则该激活值会在正向阶段被搬运至 host 侧，不会存储在 device 侧，并在反向阶段使用其之前，预取回 device 侧。

说明:

- 当某个 Cell 被标记为 offload 时，运行模型必须为“GRAPH_MODE”模式。
- 当某个 Cell 被标记为 offload 时，需要开启 lazyinline。

参数:

- **backward_prefetch** (Union[str, int], 可选) - 设置反向阶段提前预取激活值的时机。默认值: "Auto"。当为 "Auto" 时，框架将提前一个算子开始预取激活值；当为正整数时，框架将提前 backward_prefetch 个算子开始预期激活值，例如 1、20、100。

样例:

```
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore.common import Tensor, Parameter
>>> from mindspore.common.lazy_inline import lazy_inline
>>>
>>> class Block(nn.Cell):
...     def __init__(self):
...         super(Block, self).__init__()
...         self.transpose1 = ops.Transpose()
...         self.transpose2 = ops.Transpose()
...         self.transpose3 = ops.Transpose()
...         self.transpose4 = ops.Transpose()
...         self.real_div1 = ops.RealDiv()
...         self.real_div2 = ops.RealDiv()
...         self.batch_matmul1 = ops.BatchMatMul()
...         self.batch_matmul2 = ops.BatchMatMul()
...         self.softmax = ops.Softmax(-1)
...         self.expand_dims = ops.ExpandDims()
...         self.sub = ops.Sub()
...         self.y = Parameter(Tensor(np.ones((1024, 128, 128)).astype(np.float32)))
...     def construct(self, x):
...         transpose1 = self.transpose1(x, (0, 2, 1, 3))
...         real_div1 = self.real_div1(transpose1, Tensor(2.37891))
...         transpose2 = self.transpose2(x, (0, 2, 3, 1))
...         real_div2 = self.real_div2(transpose2, Tensor(2.37891))
...         batch_matmul1 = self.batch_matmul1(real_div1, real_div2)
...         expand_dims = self.expand_dims(self.y, 1)
...         sub = self.sub(Tensor([1.0]), expand_dims)
...         soft_max = self.softmax(sub)
...         transpose3 = self.transpose3(x, (0, 2, 1, 3))
...         batch_matmul2 = self.batch_matmul2(soft_max[0], transpose3)
...         transpose4 = self.transpose4(batch_matmul2, (0, 2, 1, 3))
...         return transpose4
>>>
>>> class OuterBlock(nn.Cell):
...     @lazy_inline
...     def __init__(self):
...         super(OuterBlock, self).__init__()
...         self.block = Block()
...     def construct(self, x):
...         return self.block(x)
>>>
>>> class Nets(nn.Cell):
...     def __init__(self):
...         super(Nets, self).__init__()
...         self.blocks = nn.CellList()
...         for _ in range(3):
...             b = OuterBlock()
...             b.offload()
...             self.blocks.append(b)
...     def construct(self, x):
...         out = x
...         for i in range(3):
...             out = self.blocks[i](out)
...         return out
```


property param_prefix

当前 Cell 的子 Cell 的参数名前缀。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.dense = nn.Dense(2, 2)
...
...     def construct(self, x):
...         x = self.dense(x)
...         return x
>>> net = Net()
>>> net.update_cell_prefix()
>>> print(net.dense.param_prefix)
dense
```

property parameter_layout_dict

parameter_layout_dict 表示一个参数的张量 layout，这种张量 layout 是由分片策略和分布式算子信息推断出来的。

parameters_and_names (*name_prefix="", expand=True*)

返回 Cell 中 parameter 的迭代器。

包含参数名称和参数本身。

参数:

- **name_prefix** (str) - 作用域。默认值: ''。
- **expand** (bool) - 如果为 True，则递归地获取当前 Cell 和所有子 Cell 的参数及名称；如果为 False，只生成当前 Cell 的子 Cell 的参数及名称。默认值: True。

返回:

迭代器，Cell 的名称和 Cell 本身。

样例:

```
>>> from mindspore import nn
>>> n = nn.Dense(3, 4)
>>> names = []
>>> for m in n.parameters_and_names():
...     if m[0]:
...         names.append(m[0])
```

教程样例:

- [网络构建 - 模型参数](#)

parameters_broadcast_dict (*recurse=True*)

获取这个 Cell 的参数广播字典。

参数:

- **recurse** (bool) - 是否包含子 Cell 的参数。默认值: True。

返回:

OrderedDict, 返回参数广播字典。

parameters_dict (*recurse=True*)

获取此 Cell 的 parameter 字典。

参数:

- **recurse** (bool) - 是否递归地包含所有子 Cell 的 parameter。默认值: True。

返回:

OrderedDict 类型, 返回参数字典。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, Parameter
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.dense = nn.Dense(2, 2)
...
...     def construct(self, x):
...         x = self.dense(x)
...         return x
>>> net = Net()
>>> print(net.parameters_dict())
OrderedDict([('dense.weight', Parameter(name=dense.weight, shape=(2, 2), dtype=Float32,
requires_grad=True)), ('dense.bias', Parameter(name=dense.bias, shape=(2,),
dtype=Float32,
requires_grad=True))])
```

property pipeline_stage

pipeline_stage 表示当前 Cell 所在的 stage。

place (*role, rank_id*)

为该 Cell 中所有算子设置标签。此标签告诉 MindSpore 编译器此 Cell 在哪个进程上启动。每个标签都由进程角色 *role* 和 *rank_id* 组成, 因此, 通过对不同 Cell 设置不同标签, 这些 Cell 将在不同进程启动, 使用户可以进行分布式训练/推理等任务。

说明:

- 此接口只在成功调用 *mindspore.communication.init()* 完成动态组网后才能生效。

参数:

- **role** (str) - 算子执行所在进程的角色。只支持 'MS_WORKER'。
- **rank_id** (int) - 算子执行所在进程的 ID。在相同进程角色间, *rank_id* 是唯一的。

样例:

```
>>> from mindspore import context
>>> import mindspore.nn as nn
>>> context.set_context(mode=context.GRAPH_MODE)
>>> fc = nn.Dense(2, 3)
>>> fc.place('MS_WORKER', 0)
```

recompute (***kwargs*)

设置 Cell 重计算。Cell 中输出算子以外的所有算子将被设置为重计算。如果一个算子的计算结果被输出到一些反向节点来进行梯度计算，且被设置成重计算，那么我们会在反向传播中重新计算它，而不去存储在正向传播中的中间激活层的计算结果。

说明：

- 如果计算涉及到诸如随机化或全局变量之类的操作，那么目前还不能保证等价。
- 如果该 Cell 中算子的重计算 API 也被调用，则该算子的重计算模式以算子的重计算 API 的设置为准。
- 该接口仅配置一次，即当父 Cell 配置了，子 Cell 不需再配置。
- Cell 的输出算子默认不做重计算，这一点是基于我们减少内存占用的配置经验。如果一个 Cell 里面只有一个算子，且想要把这个算子设置为重计算的，那么请使用算子的重计算 API。
- 当应用了重计算且内存充足时，可以配置 `mp_comm_recompute=False` 来提升性能。
- 当应用了重计算但内存不足时，可以配置 `parallel_optimizer_comm_recompute=True` 来节省内存。有相同融合 group 的 Cell 应该配置相同的 `parallel_optimizer_comm_recompute`。

参数：

- **mp_comm_recompute** (bool) - 表示在自动并行或半自动并行模式下，指定 Cell 内部由模型并行引入的通信操作是否重计算。默认值：True。
- **parallel_optimizer_comm_recompute** (bool) - 表示在自动并行或半自动并行模式下，指定 Cell 内部由优化器并行引入的 AllGather 通信是否重计算。默认值：False。

register_backward_hook (*hook_fn*)

设置 Cell 对象的反向 hook 函数。

说明：

- `register_backward_hook(hook_fn)` 在图模式下，或者在 PyNative 模式下使用 `jit` 装饰器功能时不起作用。
- `hook_fn` 必须有如下代码定义：`cell` 是已注册 Cell 对象的信息，`grad_input` 是 Cell 对象的反向输出梯度，`grad_output` 是反向传递给 Cell 对象的梯度。用户可以在 `hook_fn` 中返回 None 或者返回新的梯度。
- `hook_fn` 返回 None 或者新的相应于 `grad_input` 的梯度：`hook_fn(cell, grad_input, grad_output) -> New grad_input or None`。
- 为了避免脚本在切换到图模式时运行失败，不建议在 Cell 对象的 `construct` 函数中调用 `register_backward_hook(hook_fn)`。
- PyNative 模式下，如果在 Cell 对象的 `construct` 函数中调用 `register_backward_hook(hook_fn)`，那么 Cell 对象每次运行都将增加一个 `hook_fn`。

参数：

- **hook_fn** (function) - 捕获 Cell 对象信息和反向输入，输出梯度的 `hook_fn` 函数。

返回：

返回与 `hook_fn` 函数对应的 `handle` 对象。可通过调用 `handle.remove()` 来删除添加的 `hook_fn` 函数。

异常：

- **TypeError** - 如果 `hook_fn` 不是 Python 函数。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, ops
>>> ms.set_context(mode=ms.PYNATIVE_MODE)
>>> def backward_hook_fn(cell, grad_input, grad_output):
...     print("backward input: ", grad_output)
...     print("backward output: ", grad_input)
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.relu = nn.ReLU()
...         self.handle = self.relu.register_backward_hook(backward_hook_fn)
...
...     def construct(self, x):
...         x = x + x
...         x = self.relu(x)
...         return x
>>> grad = ops.GradOperation(get_all=True)
>>> net = Net()
>>> output = grad(net)(Tensor(np.ones([1]).astype(np.float32)))
backward input: (Tensor(shape=[1], dtype=Float32, value= [ 1.00000000e+00]),)
backward output: (Tensor(shape=[1], dtype=Float32, value= [ 1.00000000e+00]),)
>>> print(output)
(Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00]),)
```

register_backward_pre_hook(hook_fn)

设置 Cell 对象的反向 pre_hook 函数。

说明:

- `register_backward_pre_hook(hook_fn)` 在图模式下, 或者在 PyNative 模式下使用 `jit` 装饰器功能时不起作用。
- `hook_fn` 必须有如下代码定义: `cell` 是已注册 Cell 对象的信息, `grad_output` 是反向传递给 Cell 对象的梯度。用户可以在 `hook_fn` 中返回 `None` 或者返回新的梯度。
- `hook_fn` 返回 `None` 或者新的相应于 `grad_output` 的梯度: `hook_fn(cell, grad_output) -> New grad_output or None`。
- `register_backward_pre_hook(hook_fn)` 在 Python 环境中运行。为了避免脚本在切换到图模式时运行失败, 不建议在 Cell 对象的 `construct` 函数中调用 `register_backward_pre_hook(hook_fn)`。
- PyNative 模式下, 如果在 Cell 对象的 `construct` 函数中调用 `register_backward_pre_hook(hook_fn)`, 那么 Cell 对象每次运行都将增加一个 `hook_fn`。

参数:

- **hook_fn** (function) - 捕获 Cell 对象信息和反向输入梯度的 `hook_fn` 函数。

返回:

返回与 `hook_fn` 函数对应的 `handle` 对象。可通过调用 `handle.remove()` 来删除添加的 `hook_fn` 函数。

异常:

- **TypeError** - 如果 `hook_fn` 不是 Python 函数。

样例:

```

>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, ops
>>> ms.set_context(mode=ms.PYNATIVE_MODE)
>>> def backward_pre_hook_fn(cell, grad_output):
...     print("backward input: ", grad_output)
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.relu = nn.ReLU()
...         self.handle = self.relu.register_backward_pre_hook(backward_pre_hook_fn)
...
...     def construct(self, x):
...         x = x + x
...         x = self.relu(x)
...         return x
>>> grad = ops.GradOperation(get_all=True)
>>> net = Net()
>>> output = grad(net)(Tensor(np.ones([1]).astype(np.float32)))
backward input: (Tensor(shape=[1], dtype=Float32, value= [ 1.00000000e+00]),)
>>> print(output)
(Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00]),)

```

register_buffer (*name: str, tensor: Optional[Tensor], persistent: bool = True*)

在 Cell 添加一个缓冲区 *buffer*。

这通常用于注册不应被视为模型参数的缓冲区。例如，BatchNorm 的 *running_mean* 不是参数，而是 Cell 状态的一部分。默认情况下，缓冲区是持久的，将与参数一起保存。可以通过将 *persistent* 设置为 *False* 来更改此行为。持久缓冲区和非持久缓冲区之间的唯一区别是后者不会成为此 Cell 的 *state_dict* 的一部分。

可以使用指定的名称将缓冲区作为属性访问。

参数：

- **name** (str) - 缓冲区的名字。可以使用给定的名称访问此 Cell 的缓冲区。
- **tensor** (Tensor) - 待注册的缓冲区。如果为 *None*，则此 Cell 的 *state_dict* 不会包括该缓冲区。
- **persistent** (bool, 可选) - 缓冲区是否是此 Cell 的 *state_dict* 的一部分。默认 *True*。

样例：

```

>>> import mindspore
...
>>> class Net(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer0", mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return x + self.net_buffer
...
>>> net = Net()
>>> net.register_buffer("buffer0", mindspore.tensor([4, 5, 6]))
>>> print(net.buffer0)
[4 5 6]

```

register_forward_hook (*hook_fn*)

设置 Cell 对象的正向 hook 函数。

说明:

- `register_forward_hook(hook_fn)` 在图模式下, 或者在 PyNative 模式下使用 `jit` 装饰器功能时不起作用。
- `hook_fn` 必须有如下代码定义。`cell` 是已注册 Cell 对象。`inputs` 是网络正向传播时 Cell 对象的输入数据。`outputs` 是网络正向传播时 Cell 对象的输出数据。用户可以在 `hook_fn` 中打印数据或者返回新的输出数据。
- `hook_fn` 返回新的输出数据或者 None: `hook_fn(cell, inputs, outputs) -> New outputs or None`。
- 为了避免脚本在切换到图模式时运行失败, 不建议在 Cell 对象的 `construct` 函数中调用 `register_forward_hook(hook_fn)`。
- PyNative 模式下, 如果在 Cell 对象的 `construct` 函数中调用 `register_forward_hook(hook_fn)`, 那么 Cell 对象每次运行都将增加一个 `hook_fn`。

参数:

- **hook_fn** (function) - 捕获 Cell 对象信息和正向输入, 输出数据的 `hook_fn` 函数。

返回:

返回与 `hook_fn` 函数对应的 `handle` 对象。可通过调用 `handle.remove()` 来删除添加的 `hook_fn` 函数。

异常:

- **TypeError** - 如果 `hook_fn` 不是 Python 函数。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, ops
>>> ms.set_context(mode=ms.PYNATIVE_MODE)
>>> def forward_hook_fn(cell, inputs, output):
...     print("forward inputs: ", inputs)
...     print("forward output: ", output)
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.mul = nn.MatMul()
...         self.handle = self.mul.register_forward_hook(forward_hook_fn)
...
...     def construct(self, x, y):
...         x = x + x
...         x = self.mul(x, y)
...         return x
>>> grad = ops.GradOperation(get_all=True)
>>> net = Net()
>>> output = grad(net)(Tensor(np.ones([1]).astype(np.float32)), Tensor(np.ones([1]).
↪astype(np.float32)))
forward inputs: (Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00]), ↪
↪Tensor(shape=[1],
↪dtype=Float32, value= [ 1.00000000e+00]))
forward output: 2.0
>>> print(output)
(Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00]), Tensor(shape=[1], ↪
↪dtype=Float32,
↪value= [ 2.00000000e+00]))
```

register_forward_pre_hook(hook_fn)

设置 Cell 对象的正向 pre_hook 函数。

说明:

- `register_forward_pre_hook(hook_fn)` 在图模式下, 或者在 PyNative 模式下使用 `jit` 装饰器功能时不起作用。
- `hook_fn` 必须有如下代码定义。`cell` 是已注册 Cell 对象。`inputs` 是网络正向传播时 Cell 对象的输入数据。用户可以在 `hook_fn` 中打印输入数据或者返回新的输入数据。
- `hook_fn` 返回新的输入数据或者 None: `hook_fn(cell, inputs) -> New inputs or None`。
- 为了避免脚本在切换到图模式时运行失败, 不建议在 Cell 对象的 `construct` 函数中调用 `register_forward_pre_hook(hook_fn)`。
- PyNative 模式下, 如果在 Cell 对象的 `construct` 函数中调用 `register_forward_pre_hook(hook_fn)`, 那么 Cell 对象每次运行都将增加一个 `hook_fn`。

参数:

- **hook_fn** (function) - 捕获 Cell 对象信息和正向输入数据的 `hook_fn` 函数。

返回:

返回与 `hook_fn` 函数对应的 `handle` 对象。可通过调用 `handle.remove()` 来删除添加的 `hook_fn` 函数。

异常:

- **TypeError** - 如果 `hook_fn` 不是 Python 函数。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, ops
>>> ms.set_context(mode=ms.PYNATIVE_MODE)
>>> def forward_pre_hook_fn(cell, inputs):
...     print("forward inputs: ", inputs)
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.mul = nn.MatMul()
...         self.handle = self.mul.register_forward_pre_hook(forward_pre_hook_fn)
...
...     def construct(self, x, y):
...         x = x + x
...         x = self.mul(x, y)
...         return x
>>> grad = ops.GradOperation(get_all=True)
>>> net = Net()
>>> output = grad(net)(Tensor(np.ones([1]).astype(np.float32)), Tensor(np.ones([1]).
↪astype(np.float32)))
forward inputs: (Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00]), ↪
↪Tensor(shape=[1],
dtype=Float32, value= [ 1.00000000e+00]))
>>> print(output)
```

(下页继续)

```
(Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00]), Tensor(shape=[1],  
dtype=Float32,  
value= [ 2.00000000e+00]))
```

register_load_state_dict_post_hook(hook)

为 `mindspore.nn.Cell.load_state_dict()` 方法注册一个后钩子。

它应该具有以下签名:

`hook(cell, incompatible_keys) -> None`

参数 `cell` 是此钩子注册的当前 `cell`, 参数 `incompatible_keys` 是一个 `NamedTuple`, 由属性 `missing_keys` 和 `unexpected_keys` 组成。`missing_keys` 是包含缺失键的 `list`, 而 `unexpected_keys` 是包含意外键的 `list`。

请注意, 正如预期的那样, 在使用 `strict=True` 调用: `func: load_state_dict` 时执行的检查会受到钩子对 `missing_keys` 或 `unexpected_keys` 所做修改的影响。当 `strict=True` 时, 添加任何一组键都会导致抛出错误, 而清除缺失和意外的键将避免错误。

参数:

- **hook** (Callable) - 在调用 `load_state_dict` 之后执行的钩子。

返回:

一个句柄, 可以通过调用 `handle.remove()` 来移除已添加的钩子。

register_load_state_dict_pre_hook(hook)

为 `mindspore.nn.Cell.load_state_dict()` 方法注册一个预钩子。

它应该具有以下签名:

`hook(cell, state_dict, prefix, local_metadata, strict, missing_keys, expected_keys, error_msgs) -> None`

注册的钩子可以就地修改 `state_dict`。

参数:

- **hook** (Callable) - 在调用 `load_state_dict` 之前执行的钩子。

返回:

一个句柄, 可以通过调用 `handle.remove()` 来移除已添加的钩子。

register_state_dict_post_hook(hook)

为 `mindspore.nn.Cell.state_dict()` 方法注册一个后钩子。

它应该具有以下签名:

`hook(cell, state_dict, prefix, local_metadata) -> None`

注册的钩子可用于在调用 `state_dict` 之后执行后处理。

参数:

- **hook** (Callable) - 在调用 `state_dict` 之后执行的钩子。

返回:

一个句柄, 可以通过调用 `handle.remove()` 来移除已添加的钩子。

register_state_dict_pre_hook(hook)

为 `mindspore.nn.Cell.state_dict()` 方法注册一个预钩子。

它应该具有以下签名:

`hook(cell, prefix, keep_vars) -> None`

注册的钩子可用于在调用 `state_dict` 之前执行预处理。

参数:

- **hook** (Callable) - 在调用 `state_dict` 之前执行的钩子。

返回:

一个句柄, 可以通过调用 `handle.remove()` 来移除已添加的钩子。

样例:

```
>>> import mindspore
...
...
>>> class NetA(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer_a", mindspore.tensor([1, 2, 3]))
...         self.param_a = mindspore.Parameter(mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return x + self.buffer_a + self.param_a
...
...
>>> def _add_extra_param(cell, prefix, keep_vars):
...     cell._params["extra_param"] = mindspore.Parameter(mindspore.tensor([4, 5, 6]))
...
...
>>> net = NetA()
>>> handle = net.register_state_dict_pre_hook(_add_extra_param)
>>> net_state_dict = net.state_dict()
>>> handle.remove()
>>> print("extra_param" in net_state_dict)
True
```

remove_redundant_parameters()

删除冗余参数。

这个接口通常不需要显式调用。

run_construct (*cast_inputs*, *kwargs*)

运行 `construct` 方法。

说明: 该函数已经弃用, 将会在未来版本中删除。不推荐使用此函数。

参数:

- **cast_inputs** (tuple) - Cell 的输入。
- **kwargs** (dict) - 关键字参数。

返回:

Cell 的输出。

set_boost (*boost_type*)

为了提升网络性能, 可以配置 `boost` 内的算法让框架自动使能该算法来加速网络训练。

请确保 `boost_type` 所选择的算法在 `algorithm library` 算法库中。

说明：部分加速算法可能影响网络精度，请谨慎选择。

参数：

- **boost_type** (str) - 加速算法。

返回：

Cell 类型，Cell 本身。

异常：

- **ValueError** - 如果 *boost_type* 不在 boost 算法库内。

set_broadcast_flag (*mode=True*)

设置该 Cell 的参数广播模式。

参数：

- **mode** (bool) - 指定当前模式是否进行参数广播。默认值：True。

set_comm_fusion (*fusion_type, recurse=True*)

为 Cell 中的参数设置融合类型。请参考 [mindspore.Parameter.comm_fusion](#) 的描述。

说明：当函数被多次调用时，此属性值将被重写。

参数：

- **fusion_type** (int) - Parameter 的 *comm_fusion* 属性的设置值。
- **recurse** (bool) - 是否递归地设置子 Cell 的可训练参数。默认值：True。

set_data_parallel ()

在非自动策略搜索的情况下，如果此 Cell 的所有算子（包括此 Cell 内含嵌套的 cell）未指定并行策略，则将为这些基本算子设置为数据并行策略。

说明：仅在图模式，使用 `auto_parallel_context = ParallelMode.AUTO_PARALLEL` 生效。

样例：

```
>>> import mindspore.nn as nn
>>> net = nn.Dense(3, 4)
>>> net.set_data_parallel()
```

set_extra_state (*state: Any*)

设置加载的 *state_dict* 中包含的额外状态。

此方法由 *load_state_dict* 调用，以处理 *state_dict* 中的任何额外状态。如果您的 Cell 需要在 *state_dict* 中存储额外状态，请实现此方法及相应的 *get_extra_state* 方法。

参数：

- **state** (dict) - *state_dict* 的额外状态。

set_grad (*requires_grad=True*)

Cell 的梯度设置。

参数:

- **requires_grad** (bool) - 指定网络是否需要梯度, 如果为 True, PyNative 模式下 Cell 将构建反向网络。默认值: True。

返回:

Cell 类型, Cell 本身。

set_inputs (*inputs, **kwargs)

设置编译计算图所需的输入。输入数量需与数据集数量一致。若使用 Model 接口, 请确保所有传入 Model 的网络和损失函数都配置了 set_inputs。输入 Tensor 的 shape 可以为动态或静态。

说明: 有两种配置模式:

- 全量配置模式: 输入将被用作图编译时的完整编译参数。
- 增量配置模式: 输入被配置到 Cell 的部分输入上, 这些输入将替换图编译对应位置上的参数。

只能传入 inputs 和 kwargs 的其中一个。inputs 用于全量配置模式, kwargs 用于增量配置模式。

参数:

- **inputs** (tuple) - 全量配置模式的参数。
- **kwargs** (dict) - 增量配置模式的参数。可设置的 key 值为 *self.construct* 中定义的参数名。

警告: 这是一个实验性 API, 后续可能修改或删除。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, Tensor
>>>
>>> class ReluNet(nn.Cell):
...     def __init__(self):
...         super(ReluNet, self).__init__()
...         self.relu = nn.ReLU()
...     def construct(self, x):
...         return self.relu(x)
>>>
>>> net = ReluNet()
>>> input_dyn = Tensor(shape=[3, None], dtype=ms.float32)
>>> net.set_inputs(input_dyn)
>>> input = Tensor(np.random.random([3, 10]), dtype=ms.float32)
>>> output = net(input)
>>>
>>> net2 = ReluNet()
>>> net2.set_inputs(x=input_dyn)
>>> output = net2(input)
```

set_jit_config (jit_config)

为 Cell 设置编译时所使用的 JitConfig 配置项。

参数:

- **jit_config** (JitConfig) - Cell 的 Jit 配置信息。详情请参考 *mindspore.JitConfig*。

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.relu = nn.ReLU()
...
...     def construct(self, x):
...         x = self.relu(x)
...         return x
>>> net = Net()
>>> jitconfig = ms.JitConfig()
>>> net.set_jit_config(jitconfig)
```

set_param_ps (*recurse=True, init_in_server=False*)

设置可训练参数是否由参数服务器更新, 以及是否在服务器上初始化可训练参数。

说明: 只在运行的任务处于参数服务器模式时有效。只支持在图模式下调用。

参数:

- **recurse** (bool) - 是否设置子网络的可训练参数。默认值: True。
- **init_in_server** (bool) - 是否在服务器上初始化由参数服务器更新的可训练参数。默认值: False。

set_train (*mode=True*)

将 Cell 设置为训练模式。

设置当前 Cell 和所有子 Cell 的训练模式。对于训练和预测具有不同结构的网络层 (如 *BatchNorm*), 将通过这个属性区分分支。如果设置为 True, 则执行训练分支, 否则执行另一个分支。

说明: 当执行 `mindspore.train.Model.train()` 的时候, 框架会默认调用 `Cell.set_train(True)`。当执行 `mindspore.train.Model.eval()` 的时候, 框架会默认调用 `Cell.set_train(False)`。

参数:

- **mode** (bool) - 指定模型是否为训练模式。默认值: True。

返回:

Cell 类型, Cell 本身。

教程样例:

- 模型训练 - 训练与评估实现

shard (*in_strategy, out_strategy=None, parameter_plan=None, device='Ascend', level=0*)

指定输入/输出 Tensor 的分布策略, 通过其余算子的策略推导得到。在 PyNative 模式下, 可以利用此方法指定某个 Cell 以图模式进行分布式执行。在图模式下, 可以利用此方法设置某个模块的分布式切分策略, 未设置的会自动通过策略传播方式配置。*in_strategy/out_strategy* 需要为元组类型, 其中的每一个元素指定对应的输入/输出的 Tensor 分布策略, 可参考: `mindspore.ops.Primitive.shard()` 的描述。其余算子的并行策略由输入输出指定的策略推导得到。

说明：调用该方法后，并行模式 (`parallel_mode`) 会自动设置为“`auto_parallel`”且搜索模式 (`search_mode`) 自动设置为“`sharding_propagation`”。如果输入含有 `Parameter`，其对应的策略应该在 `in_strategy` 里设置。

参数：

- **in_strategy** (tuple) - 指定各输入的切分策略，输入元组的每个元素元组，元组即具体指定输入每一维的切分策略。
- **out_strategy** (Union[None, tuple]) - 指定各输出的切分策略，用法同 `in_strategy`，目前未使能。默认值：None。
- **parameter_plan** (Union[dict, None]) - 指定各参数的切分策略，传入字典时，键是 `str` 类型的参数名，值是一维整数 `tuple` 表示相应的切分策略，如果参数名错误或对应参数已经设置了切分策略，该参数的设置会被跳过。默认值：None。
- **device** (str) - 指定执行设备，可以为 ["CPU", "GPU", "Ascend"] 中任意一个，目前未使能。默认值："Ascend"。
- **level** (int) - 指定搜索切分策略的目标函数，即是最大化计算通信比、最小化内存消耗、最大化执行速度等。可以为 [0, 1, 2] 中任意一个，默认值：0。目前仅支持最大化计算通信比，其余模式未使能。

返回：

Function，返回一个在自动并行流程下执行的函数。

样例：

```
>>> import mindspore.nn as nn
>>>
>>> class Block(nn.Cell):
...     def __init__(self):
...         self.dense1 = nn.Dense(10, 10)
...         self.relu = nn.ReLU()
...         self.dense2 = nn.Dense2(10, 10)
...     def construct(self, x):
...         x = self.relu(self.dense2(self.relu(self.dense1(x))))
...         return x
>>>
>>> class example(nn.Cell):
...     def __init__(self):
...         self.block1 = Block()
...         self.block2 = Block()
...         self.block2_shard = self.block2.shard(in_strategy=((2, 1)),
...                                                 parameter_plan={'self.block2.shard.dense1.
...weight': (4, 1)})
...     def construct(self, x):
...         x = self.block1(x)
...         x = self.block2_shard(x)
...         return x
```

state_dict (*args, destination=None, prefix="", keep_vars=False)

返回一个包含对 `Cell` 整个状态的引用的字典。

参数和持久缓冲区（例如运行平均值）都包括在内。键是相应的参数和缓冲区名称。设置为 `None` 的参数和缓冲区不包括在内。

说明：返回的对象是一个浅拷贝。它包含对该 Cell 的参数和缓冲区的引用。

警告：

- 目前 `state_dict()` 还按顺序接受 `destination`、`prefix` 和 `keep_vars` 的位置参数。但是这即将被弃用，关键字参数将在未来的版本中强制执行。
- 请避免使用参数 `destination`，因为它不是为最终用户设计的。

参数：

- **destination** (dict, 可选) - 如果提供，Cell 的状态将更新到此字典中，并返回相同的对象。否则，将创建并返回 `OrderedDict`。默认 `None`。
- **prefix** (str, 可选) - 添加到参数和缓冲区名称的前缀，用于组成 `state_dict` 中的键。默认 `""`。
- **keep_vars** (bool, 可选) - 状态字典返回值是否为拷贝。默认 `False`，返回引用。

返回：

Dict，包含整个 Cell 状态的字典。

样例：

```
>>> import mindspore
>>> class Model(mindspore.nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.register_buffer("buffer_a", mindspore.tensor([4, 5, 6]))
...         self.param_a = mindspore.Parameter(mindspore.tensor([1, 2, 3]))
...
...     def construct(self, x):
...         return x + self.buffer_a + self.param_a
...
...
>>> model = Model()
>>> print(model.state_dict())
OrderedDict([('param_a', Parameter (name=param_a, shape=(3,), dtype=Int64, requires_
grad=True)), \
('buffer_a', Tensor(shape=[3], dtype=Int64, value= [4, 5, 6]))])
```

to_float(dst_type)

在 Cell 和所有子 Cell 的输入上添加类型转换，以使用特定的浮点类型运行。

如果 `dst_type` 是 `mindspore.dtype.float16`，Cell 的所有输入 (包括作为常量的 `input`、`Parameter`、`Tensor`) 都会被转换为 `float16`。请参考 `mindspore.amp.build_train_network()` 的源代码中的用法。

说明：多次调用将产生覆盖。

参数：

- **dst_type** (mindspore.dtype) - Cell 转换为 `dst_type` 类型运行。`dst_type` 可以是 `mindspore.dtype.float16`、`mindspore.dtype.float32` 或者 `mindspore.dtype.bfloat16`。

返回：

Cell 类型, Cell 本身。

异常:

- **ValueError** - 如果 *dst_type* 不是 *mindspore.dtype.float32* , 不是 *mindspore.dtype.float16* , 也不是 *mindspore.dtype.bfloat16* 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.nn as nn
>>> from mindspore import dtype as mstype
>>>
>>> net = nn.Conv2d(120, 240, 4, has_bias=False, weight_init='normal')
>>> net.to_float(mstype.float16)
Conv2d(input_channels=120, output_channels=240, kernel_size=(4, 4), stride=(1, 1), pad_
↪mode=same,
padding=0, dilation=(1, 1), group=1, has_bias=False, weight_init=normal, bias_init=None, ↪
↪format=NCHW)
```

trainable_params (*recurse=True*)

返回 Cell 的一个可训练参数的列表。

参数:

- **recurse** (bool) - 是否递归地包含当前 Cell 的所有子 Cell 的可训练参数。默认值: True 。

返回:

List 类型, 可训练参数列表。

教程样例:

- [模型训练 - 优化器](#)

untrainable_params (*recurse=True*)

返回 Cell 的一个不可训练参数的列表。

参数:

- **recurse** (bool) - 是否递归地包含当前 Cell 的所有子 Cell 的不可训练参数。默认值: True 。

返回:

List 类型, 不可训练参数列表。

update_cell_prefix ()

递归地更新所有子 Cell 的 *param_prefix* 。

在调用此方法后, 可以通过 Cell 的 *param_prefix* 属性获取该 Cell 的所有子 Cell 的名称前缀。

update_cell_type (*cell_type*)

量化感知训练网络场景下, 更新当前 Cell 的类型。

此方法将 Cell 类型设置为 *cell_type* 。

参数:

- **cell_type** (str) - 被更新的类型, *cell_type* 可以是 "quant" 或 "second-order" 。

update_parameters_name (*prefix=""*, *recurse=True*)

给网络参数名称添加 *prefix* 前缀字符串。

参数:

- **prefix** (str) - 前缀字符串。默认值: ''。
- **recurse** (bool) - 是否递归地包含所有子 Cell 的参数。默认值: True。

3.1.2 mindspore.nn.GraphCell

class mindspore.nn.GraphCell (graph, params_init=None, obf_random_seed=None)

运行从 MindIR 加载的计算图。

此功能仍在开发中。目前 GraphCell 不支持修改图结构, 在导出 MindIR 时只能使用 shape 和类型与输入相同的数据。

参数:

- **graph** (FuncGraph) - 从 MindIR 加载的编译图。
- **params_init** (dict) - 需要在图中初始化的参数。key 为参数名称, 类型为字符串, value 为 Tensor 或 Parameter。如果参数名在图中已经存在, 则更新其值; 如果不存在, 则忽略。默认值: None。
- **obf_random_seed** (Union[int, None]) - 用于动态混淆保护的混淆随机种子, 目前动态结构混淆已不支持。

异常:

- **NotImplementedError** - 动态结构混淆已不支持。
- **TypeError** - 如果图不是 FuncGraph 类型。
- **TypeError** - 如果 params_init 不是字典。
- **TypeError** - 如果 params_init 的 key 不是字符串。
- **TypeError** - 如果 params_init 的 value 既不是 Tensor 也不是 Parameter。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> from mindspore import context
>>> context.set_context(mode=context.GRAPH_MODE)
>>> net = nn.Conv2d(1, 1, kernel_size=3, weight_init="ones")
>>> input = Tensor(np.ones([1, 1, 3, 3]).astype(np.float32))
>>> ms.export(net, input, file_name="net", file_format="MINDIR")
>>> graph = ms.load("net.mindir")
>>> net = nn.GraphCell(graph)
>>> output = net(input)
>>> print(output)
[[[4. 6. 4.]
  [6. 9. 6.]
  [4. 6. 4.]]]]
```


3.1.3 mindspore.nn.LossBase

class mindspore.nn.LossBase (*reduction='mean'*)

损失函数的基类。

自定义损失函数时应重写 *construct*，并使用方法 *self.get_loss* 将 *reduction* 应用于 *loss* 计算。

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的（加权）平均值。
 - "sum": 计算输出元素的总和。

异常：

- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import ops, Tensor, nn
>>> import numpy as np
>>>
>>> class Net(nn.LossBase):
...     def __init__(self, reduction='mean'):
...         super(Net, self).__init__(reduction)
...         self.abs = ops.Abs()
...
...     def construct(self, logits, labels):
...         x = self.abs(logits - labels)
...         output = self.get_loss(x)
...         axis = self.get_axis(x)
...         return output, axis
>>> net = Net()
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output, axis = net(logits, labels)
>>> print(output)
0.33333334
>>> print(axis)
(0,)
>>> # Case 2: logits.shape = labels.shape = (3, 3)
>>> logits = Tensor(np.array([[1, 2, 3], [1, 2, 3], [1, 2, 3]]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 2, 2], [1, 2, 3], [1, 2, 3]]), mindspore.float32)
>>> output, axis = net(logits, labels)
>>> print(output)
0.11111111
>>> print(axis)
(0, 1)
```

get_axis(x)

获取输入的轴范围。

参数:

- **x** (Tensor) - 任何 shape 的 Tensor。

get_loss (*x*, *weights=1.0*)

计算加权损失。

参数:

- **x** (Tensor) - shape 为 (*N*, *) 的输入 Tensor, 其中 * 表示任意数量的附加维度。
- **weights** (Union[float, Tensor], 可选) - 权重。当 *weights* 为 Tensor 时, 其 rank 必须为 0 或与输入相同, 并且必须可广播到输入 (即, 所有维度必须为 1, 或与相应输入的维度相同)。默认值: 1.0。

返回:

返回加权损失。

3.1.4 mindspore.nn.Optimizer

class mindspore.nn.Optimizer (*learning_rate*, *parameters*, *weight_decay=0.0*, *loss_scale=1.0*)

用于参数更新的优化器基类。不要直接使用这个类, 请实例化它的一个子类。详见 [优化器](#)。

优化器支持参数分组。当参数分组时, 每组参数均可配置不同的学习率 (*lr*)、权重衰减 (*weight_decay*) 和梯度中心化 (*grad_centralization*) 策略。

说明:

- 在参数未分组时, 优化器配置的 *weight_decay* 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 *weight_decay*。若未配置, 则该组网络参数使用优化器中配置的 *weight_decay*。

参数:

- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule]) -
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中, 优化器将使用步数 (*step*) 作为输入, 调用 [LearningRateSchedule](#) 实例来计算当前学习率。
- **parameters** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”:
 - **params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - **lr** - 可选。如果键中存在“lr”, 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在“weight_decay”, 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 *weight_decay* 作为权重衰减值。

- **grad_centralization** - 可选。如果键中存在“grad_centralization”，则使用对应的值，该值必须为布尔类型。如果没有，则认为 *grad_centralization* 为 False。该参数仅适用于卷积层。
- **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **weight_decay** (Union[float, int], 可选) - 权重衰减的整数或浮点值。必须等于或大于 0。如果 *weight_decay* 是整数，它将被转换为浮点数。默认值：0.0。
- **loss_scale** (float, 可选) - 梯度缩放系数，必须大于 0.0。如果 *loss_scale* 是整数，它将被转换为浮点数。通常使用默认值，仅当训练时使用了 *FixedLossScaleManager*，且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 False 时，此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息，请参阅 *mindspore.amp.FixedLossScaleManager*。默认值：1.0。

异常：

- **TypeError** - *learning_rate* 不是 int、float、Tensor、Iterable 或 LearningRateSchedule。
- **TypeError** - *parameters* 的元素不是 Parameter 或字典。
- **TypeError** - *loss_scale* 不是 float。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **ValueError** - *loss_scale* 小于或等于 0。
- **ValueError** - *weight_decay* 小于 0。
- **ValueError** - *learning_rate* 是一个 Tensor，但是 Tensor 的维度大于 1。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import nn
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, ops, Tensor
>>>
>>> class MyMomentum(nn.Optimizer):
...     def __init__(self, params, learning_rate, momentum=0.9):
...         super(MyMomentum, self).__init__(learning_rate, params)
...         self.moments = self.parameters.clone(prefix="moments", init="zeros")
...         self.momentum = momentum
...         self.opt = ops.ApplyMomentum()
...
...     def construct(self, gradients):
...         params = self.parameters
...         lr = self.get_lr()
...         gradients = self.flatten_gradients(gradients)
...         gradients = self.decay_weight(gradients)
...         gradients = self.gradients_centralization(gradients)
...         gradients = self.scale_grad(gradients)
...
...         success = None
...         for param, mom, grad in zip(params, self.moments, gradients):
```

(下页继续)

(续上页)

```

...         success = self.opt(param, mom, lr, grad, self.momentum)
...         return success
>>>
>>> net = nn.Dense(2, 3)
>>> loss_fn = nn.MAELoss()
>>> opt = MyMomentum(net.trainable_params(), 0.01)
>>>
>>> device_target = opt.target
>>> opt_unique = opt.unique
>>> weight_decay_value = opt.get_weight_decay()
>>>
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>>
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, opt.parameters, has_aux=True)
>>>
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     opt(grads)
...     return loss
>>>
>>> data = Tensor(np.random.rand(4, 10, 2), mindspore.dtype.float32)
>>> label = Tensor(np.random.rand(4, 10, 3), mindspore.dtype.float32)
>>> train_step(data, label)

```

broadcast_params (*optim_result*)

按参数组的顺序进行参数广播。

参数：

- **optim_result** (bool) - 参数更新结果。该输入用来保证参数更新完成后才执行参数广播。

返回：

被广播的参数组。

decay_weight (*gradients*)

衰减权重。

一种减少深度学习神经网络模型过拟合的方法。继承 `mindspore.nn.Optimizer` 自定义优化器时，可调用该接口进行权重衰减。

参数：

- **gradients** (tuple[`Tensor`]) - 网络参数的梯度，形状（shape）与网络参数相同。

返回：

tuple[`Tensor`]，衰减权重后的梯度。

flatten_gradients (*gradients*)

如果网络参数已经使用了连续内存，则梯度也将按数据类型分组使用连续内存。

一种网络参数和梯度都使用连续内存的性能优化方法。继承 `mindspore.nn.Optimizer` 自定义优化器时，需调用该接口使能连续内存优化。

参数：

- **gradients** (tuple[`Tensor`]) - 网络参数的梯度，形状（shape）与网络参数相同。

返回:

`tuple[Tensor]`, 如果网络参数使用了连续内存, 则返回按数据类型分组使用连续内存后的梯度, 否则原样返回输入的梯度。

get_lr()

优化器调用该接口获取当前步骤 (step) 的学习率。继承 `mindspore.nn.Optimizer` 自定义优化器时, 可在参数更新前调用该接口获取学习率。

返回:

float, 当前步骤的学习率。

get_lr_parameter(param)

用于在使用网络参数分组功能, 且为不同组别配置不同的学习率时, 获取指定参数的学习率。

参数:

- **param** (`Union[Parameter, list[Parameter]]`) - *Parameter* 或 *Parameter* 列表。

返回:

单个 *Parameter* 或 *Parameter* 列表。如果使用了动态学习率, 返回用于计算学习率的 *LearningRateSchedule* 或 *LearningRateSchedule* 列表。

样例:

```
>>> from mindspore import nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'lr': 0.05},
...                  {'params': no_conv_params, 'lr': 0.01}]
>>> optim = nn.Momentum(group_params, learning_rate=0.1, momentum=0.9, weight_decay=0.0)
>>> conv_lr = optim.get_lr_parameter(conv_params)
>>> print(conv_lr[0].asnumpy())
0.05
```

get_weight_decay()

优化器调用该接口获取当前步骤 (step) 的 weight decay 值。继承 `mindspore.nn.Optimizer` 自定义优化器时, 可在参数更新前调用该接口获取 weight decay 值。

返回:

float, 当前步骤的 weight decay 值。

gradients_centralization(gradients)

梯度中心化。

一种优化卷积层参数以提高深度学习神经网络模型训练速度的方法。继承 `mindspore.nn.Optimizer` 自定义优化器时, 可调用该接口进行梯度中心化。

参数:

- **gradients** (`tuple[Tensor]`) - 网络参数的梯度, 形状 (shape) 与网络参数相同。

返回:

`tuple[Tensor]`, 梯度中心化后的梯度。

scale_grad(gradients)

用于在混合精度场景还原梯度。

继承 `mindspore.nn.Optimizer` 自定义优化器时, 可调用该接口还原梯度。

参数:

- **gradients** (tuple[`Tensor`]) - 网络参数的梯度, 形状 (shape) 与网络参数相同。

返回:

tuple[`Tensor`], 还原后的梯度。

property target
该属性用于指定在主机 (host) 上还是设备 (device) 上更新参数。输入类型为 `str`, 只能是 'CPU'、'Ascend' 或 'GPU'。

property unique
该属性表示是否在优化器中进行梯度去重, 通常用于稀疏网络。如果梯度是稀疏的, 则设置为 `True`。如果前向稀疏网络已对权重去重, 即梯度是稠密的, 则设置为 `False`。未进行任何配置时, 默认为 `True`。

3.2 容器

接口名	概述	支持平台
<code>mindspore.nn.CellDict</code>	构造 <code>Cell</code> 字典。	Ascend GPU CPU
<code>mindspore.nn.CellList</code>	构造 <code>Cell</code> 列表。	Ascend GPU CPU
<code>mindspore.nn.SequentialCell</code>	构造 <code>Cell</code> 顺序容器。	Ascend GPU CPU

3.2.1 mindspore.nn.CellDict

class `mindspore.nn.CellDict (*args, **kwargs)`
构造 `Cell` 字典。关于 `Cell` 的介绍, 可参考 `mindspore.nn.Cell`。

`CellDict` 可以像普通 Python 字典一样使用。

参数:

- **args** (iterable, 可选) - 一个可迭代对象, 通过它可迭代若干键值对 (key, `Cell`), 其中键值对的类型为 (`string`, `Cell`); 或者是一个从 `string` 到 `Cell` 的映射 (字典)。`Cell` 的类型不能为 `CellDict`, `CellList` 或者 `SequentialCell`。`key` 不能与类 `Cell` 中的属性重名, 不能包含 '.', 不能是一个空字符串。通过类型为 `string` 的键可以在 `CellDict` 中查找其对应的 `Cell`。
- **kwargs** (dict) - 为待扩展的关键字参数预留。

支持平台:

Ascend GPU CPU

样例:

```
>>> import collections
>>> from collections import OrderedDict
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, nn
>>>
>>> cell_dict = nn.CellDict({'conv': nn.Conv2d(10, 6, 5),
...                        'relu': nn.ReLU(),
...                        'max_pool2d': nn.MaxPool2d(kernel_size=4, stride=4)})
>>> print(len(cell_dict))
```

(下页继续)

(续上页)

```

3
>>> cell_dict.clear()
>>> print(len(cell_dict))
0
>>> ordered_cells = OrderedDict([('conv', nn.Conv2d(10, 6, 5, pad_mode='valid')),
...                               ('relu', nn.ReLU()),
...                               ('max_pool2d', nn.MaxPool2d(kernel_size=2, stride=2))])
>>> cell_dict.update(ordered_cells)
>>> x = Tensor(np.ones([1, 10, 6, 10]), ms.float32)
>>> for cell in cell_dict.values():
...     x = cell(x)
>>> print(x.shape)
(1, 6, 1, 3)
>>> x = Tensor(np.ones([1, 10, 6, 10]), ms.float32)
>>> for item in cell_dict.items():
...     x = item[1](x)
>>> print(x.shape)
(1, 6, 1, 3)
>>> print(cell_dict.keys())
odict_keys(['conv', 'relu', 'max_pool2d'])
>>> pop_cell = cell_dict.pop('conv')
>>> x = Tensor(np.ones([1, 10, 6, 5]), ms.float32)
>>> x = pop_cell(x)
>>> print(x.shape)
(1, 6, 2, 1)
>>> print(len(cell_dict))
2

```

clear()

移除 CellDict 中的所有 Cell。

items()

返回包含 CellDict 中所有键值对的可迭代对象。

返回:

一个可迭代对象。

keys()

返回包含 CellDict 中所有键的可迭代对象。

返回:

一个可迭代对象。

pop(key)

从 CellDict 中移除键为 *key* 的 Cell，并将这个 Cell 返回。

参数:

- **key** (str) - 从 CellDict 中移除的 Cell 的键。

异常:

- **KeyError** - *key* 对应的 Cell 在 CellDict 中不存在。

update(cells)

使用映射或者可迭代对象中的键值对来更新 CellDict 中已存在的 Cell。

参数:

- **cells** (iterable) - 一个可迭代对象, 通过它可迭代若干键值对 (key, Cell), 其中键值对的类型为 (string, Cell); 或者是一个从 string 到 Cell 的映射 (字典)。Cell 的类型不能为 CellDict、CellList 或者 SequentialCell。key 不能与类 Cell 中的属性重名, 不能包含 '.', 不能是一个空字符串。

说明: 如果 *cells* 是一个 CellDict、一个 OrderedDict 或者是一个包含键值对的可迭代对象, 那么新增元素的顺序在 CellDict 中仍会被保留。

异常:

- **TypeError** - 如果 *cells* 不是一个可迭代对象。
- **TypeError** - 如果 *cells* 中的键值对不是可迭代对象。
- **ValueError** - 如果 *cells* 中键值对的长度不是 2。
- **TypeError** - 如果 *cells* 中的 cell 是 None。
- **TypeError** - 如果 *cells* 中的 cell 的类型不是 Cell。
- **TypeError** - 如果 *cells* 中的 cell 的类型是 CellDict, CellList 或者 SequentialCell。
- **TypeError** - 如果 *cells* 中的 key 的类型不是 String 类型。
- **KeyError** - 如果 *cells* 中的 key 与类 Cell 中的属性重名。
- **KeyError** - 如果 *cells* 中的 key 包含 "."。
- **KeyError** - 如果 *cells* 中的 key 是一个空字符串。

values()

返回包含 CellDict 中所有 Cell 的可迭代对象。

返回:

一个可迭代对象。

3.2.2 mindspore.nn.CellList

class mindspore.nn.CellList (*args, **kwargs)

构造 Cell 列表。关于 Cell 的介绍, 可参考 [Cell](#)。

CellList 可以像普通 Python 列表一样使用, 其包含的 Cell 均已初始化。CellList 包含的 Cell 的类型不能为 CellDict。

参数:

- **args** (list, 可选) - Cell 列表。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>>
>>> conv = ms.nn.Conv2d(100, 20, 3)
>>> bn = ms.nn.BatchNorm2d(20)
>>> relu = ms.nn.ReLU()
>>> cell_ls = ms.nn.CellList([bn])
```

(下页继续)

(续上页)

```

>>> cell_ls.insert(0, conv)
>>> cell_ls.append(relu)
>>> cell_ls.extend([relu, relu])
>>> cell_ls_3 = cell_ls[3]
>>> input1 = ms.Tensor(np.ones([2, 3]), ms.float32)
>>> output = cell_ls_3(input1)
>>> print(output)
[[1. 1. 1.]
 [1. 1. 1.]]

```

append(*cell*)

在列表末尾添加一个 Cell。

参数：

- **cell** (Cell) - 要添加的 Cell。

extend(*cells*)

将 cells 中的 Cell 添加到列表末尾。

参数：

- **cells** (list) - 要添加的 Cell 列表，Cell 的类型不能为 CellDict。

异常：

- **TypeError** - cells 中的元素不是 Cell。

insert(*index, cell*)

在列表中的给定索引之前插入给定的 Cell。

参数：

- **index** (int) - 给定的列表索引。
- **cell** (Cell) - 要插入的 Cell。

3.2.3 mindspore.nn.SequentialCell

class mindspore.nn.SequentialCell(*args)

构造 Cell 顺序容器。关于 Cell 的介绍，可参考 [Cell](#)。

SequentialCell 将按照传入 List 的顺序依次添加 Cell。此外，也支持 OrderedDict 作为构造器传入。

参数：

- **args** (list, OrderedDict) - 仅包含 Cell 子类的列表或有序字典。

输入：

- **x** (Tensor) - Tensor，其 shape 取决于序列中的第一个 Cell。

输出：

Tensor，输出 Tensor，其 shape 取决于输入 *x* 和定义的 Cell 序列。

异常：

- **TypeError** - args 的类型不是列表或有序字典。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>>
>>> conv = nn.Conv2d(3, 2, 3, pad_mode='valid', weight_init="ones")
>>> relu = nn.ReLU()
>>> seq = nn.SequentialCell([conv, relu])
>>> x = Tensor(np.ones([1, 3, 4, 4]), dtype = mindspore.float32)
>>> output = seq(x)
>>> print(output)
[[[27. 27.]
  [27. 27.]]
 [27. 27.]
 [27. 27.]]]
>>> from collections import OrderedDict
>>> d = OrderedDict()
>>> d["conv"] = conv
>>> d["relu"] = relu
>>> seq = nn.SequentialCell(d)
>>> x = Tensor(np.ones([1, 3, 4, 4]), dtype=mindspore.float32)
>>> output = seq(x)
>>> print(output)
[[[27. 27.]
  [27. 27.]]
 [27. 27.]
 [27. 27.]]]
```

append(*cell*)

在容器末尾添加一个 Cell。

参数:

- **cell (Cell)** - 要添加的 Cell。

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>>
>>> conv = nn.Conv2d(3, 2, 3, pad_mode='valid', weight_init="ones")
>>> bn = nn.BatchNorm2d(2)
>>> relu = nn.ReLU()
>>> seq = nn.SequentialCell([conv, bn])
>>> seq.append(relu)
>>> x = Tensor(np.ones([1, 3, 4, 4]), dtype=mindspore.float32)
>>> output = seq(x)
>>> print(output)
[[[26.999863 26.999863]
  [26.999863 26.999863]]
 [26.999863 26.999863]
 [26.999863 26.999863]]]
```

3.3 封装层

接口名	概述	支持平台
<code>mindspore.nn.DistributedGradReducer</code>	分布式优化器。	Ascend GPU
<code>mindspore.nn.DynamicLossScaleUpdateCell</code>	用于动态更新损失缩放系数（loss scale）的神经元。	Ascend GPU
<code>mindspore.nn.FixedLossScaleUpdateCell</code>	固定损失缩放系数的神经元。	Ascend GPU
<code>mindspore.nn.ForwardValueAndGrad</code>	训练网络的封装。	Ascend GPU CPU
<code>mindspore.nn.GetNextSingleOp</code>	用于获取下一条数据的 Cell。	Ascend GPU
<code>mindspore.nn.GradAccumulationCell</code>	将 MiniBatch 切分成更细粒度的 MicroBatch, 用于半自动/全自动并行模式下的梯度累加训练中。	Ascend GPU
<code>mindspore.nn.ParameterUpdate</code>	更新参数的 Cell。	Ascend GPU CPU
<code>mindspore.nn.PipelineCell</code>	将 MiniBatch 切分成更细粒度的 MicroBatch, 用于流水线并行的训练中。	Ascend GPU
<code>mindspore.nn.TimeDistributed</code>	时间序列封装层。	Ascend GPU CPU
<code>mindspore.nn.TrainOneStepCell</code>	训练网络封装类。	Ascend GPU CPU
<code>mindspore.nn.TrainOneStepWithLossScaleCell</code>	使用混合精度功能的训练网络。	Ascend GPU
<code>mindspore.nn.WithEvalCell</code>	封装前向网络和损失函数。	Ascend GPU CPU
<code>mindspore.nn.WithLossCell</code>	包含损失函数的 Cell。	Ascend GPU CPU

3.3.1 mindspore.nn.DistributedGradReducer

class `mindspore.nn.DistributedGradReducer` (*parameters*, *mean=None*, *degree=None*, *fusion_type=1*, *group=GlobalComm.WORLD_COMM_GROUP*)

分布式优化器。

用于数据并行模式中，利用 AllReduce 对所有卡的梯度进行聚合。

参数：

- **parameters** (list) - 需要更新的参数。
- **mean** (bool) - 当 mean 为 True 时，对 AllReduce 之后的梯度求均值。未指定时，使用 `auto_parallel_context` 中的配置 “gradients_mean”。默认值：None。
- **degree** (int) - 平均系数，通常等于设备编号。默认值：None。
- **fusion_type** (int) - AllReduce 算子的融合类型。默认值：1。
- **group** (str) - AllReduce 算子的通信域，若需要自定义通信域，需要调用 `create_group` 接口。默认值：GlobalComm.WORLD_COMM_GROUP。

异常：

- **ValueError** - 如果 degree 不是 int 或小于 0。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在多卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore import Parameter, Tensor, ops, nn
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> ms.reset_auto_parallel_context()
>>> ms.set_auto_parallel_context(parallel_mode=ms.ParallelMode.DATA_PARALLEL)
>>>
>>> class TrainingWrapper(nn.Cell):
...     def __init__(self, network, optimizer, sens=1.0):
...         super(TrainingWrapper, self).__init__(auto_prefix=False)
...         self.network = network
...         self.network.add_flags(defer_inline=True)
...         self.weights = optimizer.parameters
...         self.optimizer = optimizer
...         self.grad = ops.GradOperation(get_by_list=True, sens_param=True)
...         self.sens = sens
...         self.reducer_flag = False
...         self.grad_reducer = None
...         self.parallel_mode = context.get_auto_parallel_context("parallel_mode")
...         self.depend = ops.Depend()
...         if self.parallel_mode in [ms.ParallelMode.DATA_PARALLEL, ms.ParallelMode.HYBRID_
... ↪PARALLEL]:
...             self.reducer_flag = True
...             if self.reducer_flag:
...                 mean = context.get_auto_parallel_context("gradients_mean")
...                 degree = context.get_auto_parallel_context("device_num")
...                 self.grad_reducer = nn.DistributedGradReducer(optimizer.parameters, mean,
... ↪degree)
...
...     def construct(self, *args):
...         weights = self.weights
...         loss = self.network(*args)
...         sens = F.fill(ops.DType()(loss), ops.Shape()(loss), self.sens)
...         grads = self.grad(self.network, weights)(*args, sens)
...         if self.reducer_flag:
...             # apply grad reducer on grads
...             grads = self.grad_reducer(grads)
...         return self.depend(loss, self.optimizer(grads))
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
... ↪float32)),
...                                 name='weight')
...         self.matmul = ops.MatMul()
```

(下页继续)

(续上页)

```

...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output
>>>
>>> size, in_features, out_features = 16, 16, 10
>>> network = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> net_with_loss = nn.WithLossCell(network, loss)
>>> optimizer = nn.Momentum(net_with_loss.trainable_params(), learning_rate=0.1, momentum=0.
    ↪9)
>>> train_cell = TrainingWrapper(net_with_loss, optimizer)
>>> inputs = Tensor(np.ones([size, in_features]).astype(np.float32))
>>> label = Tensor(np.zeros([size, out_features]).astype(np.float32))
>>> grads = train_cell(inputs, label)
>>> print(grads)
256.0

```

3.3.2 mindspore.nn.DynamicLossScaleUpdateCell

class mindspore.nn.DynamicLossScaleUpdateCell (loss_scale_value, scale_factor, scale_window)

用于动态更新损失缩放系数 (loss scale) 的神经元。

使用混合精度功能进行训练时, 初始损失缩放系数值为 *loss_scale_value*。在每个训练步骤中, 当出现溢出时, 通过计算公式 $loss\_scale / scale\_factor$ 减小损失缩放系数。如果连续 *scale_window* step 未溢出, 则将通过 $loss\_scale * scale\_factor$ 增大损失缩放系数。

该类是 *mindspore.amp.DynamicLossScaleManager* 的 *get_update_cell* 方法的返回值。训练过程中, 类 *mindspore.nn.TrainOneStepWithLossScaleCell* 会调用该 Cell 来更新损失缩放系数。

参数:

- **loss_scale_value** (float) - 初始的损失缩放系数。
- **scale_factor** (int) - 增减系数。
- **scale_window** (int) - 未溢出时, 增大损失缩放系数的最大连续训练步数。

输入:

- **loss_scale** (Tensor) - 训练期间的损失缩放系数, 是一个标量, shape 为 ()。
- **overflow** (bool) - 是否发生溢出。

输出:

Bool, 即输入 *overflow*。

支持平台:

Ascend GPU

样例:

```

>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, Parameter, nn, ops
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):

```

(下页继续)

(续上页)

```

...     super(Net, self).__init__()
...     self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
↪float32)),
...                               name='weight')
...     self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output
...
>>> in_features, out_features = 16, 10
>>> net = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = nn.WithLossCell(net, loss)
>>> manager = nn.DynamicLossScaleUpdateCell(loss_scale_value=2**12, scale_factor=2, scale_
↪window=1000)
>>> train_network = nn.TrainOneStepWithLossScaleCell(net_with_loss, optimizer, scale_
↪sense=manager)
>>> input = Tensor(np.ones([out_features, in_features]), mindspore.float32)
>>> labels = Tensor(np.ones([out_features,]), mindspore.float32)
>>> output = train_network(input, labels)

```

get_loss_scale()

获取当前损失缩放系数。

返回:

float, 损失缩放系数。

样例:

```

>>> from mindspore import nn
>>> manager = nn.DynamicLossScaleUpdateCell(loss_scale_value=212, scale_factor=2, scale_
↪window=1000)
>>> output = manager.get_loss_scale()
>>> print(output)
212

```

3.3.3 mindspore.nn.FixedLossScaleUpdateCell

class mindspore.nn.FixedLossScaleUpdateCell (*loss_scale_value*)

固定损失缩放系数的神经元。

该类是 *mindspore.amp.FixedLossScaleManager* 的 *get_update_cell* 方法的返回值。训练过程中, 类 *mindspore.nn.TrainOneStepWithLossScaleCell* 会调用该 Cell。

参数:

- **loss_scale_value** (float) - 初始损失缩放系数。

输入:

- **loss_scale** (Tensor) - 训练期间的损失缩放系数, 是一个标量, shape 为 ()。在当前类中, 该值被忽略。
- **overflow** (bool) - 是否发生溢出。

输出:

Bool, 即输入 *overflow*。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, Parameter, nn, ops
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
↪float32)),
...                                 name='weight')
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output
...
>>> in_features, out_features = 16, 10
>>> net = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = nn.WithLossCell(net, loss)
>>> manager = nn.FixedLossScaleUpdateCell(loss_scale_value=2**12)
>>> train_network = nn.TrainOneStepWithLossScaleCell(net_with_loss, optimizer, scale_
↪sense=manager)
>>> input = Tensor(np.ones([out_features, in_features]), mindspore.float32)
>>> labels = Tensor(np.ones([out_features,]), mindspore.float32)
>>> output = train_network(input, labels)
```

get_loss_scale()

获取当前损失缩放系数。

返回:

float, 损失缩放系数。

样例:

```
>>> from mindspore import nn
>>> manager = nn.FixedLossScaleUpdateCell(loss_scale_value=212)
>>> output = manager.get_loss_scale()
>>> print(output)
212
```

3.3.4 mindspore.nn.ForwardValueAndGrad

class mindspore.nn.**ForwardValueAndGrad** (*network, weights=None, get_all=False, get_by_list=False, sens_param=False*)

训练网络的封装。

包括正向网络和梯度函数。该类生成的 Cell 使用 '*inputs' 输入来训练。通过梯度函数来创建反向图，用于计算梯度。

参数：

- **network** (Union[Cell, Function, MethodType]) - 训练网络。
- **weights** (ParameterTuple) - 训练网络中需要计算梯度的参数。默认值：None。
- **get_all** (bool) - 如果为 True，则计算网络输入对应的梯度。默认值：False。
- **get_by_list** (bool) - 如果为 True，则计算参数变量对应的梯度。如果 **get_all** 和 **get_by_list** 都为 False，则计算第一个输入对应的梯度。如果 **get_all** 和 **get_by_list** 都为 True，则以 ((输入的梯度), (参数的梯度)) 的形式同时获取输入和参数变量的梯度。默认值：False。
- **sens_param** (bool) - 是否将 sens 作为输入。如果 **sens_param** 为 False，则 sens 默认为 'ones_like(outputs)'。默认值：False。如果 **sens_param** 为 True，则需要指定 sens 的值。

输入：

- ***inputs** (Tuple(Tensor...)) - shape 为 (N,...) 的输入 tuple。
- **sens** - 反向传播梯度的缩放值。如果网络有单个输出，则 sens 是 tensor。如果网络有多个输出，则 sens 是 tuple(tensor)。

输出：

- **forward value** - 网络运行的正向结果。
- **gradients** (tuple(tensor)) - 网络反向传播的梯度。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, nn, ops, ParameterTuple, Parameter
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="weight")
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         out = self.matmul(x, self.weight)
...         return out
...
>>> net = Net()
>>> criterion = nn.SoftmaxCrossEntropyWithLogits()
>>> net_with_criterion = nn.WithLossCell(net, criterion)
>>> weight = ParameterTuple(net.trainable_params())
>>> train_network = nn.ForwardValueAndGrad(net_with_criterion, weights=weight, get_all=True,
->get_by_list=True)
>>> inputs = Tensor(np.ones([1, 2]).astype(np.float32))
```

(下页继续)

(续上页)

```
>>> labels = Tensor(np.ones([1, 2]).astype(np.float32))
>>> result = train_network(inputs, labels)
>>> print(result)
(Tensor(shape=[1], dtype=Float32, value= [ 1.38629436e+00]), ((Tensor(shape=[1, 2],
dtype=Float32, value=
[[ -1.00000000e+00,  -1.00000000e+00]]), Tensor(shape=[1, 2], dtype=Float32, value=
[[ 0.00000000e+00,  0.00000000e+00]])), (Tensor(shape=[2, 2], dtype=Float32, value=
[[ -5.00000000e-01,  -5.00000000e-01],
[ -5.00000000e-01,  -5.00000000e-01]])),))
```

3.3.5 mindspore.nn.GetNextSingleOp

class mindspore.nn.GetNextSingleOp (dataset_types, dataset_shapes, queue_name)

用于获取下一条数据的 Cell。更详细的信息请参考[mindspore.ops.GetNext](#)。

参数：

- **dataset_types** (list[mindspore.dtype]) - 数据集类型。
- **dataset_shapes** (list[tuple[int]]) - 数据集的 shape。
- **queue_name** (str) - 待获取数据的队列名称。

输出：

tuple[Tensor]，从数据集中获取的数据。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> from mindspore import ops, nn
>>> from mindspore import dataset as ds
>>> from mindspore import dtype as mstype
>>>
>>> data_path = "/path/to/MNIST_Data/train/"
>>> train_dataset = ds.MnistDataset(data_path, num_samples=10)
>>> dataset_helper = mindspore.DatasetHelper(train_dataset, dataset_sink_mode=True)
>>> dataset = dataset_helper.iter_dataset
>>> dataset_types, dataset_shapes = dataset_helper.types_shapes()
>>> queue_name = dataset.__transfer_dataset__.queue_name
>>> get_next_single_op_net = nn.GetNextSingleOp(dataset_types, dataset_shapes, queue_name)
>>> data, label = get_next_single_op_net()
>>> relu = ops.ReLU()
>>> result = relu(data.astype(mstype.float32))
>>> print(result.shape)
(28, 28, 1)
```

3.3.6 mindspore.nn.GradAccumulationCell

class mindspore.nn.GradAccumulationCell (*network, micro_size*)

将 MiniBatch 切分成更细粒度的 MicroBatch，用于半自动/全自动并行模式下的梯度累加训练中。

说明：接口即将废弃，请使用接口 `mindspore.parallel.nn.GradAccumulation` 代替。

参数：

- **network** (Cell) - 要修饰的目标网络。
- **micro_size** (int) - MicroBatch 大小。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore.nn as nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> net = nn.GradAccumulationCell(net, 4)
```

3.3.7 mindspore.nn.ParameterUpdate

class mindspore.nn.ParameterUpdate (*param*)

更新参数的 Cell。

使用输入的 *Tensor* 值更新 *param* 的值。

参数：

- **param** (Parameter) - 输入的参数。

输入：

- **x** (Tensor) - shape 和 type 与 *param* 相同的 Tensor。

输出：

Tensor，更新后的值。

异常：

- **KeyError** - 指定名称的参数不存在。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, Tensor
>>> network = nn.Dense(3, 4)
>>> param = network.parameters_dict()['weight']
```

(下页继续)

(续上页)

```
>>> update = nn.ParameterUpdate(param)
>>> update.phase = "update_param"
>>> weight = Tensor(np.arange(12).reshape((4, 3)), mindspore.float32)
>>> output = update(weight)
>>> print(output)
[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]
 [ 9. 10. 11.]]
```

3.3.8 mindspore.nn.PipelineCell

class mindspore.nn.PipelineCell (network, micro_size, stage_config=None)

将 MiniBatch 切分成更细粒度的 MicroBatch，用于流水线并行的训练中。

说明：

- **micro_size** 必须大于或等于流水线 stage 的个数。
- 接口即将废弃，请使用接口 `mindspore.parallel.nn.Pipeline` 代替。

参数：

- **network** (Cell) - 要修饰的目标网络。
- **micro_size** (int) - MicroBatch 大小。
- **stage_config** (dict, 可选) - 流水线并行对于每个 cell 的 stage 配置。默认值：None。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore.nn as nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> net = nn.PipelineCell(net, 4)
```

3.3.9 mindspore.nn.TimeDistributed

class mindspore.nn.TimeDistributed (layer, time_axis, reshape_with_axis=None)

时间序列封装层。

TimeDistributed 是一个封装层，它允许将一个网络层应用到输入的每个时间切片。 x 至少是三维。执行中有两种情况：当提供 `reshape_with_axis` 时，选择 `reshape` 方法会更高效；否则，将使用沿 `time_axis` 划分输入的方法，这种方法更通用。比如，在处理 BN 时无法提供 `reshape_with_axis`。

参数：

- **layer** (Union[Cell, Primitive]) - 需被封装的 Cell 或 Primitive。
- **time_axis** (int) - 指定各个时间切片上的轴。

- **reshape_with_axis** (int) - 将使用 *time_axis* 调整该轴。默认值: None。

输入:

- **x** (Tensor) - shape 为 $(N, T, *)$ 的 Tensor。其中 $*$ 表示任意数量的附加维度。

输出:

shape 为 $(N, T, *)$ 的 Tensor。

异常:

- **TypeError** - layer 不是 Cell 或 Primitive 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> x = ms.Tensor(np.random.random([32, 10, 3]), ms.float32)
>>> dense = ms.nn.Dense(3, 6)
>>> net = ms.nn.TimeDistributed(dense, time_axis=1, reshape_with_axis=0)
>>> output = net(x)
>>> print(output.shape)
(32, 10, 6)
```

3.3.10 mindspore.nn.TrainOneStepCell

class mindspore.nn.TrainOneStepCell (network, optimizer, sens=None, return_grad=False)

训练网络封装类。

封装 *network* 和 *optimizer*。构建一个输入 **inputs* 的用于训练的 Cell。执行函数 *construct* 中会构建反向图以更新网络参数。支持不同的并行训练模式。

参数:

- **network** (Cell) - 训练网络。只支持单输出网络。
- **optimizer** (Union[Cell]) - 用于更新网络参数的优化器。
- **sens** (numbers.Number, 可选) - 反向传播的输入, 缩放系数。默认值: None, 即取系数为 1.0。
- **return_grad** (bool, 可选) - 是否返回梯度。若为 True, 则会在返回损失函数值 (loss) 的同时以字典的形式返回梯度, 字典的 key 为梯度对应的参数名, value 为梯度值。默认值: False。

输入:

- ***inputs** (Tuple(Tensor)) - shape 为 $(N, \dots)$ 的 Tensor 组成的 tuple。

输出:

Tensor, 损失函数值 (loss), 其 shape 通常为 ()。

异常:

- **TypeError** - *sens* 不是 numbers.Number。

支持平台:

Ascend GPU CPU

样例:

```

>>> import mindspore.nn as nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits()
>>> optim = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> #1) Using the WithLossCell provided by MindSpore
>>> loss_net = nn.WithLossCell(net, loss_fn)
>>> train_net = nn.TrainOneStepCell(loss_net, optim)
>>>
>>> #2) Using user-defined WithLossCell
>>> class MyWithLossCell(nn.Cell):
...     def __init__(self, backbone, loss_fn):
...         super(MyWithLossCell, self).__init__(auto_prefix=False)
...         self._backbone = backbone
...         self._loss_fn = loss_fn
...
...     def construct(self, x, y, label):
...         out = self._backbone(x, y)
...         return self._loss_fn(out, label)
...
...     @property
...     def backbone_network(self):
...         return self._backbone
...
>>> loss_net = MyWithLossCell(net, loss_fn)
>>> train_net = nn.TrainOneStepCell(loss_net, optim)

```

3.3.11 mindspore.nn.TrainOneStepWithLossScaleCell

class mindspore.nn.TrainOneStepWithLossScaleCell (network, optimizer, scale_sense)

使用混合精度功能的训练网络。

实现了包含损失缩放 (loss scale) 的单次训练。它使用网络、优化器和用于更新损失缩放系数 (loss scale) 的 Cell (或一个 Tensor) 作为参数。可在 host 侧或 device 侧更新损失缩放系数。如果需要在 host 侧更新, 使用 Tensor 作为 scale_sense, 否则, 使用可更新损失缩放系数的 Cell 实例作为 scale_sense。

参数:

- **network** (Cell) - 训练网络。仅支持单输出网络。
- **optimizer** (Cell) - 用于更新网络参数的优化器。
- **scale_sense** (Union[Tensor, Cell]) - 如果此值为 Cell 类型, TrainOneStepWithLossScaleCell 会调用它来更新损失缩放系数。如果此值为 Tensor 类型, 可调用 set_sense_scale 来更新损失缩放系数, shape 为 () 或 (1,)。

输入:

- ***inputs** (Tuple(Tensor)) - shape 为 (N, ...) 的 Tensor 组成的元组。

输出:

Tuple, 包含三个 Tensor, 分别为损失函数值、溢出状态和当前损失缩放系数。

- **loss** (Tensor) - 标量, 表示损失函数值。
- **overflow** (Tensor) - 类型为 bool 的标量, 表示是否发生溢出。
- **loss scale** (Tensor) - 表示损失放大系数, shape 为 () 或 (1,)。

异常:

- **TypeError** - `scale_sense` 既不是 `Cell`, 也不是 `Tensor`。
- **ValueError** - `scale_sense` 的 `shape` 既不是 `(1,)` 也不是 `()`。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, Parameter, nn, ops
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
↪float32))),
...                                     name='weight')
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output
...
>>> size, in_features, out_features = 16, 16, 10
>>> #1) when the type of scale_sense is Cell:
>>> net = Net(in_features, out_features)
>>> loss_fn = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = nn.WithLossCell(net, loss_fn)
>>> input = Tensor(np.ones([out_features, in_features]), mindspore.float32)
>>> labels = Tensor(np.ones([out_features,]), mindspore.float32)
>>> loss = net_with_loss(input, labels)
>>> manager = nn.DynamicLossScaleUpdateCell(loss_scale_value=2**12, scale_factor=2, scale_
↪window=1000)
>>> train_network = nn.TrainOneStepWithLossScaleCell(net_with_loss, optimizer, scale_
↪sense=manager)
>>> status = Tensor([0] * 8, mindspore.int32)
>>> scaling_sens = train_network.scale_sense
>>> scaling_sens_filled = ops.ones_like(loss) * ops.cast(scaling_sens, ops.dtype(loss))
>>> grads = train_network.grad(train_network.network, train_network.weights)(input, labels,
↪scaling_sens_filled)
>>> grads = train_network.grad_reducer(grads)
>>> cond = train_network.get_overflow_status(status, grads)
>>> overflow = train_network.process_loss_scale(cond)
>>>
>>> #2) when the type of scale_sense is Tensor:
>>> net = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = nn.WithLossCell(net, loss)
>>> inputs = Tensor(np.ones([size, in_features]).astype(np.float32))
>>> label = Tensor(np.zeros([size, out_features]).astype(np.float32))
>>> scaling_sens = Tensor([1024], dtype=mindspore.float32)
>>> train_network = nn.TrainOneStepWithLossScaleCell(net_with_loss, optimizer, scale_
↪sense=scaling_sens)
```

(下页继续)

(续上页)

```

>>> scaling_sens = Tensor([1], dtype=mstype.float32)
>>> train_network.set_sense_scale(scaling_sens)
>>> output = train_network(inputs, label)
>>>
>>> # update scaling sens and train the network
>>> scaling_sens = Tensor([1], dtype=mindspore.float32)
>>> train_network.set_sense_scale(scaling_sens)
>>> output = train_network(inputs, label)

```

get_overflow_status (*status, compute_output*)

获取浮点溢出状态。

溢出检测的目标过程执行完成后，获取溢出结果。继承该类自定义训练网络时，可复用该接口。

参数：

- **status** (object) - 用于控制与 *start_overflow_check* 的执行序，应设置为 *start_overflow_check* 的第一输出。
- **compute_output** - 对特定计算过程进行溢出检测时，将 *compute_output* 设置为该计算过程的输出。

返回：

bool，是否发生溢出。

process_loss_scale (*overflow*)

根据溢出状态计算损失缩放系数。

继承该类自定义训练网络时，可复用该接口。

参数：

- **overflow** (bool) - 是否发生溢出。

返回：

bool，溢出状态，即输入。

set_sense_scale (*sens*)

如果使用了 Tensor 类型的 *scale_sense*，可调用此函数修改它的值。

参数：

- **sens** (Tensor) - 新的损失缩放系数，其 *shape* 和类型需要与原始 *scale_sense* 相同。

start_overflow_check (*pre_cond, compute_input*)

启动浮点溢出检测。创建并清除溢出检测状态。

指定参数 *pre_cond* 和 *compute_input*，以确保在正确的时间清除溢出状态。以当前接口为例，我们需要在损失函数计算后进行清除状态，在梯度计算过程中检测溢出。在这种情况下，*pre_cond* 应为损失函数的输出，而 *compute_input* 应为梯度计算函数的输入。继承该类自定义训练网络时，可复用该接口。

参数：

- **pre_cond** (Tensor) - 启动溢出检测的先决条件。它决定溢出状态清除和先前处理的执行顺序。它确保函数 *start_overflow* 在执行完先决条件后清除状态。
- **compute_input** (object) - 后续运算的输入。需要对特定的计算过程进行溢出检测。将 *compute_input* 设置这一计算过程的输入，以确保在执行该计算之前清除了溢出状态。

返回：

Tuple[object, object]，第一输出用于控制执行序，为保证编译优化后 *start_overflow_check* 在 *get_overflow_status* 前执行，该值应作为 *get_overflow_status* 的第一个输入。第二输出与 *compute_input* 的输入相同，用于控制执行序，保证在函数返回时完成对溢出标志的清理。

3.3.12 mindspore.nn.WithEvalCell

class mindspore.nn.**WithEvalCell** (*network, loss_fn, add_cast_fp32=False*)

封装前向网络和损失函数。返回用于计算评估指标的损失函数值、前向输出和标签。

参数：

- **network** (Cell) - 前向网络。
- **loss_fn** (Cell) - 损失函数。
- **add_cast_fp32** (bool) - 是否将数据类型调整为 float32。默认值：False。

输入：

- **data** (Tensor) - shape 为 $(N, \dots)$ 的 Tensor。
- **label** (Tensor) - shape 为 $(N, \dots)$ 的 Tensor。

输出：

Tuple(Tensor)，包括标量损失函数、shape 为 $(N, \dots)$ 的网络输出和 shape 为 $(N, \dots)$ 的标签。

异常：

- **TypeError** - *add_cast_fp32* 不是 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.nn as nn
>>> # Define a forward network without loss function, taking LeNet5 as an example.
>>> # Refer to https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits()
>>> eval_net = nn.WithEvalCell(net, loss_fn)
```

3.3.13 mindspore.nn.WithLossCell

class mindspore.nn.**WithLossCell** (*backbone, loss_fn*)

包含损失函数的 Cell。

封装 *backbone* 和 *loss_fn*。此 Cell 接受数据和标签作为输入，并将返回损失函数作为计算结果。

参数：

- **backbone** (Cell) - 要封装的骨干网络。
- **loss_fn** (Cell) - 用于计算损失函数。

输入：

- **data** (Tensor) - shape 为 $(N, \dots)$ 的 Tensor。*data* 的数据类型必须是 float16 或 float32。
- **label** (Tensor) - shape 为 $(N, \dots)$ 的 Tensor。*label* 的数据类型必须是 float16 或 float32。

输出：

Tensor, loss 值，其 shape 通常为 ()。

异常：

- **TypeError** - *data* 或 *label* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> net_with_criterion = nn.WithLossCell(net, loss_fn)
>>>
>>> batch_size = 2
>>> data = Tensor(np.ones([batch_size, 1, 32, 32]).astype(np.float32) * 0.01)
>>> label = Tensor(np.ones([batch_size, 10]).astype(np.float32))
>>>
>>> output_data = net_with_criterion(data, label)
```

property backbone_network

获取骨干网络。

返回：

Cell，骨干网络。

样例：

```
>>> from mindspore import nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> net_with_criterion = nn.WithLossCell(net, loss_fn)
>>> backbone = net_with_criterion.backbone_network
```

3.4 卷积神经网络层

接口名	概述	支持平台
<code>mindspore.nn.Conv1d</code>	一维卷积层。	Ascend GPU CPU
<code>mindspore.nn.Conv1dTranspose</code>	计算一维转置卷积，可以视为 <code>Conv1d</code> 对输入求梯度，也称为反卷积（实际不是真正的反卷积）。	Ascend GPU CPU
<code>mindspore.nn.Conv2d</code>	二维卷积层。	Ascend GPU CPU
<code>mindspore.nn.Conv2dTranspose</code>	计算二维转置卷积，可以视为 <code>Conv2d</code> 对输入求梯度，也称为反卷积（实际不是真正的反卷积）。	Ascend GPU CPU
<code>mindspore.nn.Conv3d</code>	三维卷积层。	Ascend GPU CPU
<code>mindspore.nn.Conv3dTranspose</code>	计算三维转置卷积，可以视为 <code>Conv3d</code> 对输入求梯度，也称为反卷积（实际不是真正的反卷积）。	Ascend GPU CPU

下页继续

表 4 - 续上页

<code>mindspore.nn.Unfold</code>	从图像中提取滑窗的区域块。	Ascend GPU

3.4.1 mindspore.nn.Conv1d

class mindspore.nn.Conv1d (*in_channels, out_channels, kernel_size, stride=1, pad_mode='same', padding=0, dilation=1, group=1, has_bias=False, weight_init=None, bias_init=None, dtype=mstype.float32*)

一维卷积层。

对输入 Tensor 计算一维卷积。通常输入的 shape 为 (N, C_{in}, L_{in}) ，其中 N 为 batch size， C 为通道数， L 为输入序列的长度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中， bias 为输出偏置， ccor 为 cross-correlation 操作， weight 为卷积核的值， X 为输入的特征图。

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中， $\text{bias}(C_{out_j})$ 为第 j 个输出通道的偏置， $\text{weight}(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。卷积核 shape 为 (kernel_size) ，其中 kernel_size 是卷积核的宽度。若考虑到输入输出通道以及 group ，则完整卷积核的 shape 为 $(C_{out}, C_{in}/\text{group}, \text{kernel\_size})$ ，其中 group 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#) 和 [ConvNets](#)。

说明：在 Ascend 平台上，目前只支持深度卷积场景下的分组卷积运算。也就是说，当 $\text{group} > 1$ 的场景下，必须要满足 $\text{in\_channels} = \text{out\_channels} = \text{group}$ 的约束条件。

参数：

- **in_channels** (int) - Conv1d 层输入 Tensor 的空间维度。
- **out_channels** (int) - Conv1d 层输出 Tensor 的空间维度。
- **kernel_size** (int) - 指定一维卷积核的宽度。
- **stride** (int, 可选) - 一维卷积核的移动步长，默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："same"。
 - "same"：在输入的两端填充，使得当 stride 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在右端。如果设置了此模式， padding 必须为 0。
 - "valid"：不对输入进行填充，返回输出可能的最大长度，如果不能构成一个完整 stride ，那么额外的像素将被丢弃。如果设置了此模式， padding 必须为 0。
 - "pad"：对输入填充指定的量。在这种模式下，填充的量由 padding 参数指定。如果设置此模式， padding 必须大于或等于 0。

- **padding** (Union(int, tuple[int], list[int]), 可选) - 当 *pad_mode* 为 "pad" 时, 指定在输入 *input* 的宽度方向上填充的数量。数据类型为 int 或包含 1 个 int 组成的 tuple。表示宽度方向的 *padding* 数量 (左右两边均为该值)。值必须大于等于 0, 默认值: 0。
- **dilation** (Union(int, tuple[int]), 可选) - 卷积核膨胀尺寸。可以为单个 int, 或者由一个 int 组成的 tuple。假设 *dilation* = (*d0*,), 则卷积核在宽度方向间隔 *d0* - 1 个元素进行采样。取值范围为 [1, L]。默认值: 1。
- **group** (int, 可选) - 将过滤器拆分为组, *in_channels* 和 *out_channels* 必须可被 *group* 整除。默认值: 1。
- **has_bias** (bool, 可选) - Conv1d 层是否添加偏置参数。默认值: False。
- **weight_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 权重参数的初始化方法。它可以是 Tensor, str, Initializer 或 numbers.Number。当使用 str 时, 可选 "TruncatedNormal", "Normal", "Uniform", "HeUniform" 和 "XavierUniform" 分布以及常量 "One" 和 "Zero" 分布的值, 可接受别名 "xavier_uniform", "he_uniform", "ones" 和 "zeros"。上述字符串大小写均可。更多细节请参考 [Initializer](#), 的值。默认值: None, 权重使用 "HeUniform" 初始化。
- **bias_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 偏置参数的初始化方法。可以使用的初始化方法与 *weight_init* 相同。更多细节请参考 [Initializer](#), 的值。默认值: None, 偏差使用 "Uniform" 初始化。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: mstype.float32。

输入:

- **x** (Tensor) - shape 为 (*N*, *C_{in}*, *L_{in}*) 的 Tensor。

输出:

Tensor, shape 为 (*N*, *C_{out}*, *L_{out}*)。

pad_mode 为 "same" 时:

$$L_{out} = \left\lceil \frac{L_{in}}{stride} \right\rceil$$

pad_mode 为 "valid" 时:

$$L_{out} = \left\lceil \frac{L_{in} - dilation \times (kernel\_size - 1)}{stride} \right\rceil$$

pad_mode 为 "pad" 时:

$$L_{out} = \left\lceil \frac{L_{in} + 2 \times padding - (kernel\_size - 1) \times dilation - 1}{stride} + 1 \right\rceil$$

异常:

- **TypeError** - *in_channels*、*out_channels*、*kernel_size*、*stride*、*padding* 或 *dilation* 不是 int。
- **ValueError** - *in_channels*、*out_channels*、*kernel_size*、*stride* 或 *dilation* 小于 1。
- **ValueError** - *padding* 小于 0。
- **ValueError** - *pad_mode* 不是 "same", "valid" 或 "pad"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> net = nn.Conv1d(120, 240, 4, has_bias=False, weight_init='normal')
>>> x = Tensor(np.ones([1, 120, 640]), mindspore.float32)
>>> output = net(x).shape
>>> print(output)
(1, 240, 640)
```

3.4.2 mindspore.nn.Conv1dTranspose

```
class mindspore.nn.Conv1dTranspose(in_channels, out_channels, kernel_size, stride=1, pad_mode='same', padding=0,
                                   dilation=1, group=1, has_bias=False, weight_init=None, bias_init=None,
                                   dtype=mstype.float32)
```

计算一维转置卷积，可以视为 Conv1d 对输入求梯度，也称为反卷积（实际不是真正的反卷积）。

输入的 shape 通常是 (N, C_{in}, L_{in}) ，其中 N 是 batch size， C_{in} 是通道数， L_{in} 是序列的长度。当 Conv1d 和 ConvTranspose1d 使用相同的参数初始化时，且 `pad_mode` 设置为“pad”，它们会在输入的两端填充 $dilation * (kernel\_size - 1) - padding$ 个零，这种情况下它们的输入和输出 shape 是互逆的。然而，当 `stride` 大于 1 时，Conv1d 会将多个输入的 shape 映射到同一个输出 shape。反卷积网络可以参考 [Deconvolutional Networks](#)。

参数：

- **in_channels** (int) - Conv1dTranspose 层输入 Tensor 的空间维度。
- **out_channels** (int) - Conv1dTranspose 层输出 Tensor 的空间维度。
- **kernel_size** (int) - 指定一维卷积核的宽度。
- **stride** (int) - 一维卷积核的移动步长，默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："same"。
 - "same"：在输入的两端填充，使得当 `stride` 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在右端。如果设置了此模式，`padding` 必须为 0。
 - "valid"：不对输入进行填充，返回输出可能的最大长度，如果不能构成一个完整 `stride`，那么额外的像素将被丢弃。如果设置了此模式，`padding` 必须为 0。
 - "pad"：对输入填充指定的量。在这种模式下，填充的量由 `padding` 参数指定。如果设置此模式，`padding` 必须大于或等于 0。
- **padding** (int) - 输入两侧填充的数量。默认值：0。
- **dilation** (int) - 一维卷积核膨胀尺寸。若 $k > 1$ ，则 kernel 间隔 k 个元素进行采样。 k 取值范围为 $[1, L]$ 。默认值：1。
- **group** (int) - 将过滤器拆分为组，`in_channels` 和 `out_channels` 必须可被 `group` 整除。当 `group` 大于 1 时，暂不支持 Ascend 平台。默认值：1。
- **has_bias** (bool) - Conv1dTranspose 层是否添加偏置参数。默认值：False。
- **weight_init** (Union[Tensor, str, Initializer, numbers.Number]) - 权重参数的初始化方法。它可以是 Tensor, str, Initializer 或 numbers.Number。当使用 str 时，可选 "TruncatedNormal"，"Normal"，"Uniform"，"HeUniform" 和 "XavierUniform" 分布以及常量 "One" 和 "Zero" 分布的值，可接受别名 "xavier_uniform"，"he_uniform"，"ones" 和 "zeros"。上述字符串大小写均可。更多细节请参考 Initializer 的值。默认值：None，权重使用 HeUniform 初始化。

- **bias_init** (Union[`Tensor`, `str`, `Initializer`, `numbers.Number`]) - 偏置参数的初始化方法。可以使用的初始化方法与“weight_init”相同。更多细节请参考 `Initializer` 的值。默认值: `None` , 偏差使用 `Uniform` 初始化。
- **dtype** (`mindspore.dtype`) - `Parameters` 的 `dtype`。默认值: `mstype.float32`。

输入:

- **x** (`Tensor`) - shape 为 (N, C_{in}, L_{in}) 的 `Tensor`。

输出:

`Tensor`, shape 为 (N, C_{out}, L_{out}) 。

- 当 `pad_mode` 设置为 "same" 时: $L_{out} = \frac{L_{in} + \text{stride} - 1}{\text{stride}}$
- 当 `pad_mode` 设置为 "valid" 时: $L_{out} = (L_{in} - 1) \times \text{stride} + \text{dilation} \times (\text{kernel\_size} - 1) + 1$
- 当 `pad_mode` 设置为 "pad" 时: $L_{out} = (L_{in} - 1) \times \text{stride} - 2 \times \text{padding} + \text{dilation} \times (\text{kernel\_size} - 1) + 1$

异常:

- **TypeError** - `in_channels`、`out_channels`、`kernel_size`、`stride`、`padding` 或 `dilation` 不是 `int`。
- **ValueError** - `in_channels`、`out_channels`、`kernel_size`、`stride` 或 `dilation` 小于 1。
- **ValueError** - `padding` 小于 0。
- **ValueError** - `pad_mode` 不是 "same", "valid" 或 "pad"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> net = nn.Conv1dTranspose(3, 64, 4, has_bias=False, weight_init='normal', pad_mode='pad')
>>> x = Tensor(np.ones([1, 3, 50]), mindspore.float32)
>>> output = net(x).shape
>>> print(output)
(1, 64, 53)
```

3.4.3 mindspore.nn.Conv2d

class `mindspore.nn.Conv2d` (`in_channels`, `out_channels`, `kernel_size`, `stride=1`, `pad_mode='same'`, `padding=0`, `dilation=1`, `group=1`, `has_bias=False`, `weight_init=None`, `bias_init=None`, `data_format='NCHW'`, `dtype=mstype.float32`)

二维卷积层。

对输入 `Tensor` 计算二维卷积, 通常输入的 shape 为 $(N, C_{in}, H_{in}, W_{in})$, 其中 N 为 batch size, C 为通道数, H 为特征图的高度, W 为特征图的宽度。

根据以下公式计算输出:

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中, `bias` 为输出偏置, `ccor` 为 `cross-correlation` 操作, `weight` 为卷积核的值, X 为输入的特征图。

- i 对应 batch 数, 其范围为 $[0, N - 1]$, 其中 N 为输入 batch。

- j 对应输出通道, 其范围为 $[0, C_{out} - 1]$, 其中 C_{out} 为输出通道数, 该值也等于卷积核的个数。
- k 对应输入通道数, 其范围为 $[0, C_{in} - 1]$, 其中 C_{in} 为输入通道数, 该值也等于卷积核的通道数。

因此, 上面的公式中, $bias(C_{out_j})$ 为第 j 个输出通道的偏置, $weight(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片, $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。卷积核 shape 为 $(kernel\_size[0], kernel\_size[1])$, 其中 $kernel\_size[0]$ 和 $kernel\_size[1]$ 是卷积核的高度和宽度。若考虑到输入输出通道以及 `group`, 则完整卷积核的 shape 为 $(C_{out}, C_{in}/group, kernel\_size[0], kernel\_size[1])$, 其中 `group` 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层, 请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

说明: 在 Ascend 平台上, 目前只支持深度卷积场景下的分组卷积运算。也就是说, 当 $group > 1$ 的场景下, 必须要满足 $in\_channels = out\_channels = group$ 的约束条件。

参数:

- **in_channels** (int) - Conv2d 层输入 Tensor 的空间维度。
- **out_channels** (int) - Conv2d 层输出 Tensor 的空间维度。
- **kernel_size** (Union[int, tuple[int]]) - 指定二维卷积核的高度和宽度。数据类型为整型或两个整型的 tuple。一个整数表示卷积核的高度和宽度均为该值。两个整数的 tuple 分别表示卷积核的高度和宽度。
- **stride** (Union[int, tuple[int]], 可选) - 二维卷积核的移动步长。数据类型为整型或者长度为二或四的整型 tuple。一个整数表示在高度和宽度方向的移动步长均为该值。两个整数的 tuple 分别表示在高度和宽度方向的移动步长。默认值: 1。
- **pad_mode** (str, 可选) - 指定填充模式, 填充值为 0。可选值为 "same", "valid" 或 "pad"。默认值: "same"。
 - "same": 在输入的四周填充, 使得当 `stride` 为 1 时, 输入和输出的 shape 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在底部/右侧。如果设置了此模式, `padding` 必须为 0。
 - "valid": 不对输入进行填充, 返回输出可能的最大高度和宽度, 如果不能构成一个完整 `stride`, 那么额外的像素将被丢弃。如果设置了此模式, `padding` 必须为 0。
 - "pad": 对输入填充指定的量。在这种模式下, 在输入的高度和宽度方向上填充的量由 `padding` 参数指定。如果设置此模式, `padding` 必须大于或等于 0。
- **padding** (Union[int, tuple[int]], 可选) - 输入的高度和宽度方向上填充的数量。数据类型为 int 或包含 4 个整数的 tuple。如果 `padding` 是一个整数, 那么上、下、左、右的填充都等于 `padding`。如果 `padding` 是一个有 4 个整数的 tuple, 那么上、下、左、右的填充分别等于 `padding[0]`、`padding[1]`、`padding[2]` 和 `padding[3]`。值应该要大于等于 0, 默认值: 0。
- **dilation** (Union[int, tuple[int]], 可选) - 卷积核膨胀尺寸。可以为单个 int, 或者由两个/四个 int 组成的 tuple。单个 int 表示在高度和宽度方向的膨胀尺寸均为该值。两个 int 组成的 tuple 分别表示在高度和宽度方向的膨胀尺寸。若为四个 int, N、C 两维度 int 默认为 1, H、W 两维度分别对应高度和宽度上的膨胀尺寸。假设 $dilation = (d0, d1)$, 则卷积核在高度方向间隔 $d0 - 1$ 个元素进行采样, 在宽度方向间隔 $d1 - 1$ 个元素进行采样。高度和宽度上取值范围分别为 $[1, H]$ 和 $[1, W]$ 。默认值: 1。
- **group** (int, 可选) - 将过滤器拆分为组, `in_channels` 和 `out_channels` 必须可被 `group` 整除。如果组数等于 `in_channels` 和 `out_channels`, 这个二维卷积层也被称为二维深度卷积层。默认值: 1。
- **has_bias** (bool, 可选) - Conv2d 层是否添加偏置参数。默认值: False。
- **weight_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 权重参数的初始化方法。它可以是 Tensor, str, Initializer 或 numbers.Number。当使用 str 时, 可选 "TruncatedNormal", "Normal", "Uniform", "HeUniform" 和 "XavierUniform" 分布以及常量 "One" 和 "Zero" 分布的值, 可接受别名

"xavier_uniform", "he_uniform", "ones" 和 "zeros"。上述字符串大小写均可。更多细节请参考 [Initializer](#), 的值。默认值: None, 权重使用 "HeUniform" 初始化。

- **bias_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 偏置参数的初始化方法。可以使用的初始化方法与 *weight_init* 相同。更多细节请参考 [Initializer](#) 的值。默认值: None, 偏差使用 "Uniform" 初始化。
- **data_format** (str, 可选) - 数据格式的可选值有 "NHWC", "NCHW"。默认值: "NCHW"。(目前仅 GPU 支持 NHWC。)
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: *mstype.float32*。

输入:

- **x** (Tensor) - Shape 为 $(N, C_{in}, H_{in}, W_{in})$ 或者 $(N, H_{in}, W_{in}, C_{in})$ 的 Tensor。

输出:

Tensor, shape 为 $(N, C_{out}, H_{out}, W_{out})$ 或者 $(N, H_{out}, W_{out}, C_{out})$ 。

pad_mode 为 "same" 时:

$$\begin{aligned} H_{out} &= \left\lceil \frac{H_{in}}{\text{stride}[0]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in}}{\text{stride}[1]} \right\rceil \end{aligned}$$

pad_mode 为 "valid" 时:

$$\begin{aligned} H_{out} &= \left\lceil \frac{H_{in} - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1)}{\text{stride}[0]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1)}{\text{stride}[1]} \right\rceil \end{aligned}$$

pad_mode 为 "pad" 时:

$$\begin{aligned} H_{out} &= \left\lceil \frac{H_{in} + \text{padding}[0] + \text{padding}[1] - (\text{kernel\_size}[0] - 1) \times \text{dilation}[0] - 1}{\text{stride}[0]} + 1 \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} + \text{padding}[2] + \text{padding}[3] - (\text{kernel\_size}[1] - 1) \times \text{dilation}[1] - 1}{\text{stride}[1]} + 1 \right\rceil \end{aligned}$$

异常:

- **TypeError** - 如果 *in_channels*, *out_channels* 或者 *group* 不是整数。
- **TypeError** - 如果 *kernel_size*, *stride*, *padding* 或者 *dilation* 既不是整数也不是 tuple。
- **ValueError** - 如果 *in_channels*, *out_channels*, *kernel_size*, *stride* 或者 *dilation* 小于 1。
- **ValueError** - 如果 *padding* 小于 0。
- **ValueError** - 如果 *pad_mode* 不是 "same", "valid" 或 "pad"。
- **ValueError** - 如果 *padding* 是一个长度不等于 4 的 tuple。
- **ValueError** - 如果 *pad_mode* 不等于 "pad" 且 *padding* 不等于 (0,0,0,0)。
- **ValueError** - 如果 *data_format* 既不是 "NCHW" 也不是 "NHWC"。

支持平台:

Ascend GPU CPU

样例:


```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> net = nn.Conv2d(120, 240, 4, has_bias=False, weight_init='normal')
>>> x = Tensor(np.ones([1, 120, 1024, 640]), mindspore.float32)
>>> output = net(x).shape
>>> print(output)
(1, 240, 1024, 640)
```

3.4.4 mindspore.nn.Conv2dTranspose

```
class mindspore.nn.Conv2dTranspose(in_channels, out_channels, kernel_size, stride=1, pad_mode='same', padding=0,
                                   output_padding=0, dilation=1, group=1, has_bias=False, weight_init=None,
                                   bias_init=None, dtype=mstype.float32)
```

计算二维转置卷积，可以视为 Conv2d 对输入求梯度，也称为反卷积（实际不是真正的反卷积）。

输入的 shape 通常为 $(N, C_{in}, H_{in}, W_{in})$ ，其中 N 是 batch size， C_{in} 是空间维度， H_{in}, W_{in} 分别为特征层的高度和宽度。当 Conv2d 和 ConvTranspose2d 使用相同的参数初始化时，且 `pad_mode` 设置为“pad”，它们会在输入的高度和宽度方向上填充 $dilation * (kernel\_size - 1) - padding$ 个零，这种情况下它们的输入和输出 shape 是互逆的。然而，当 `stride` 大于 1 时，Conv2d 会将多个输入的 shape 映射到同一个输出 shape。反卷积网络可以参考 [Deconvolutional Networks](#)。

参数：

- **in_channels** (int) - Conv2dTranspose 层输入 Tensor 的空间维度。
- **out_channels** (int) - Conv2dTranspose 层输出 Tensor 的空间维度。
- **kernel_size** (Union[int, tuple[int]]) - 指定二维卷积核的高度和宽度。数据类型为整型或两个整型的 tuple。一个整数表示卷积核的高度和宽度均为该值。两个整数的 tuple 分别表示卷积核的高度和宽度。
- **stride** (Union[int, tuple[int]]) - 二维卷积核的移动步长。数据类型为整型或两个整型的 tuple。一个整数表示在高度和宽度方向的移动步长均为该值。两个整数的 tuple 分别表示在高度和宽度方向的移动步长。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same", "valid" 或 "pad"。默认值："same"。
 - "same": 在输入的四周填充，使得当 `stride` 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在底部/右侧。如果设置了此模式，`padding` 必须为 0。
 - "valid": 不对输入进行填充，返回输出可能的最大高度和宽度，如果不能构成一个完整 `stride`，那么额外的像素将被丢弃。如果设置了此模式，`padding` 必须为 0。
 - "pad": 对输入填充指定的量。在这种模式下，在输入的高度和宽度方向上填充的量由 `padding` 参数指定。如果设置此模式，`padding` 必须大于或等于 0。
- **padding** (Union[int, tuple[int]]) - 输入的高度和宽度方向上填充的数量。数据类型为整型或包含四个整数的 tuple。如果 `padding` 是一个整数，那么上、下、左、右的填充都等于 `padding`。如果 `padding` 是一个有四个整数的 tuple，那么上、下、左、右的填充分别等于 `padding[0]`、`padding[1]`、`padding[2]` 和 `padding[3]`。值应该要大于等于 0，默认值：0。
- **output_padding** (Union[int, tuple[int]]) - 输出的高度和宽度方向上填充的数量。数据类型为整型或包含两个整数的 tuple。如果 `output_padding` 是一个整数，那么下、右的填充都等于 `output_padding`。如果 `output_padding` 是一个有两个整数的 tuple，那么下、右的填充分别等于 `output_padding[0]`、`output_padding[1]`。如果 `output_padding` 不为 0，`pad_mode` 必须为 `pad`。`output_padding` 取值范围为 $[0, \max(stride, dilation))$ ，默认值：0。
- **dilation** (Union[int, tuple[int]]) - 二维卷积核膨胀尺寸。可以为单个 int，或者由两个 int 组成的 tuple。单个 int 表示在高度和宽度方向的膨胀尺寸均为该值。两个 int 组成的 tuple 分别表示在高度和宽度方向的膨胀尺寸。

假设 $dilation = (d0, d1)$, 则卷积核在高度方向间隔 $d0 - 1$ 个元素进行采样, 在宽度方向间隔 $d1 - 1$ 个元素进行采样。高度和宽度上取值范围分别为 $[1, H]$ 和 $[1, W]$ 。默认值: 1。

- **group** (int) - 将过滤器拆分为组, $in\_channels$ 和 $out\_channels$ 必须可被 $group$ 整除。如果组数等于 $in\_channels$ 和 $out\_channels$, 这个二维卷积层也被称为二维深度卷积层。默认值: 1。
- **has_bias** (bool) - Conv2dTranspose 层是否添加偏置参数。默认值: False。
- **weight_init** (Union[Tensor, str, Initializer, numbers.Number]) - 权重参数的初始化方法。它可以是 Tensor, str, Initializer 或 numbers.Number。当使用 str 时, 可选 "TruncatedNormal", "Normal", "Uniform", "HeUniform" 和 "XavierUniform" 分布以及常量 "One" 和 "Zero" 分布的值, 可接受别名 "xavier_uniform", "he_uniform", "ones" 和 "zeros"。上述字符串大小写均可。更多细节请参考 Initializer 的值。默认值: None, 权重使用 HeUniform 初始化。
- **bias_init** (Union[Tensor, str, Initializer, numbers.Number]) - 偏置参数的初始化方法。可以使用的初始化方法与 "weight_init" 相同。更多细节请参考 Initializer 的值。默认值: None, 偏差使用 Uniform 初始化。
- **dtype** (mindspore.dtype) - Parameters 的 dtype。默认值: mstype.float32。

输入:

- **x** (Tensor) - Shape 为 $(N, C_{in}, H_{in}, W_{in})$ 的 Tensor。

输出:

Tensor, shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。

pad_mode 为 "same" 时:

$$\begin{aligned} H_{out} &= H_{in} \times \text{stride}[0] \\ W_{out} &= W_{in} \times \text{stride}[1] \end{aligned}$$

pad_mode 为 "valid" 时:

$$\begin{aligned} H_{out} &= H_{in} \times \text{stride}[0] + \max\{(\text{dilation}[0] - 1) \times (\text{kernel\_size}[0] - 1) - \text{stride}[0], 0\} \\ W_{out} &= W_{in} \times \text{stride}[1] + \max\{(\text{dilation}[1] - 1) \times (\text{kernel\_size}[1] - 1) - \text{stride}[1], 0\} \end{aligned}$$

pad_mode 为 "pad" 时:

$$\begin{aligned} H_{out} &= H_{in} \times \text{stride}[0] - (\text{padding}[0] + \text{padding}[1]) + \text{kernel\_size}[0] + (\text{dilation}[0] - 1) \times (\text{kernel\_size}[0] - 1) - \text{stride}[0] + \text{outp} \\ W_{out} &= W_{in} \times \text{stride}[1] - (\text{padding}[2] + \text{padding}[3]) + \text{kernel\_size}[1] + (\text{dilation}[1] - 1) \times (\text{kernel\_size}[1] - 1) - \text{stride}[1] + \text{outp} \end{aligned}$$

异常:

- **TypeError** - 如果 $in\_channels$, $out\_channels$ 或者 $group$ 不是整数。
- **TypeError** - 如果 $kernel\_size$, $stride$, $padding$ 或者 $dilation$ 既不是整数也不是 tuple。
- **ValueError** - 如果 $in\_channels$, $out\_channels$, $kernel\_size$, $stride$ 或者 $dilation$ 小于 1。
- **ValueError** - 如果 $padding$ 小于 0。
- **ValueError** - 如果 pad_mode 不是 "same", "valid" 或 "pad"。
- **ValueError** - 如果 $padding$ 是一个长度不等于 4 的 tuple。
- **ValueError** - 如果 pad_mode 不等于 "pad" 且 $padding$ 不等于 (0,0,0,0)。

支持平台:

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> net = nn.Conv2dTranspose(3, 64, 4, has_bias=False, weight_init='normal', pad_mode='pad')
>>> x = Tensor(np.ones([1, 3, 16, 50]), mindspore.float32)
>>> output = net(x).shape
>>> print(output)
(1, 64, 19, 53)
```

3.4.5 mindspore.nn.Conv3d

class mindspore.nn.Conv3d(*in_channels, out_channels, kernel_size, stride=1, pad_mode='same', padding=0, dilation=1, group=1, has_bias=False, weight_init=None, bias_init=None, data_format='NCDHW', dtype=mstype.float32*)

三维卷积层。

对输入 Tensor 计算三维卷积。通常，输入 Tensor 的 shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C 为通道数， D, H, W 分别为特征图的深度、高度和宽度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中， bias 为输出偏置， ccor 为 [cross-correlation](#) 操作， weight 为卷积核的值， X 为输入的特征图。

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中， $\text{bias}(C_{out_j})$ 为第 j 个输出通道的偏置， $\text{weight}(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。

卷积核 shape 为 $(\text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ ，其中 $\text{kernel\_size}[0]$ 、 $\text{kernel\_size}[1]$ 和 $\text{kernel\_size}[2]$ 分别是卷积核的深度、高度和宽度。若考虑到输入输出通道以及 group ，则完整卷积核的 shape 为 $(C_{out}, C_{in}/\text{group}, \text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ ，其中 group 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

说明：在 Ascend 平台上，目前只支持深度卷积场景下的分组卷积运算。也就是说，当 $\text{group} > 1$ 的场景下，必须要满足 $\text{in\_channels} = \text{out\_channels} = \text{group}$ 的约束条件。

参数：

- **in_channels** (int) - Conv3d 层输入 Tensor 的空间维度。
- **out_channels** (int) - Conv3d 层输出 Tensor 的空间维度。
- **kernel_size** (Union[int, tuple[int]]) - 指定三维卷积核的高度和宽度。可以为单个 int 或一个包含 3 个 int 组成的元组。单个整数表示该值同时适用于内核的深度、高度和宽度。包含 3 个整数的元组表示第一个值用于深度，另两个值用于高度和宽度。

- **stride** (Union[int, tuple[int]], 可选) - 三维卷积核的移动步长。数据类型为整型或三个整型的 tuple。一个整数表示在深度、高度和宽度方向的移动步长均为该值。三个整数的 tuple 分别表示在深度、高度和宽度方向的移动步长。默认值: 1。
- **pad_mode** (str, 可选) - 指定填充模式, 填充值为 0。可选值为 "same", "valid" 或 "pad"。默认值: "same"。
 - "same": 在输入的深度、高度和宽度维度进行填充, 使得当 *stride* 为 1 时, 输入和输出的 shape 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在前方/底部/右侧。如果设置了此模式, *padding* 必须为 0。
 - "valid": 不对输入进行填充, 返回输出可能的最大深度、高度和宽度, 如果不能构成一个完整 stride, 那么额外的像素将被丢弃。如果设置了此模式, *padding* 必须为 0。
 - "pad": 对输入填充指定的量。在这种模式下, 在输入的深度、高度和宽度方向上填充的量由 *padding* 参数指定。如果设置此模式, *padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int]), 可选) - 输入的深度、高度和宽度方向上填充的数量。数据类型为 int 或包含 6 个整数的 tuple。如果 *padding* 是一个整数, 则前部、后部、顶部、底部, 左边和右边的填充都等于 *padding*。如果 *padding* 是 6 个整数的 tuple, 则前部、尾部、顶部、底部、左边和右边的填充分别等于填充 *padding*[0]、*padding*[1]、*padding*[2]、*padding*[3]、*padding*[4] 和 *padding*[5]。值应该要大于等于 0, 默认值: 0。
- **dilation** (Union[int, tuple[int]], 可选) - 卷积核膨胀尺寸。可以为单个 int, 或者由三个 int 组成的 tuple。单个 int 表示在深度、高度和宽度方向的膨胀尺寸均为该值。三个 int 组成的 tuple 分别表示在深度、高度和宽度方向的膨胀尺寸。假设 *dilation* = (*d0*, *d1*, *d2*), 则卷积核在深度方向间隔 *d0* - 1 个元素进行采样, 在高度方向间隔 *d1* - 1 个元素进行采样, 在宽度方向间隔 *d2* - 1 个元素进行采样。深度、高度和宽度上取值范围分别为 [1, D]、[1, H] 和 [1, W]。默认值: 1。
- **group** (int, 可选) - 将过滤器拆分为组, *in_channels* 和 *out_channels* 必须可被 *group* 整除。默认值: 1。
- **has_bias** (bool, 可选) - Conv3d 层是否添加偏置参数。默认值: False。
- **weight_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 权重参数的初始化方法。它可以是 Tensor, str, Initializer 或 numbers.Number。当使用 str 时, 可选 "TruncatedNormal", "Normal", "Uniform", "HeUniform" 和 "XavierUniform" 分布以及常量 "One" 和 "Zero" 分布的值, 可接受别名 "xavier_uniform", "he_uniform", "ones" 和 "zeros"。上述字符串大小写均可。更多细节请参考 [Initializer](#), 的值。默认值: None, 权重使用 "HeUniform" 初始化。
- **bias_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 偏置参数的初始化方法。可以使用的初始化方法与 *weight_init* 相同。更多细节请参考 [Initializer](#), 的值。默认值: None, 偏差使用 "Uniform" 初始化。
- **data_format** (str, 可选) - 数据格式的可选值。目前仅支持 'NCDHW'。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: mstype.float32。

输入:

- **x** (Tensor) - shape 为 (*N*, *C_{in}*, *D_{in}*, *H_{in}*, *W_{in}*) 的 Tensor。目前, CPU 和 GPU 平台上输入数据类型支持 float16 和 float32, Ascend 平台上输入数据类型只支持 float16。

输出:

Tensor, shape 为 (*N*, *C_{out}*, *D_{out}*, *H_{out}*, *W_{out}*)。

pad_mode 为 "same" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in}}{\text{stride}[0]} \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in}}{\text{stride}[1]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in}}{\text{stride}[2]} \right\rceil \end{aligned}$$

`pad_mode` 为 "valid" 时:

$$\begin{aligned} D_{out} &= \left\lfloor \frac{D_{in} - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1)}{\text{stride}[0]} + 1 \right\rfloor \\ H_{out} &= \left\lfloor \frac{H_{in} - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1)}{\text{stride}[1]} + 1 \right\rfloor \\ W_{out} &= \left\lfloor \frac{W_{in} - \text{dilation}[2] \times (\text{kernel\_size}[2] - 1)}{\text{stride}[2]} + 1 \right\rfloor \end{aligned}$$

`pad_mode` 为 "pad" 时:

$$\begin{aligned} D_{out} &= \left\lfloor \frac{D_{in} + \text{padding}[0] + \text{padding}[1] - (\text{dilation}[0] - 1) \times \text{kernel\_size}[0] - 1}{\text{stride}[0]} + 1 \right\rfloor \\ H_{out} &= \left\lfloor \frac{H_{in} + \text{padding}[2] + \text{padding}[3] - (\text{dilation}[1] - 1) \times \text{kernel\_size}[1] - 1}{\text{stride}[1]} + 1 \right\rfloor \\ W_{out} &= \left\lfloor \frac{W_{in} + \text{padding}[4] + \text{padding}[5] - (\text{dilation}[2] - 1) \times \text{kernel\_size}[2] - 1}{\text{stride}[2]} + 1 \right\rfloor \end{aligned}$$

异常:

- **TypeError** - `in_channels`、`out_channels` 或 `group` 不是 `int`。
- **TypeError** - `kernel_size`、`stride`、`padding` 或 `dilation` 既不是 `int` 也不是 `tuple`。
- **ValueError** - `out_channels`、`kernel_size`、`stride` 或 `dilation` 小于 1。
- **ValueError** - `padding` 小于 0。
- **ValueError** - `pad_mode` 不是 "same", "valid" 或 "pad"。
- **ValueError** - `padding` 是长度不等于 6 的 `tuple`。
- **ValueError** - `pad_mode` 不等于 "pad" 且 `padding` 不等于 (0, 0, 0, 0, 0, 0)。
- **ValueError** - `data_format` 不是 "NCDHW"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.ones([16, 3, 10, 32, 32]), mindspore.float32)
>>> conv3d = nn.Conv3d(in_channels=3, out_channels=32, kernel_size=(4, 3, 3))
>>> output = conv3d(x)
>>> print(output.shape)
(16, 32, 10, 32, 32)
```

3.4.6 mindspore.nn.Conv3dTranspose

```
class mindspore.nn.Conv3dTranspose(in_channels, out_channels, kernel_size, stride=1, pad_mode='same', padding=0,
                                   dilation=1, group=1, output_padding=0, has_bias=False, weight_init=None,
                                   bias_init=None, data_format='NCDHW', dtype=mstype.float32)
```

计算三维转置卷积，可以视为 `Conv3d` 对输入求梯度，也称为反卷积（实际不是真正的反卷积）。

输入的 `shape` 通常为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C_{in} 是空间维度， D_{in} , H_{in} , W_{in} 分别为特征层的深度、高度和宽度。当 `Conv3d` 和 `ConvTranspose3d` 使用相同的参数初始化时，且 `pad_mode` 设置为 "pad"，它们会在

输入的深度、高度和宽度方向上填充 $dilation * (kernel\_size - 1) - padding$ 个 0，这种情况下它们的输入 shape 和输出 shape 是互逆的。然而，当 *stride* 大于 1 时，Conv3d 会将多个输入的 shape 映射到同一个输出 shape。反卷积网络的详细介绍可以参考论文：[Deconvolutional Networks](#)。

说明：

- Atlas A2 训练系列产品暂不支持 *output_padding*。

参数：

- in_channels** (int) - Conv3dTranspose 层输入 Tensor 的空间维度。
- out_channels** (int) - Conv3dTranspose 层输出 Tensor 的空间维度。
- kernel_size** (Union[int, tuple[int]]) - 指定三维卷积核的深度、高度和宽度。数据类型为 int，或包含三个整数的 tuple。若为一个整数，则表示卷积核的深度、高度和宽度均为该整数值；若为包含三个整数的 tuple，则分别表示卷积核的深度、高度和宽度。
- stride** (Union[int, tuple[int]])，可选 - 三维卷积核的移动步长。数据类型为 int，或包含三个整型的 tuple。若为一个整数，则表示在深度、高度和宽度方向的移动步长均为该整数值；若为包含三个整数的 tuple，则分别表示在深度、高度和宽度方向的移动步长。默认值：1。
- pad_mode** (str，可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："same"。
 - "same"：在输入的深度、高度和宽度维度进行填充，使得当 *stride* 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周；若为奇数，则多余的填充量将补充在前方/底部/右侧。如果设置了此模式，*padding* 必须为 0。
 - "valid"：不对输入进行填充，返回输出可能的最大深度、高度和宽度，如果不能构成一个完整 stride，那么额外的像素将被丢弃。如果设置了此模式，*padding* 必须为 0。
 - "pad"：对输入填充指定的量。在这种模式下，在输入的深度、高度和宽度方向上填充的量由 *padding* 参数指定。如果设置此模式，*padding* 必须大于或等于 0。
- padding** (Union(int, tuple[int]))，可选 - 输入的深度、高度和宽度方向上填充的数量。数据类型为 int，或包含六个整数的 tuple。若 *padding* 是一个整数，则前部、后部、顶部、底部，左边和右边的填充都等于 *padding*；若 *padding* 是包含六个整数的 tuple，则前部、尾部、顶部、底部、左边和右边的填充分别等于填充 *padding*[0]、*padding*[1]、*padding*[2]、*padding*[3]、*padding*[4] 和 *padding*[5]。值要大于等于 0，默认值：0。
- dilation** (Union[int, tuple[int]])，可选 - 三维卷积核的膨胀尺寸。数据类型为 int，或包含三个整数的 tuple。若为一个整数，则表示在深度、高度和宽度方向的膨胀尺寸均为该整数值；若为包含三个整数的 tuple，则分别表示在深度、高度和宽度方向的膨胀尺寸。假设 $dilation = (d0, d1, d2)$ ，则卷积核在深度方向间隔 $d0 - 1$ 个元素进行采样，在高度方向间隔 $d1 - 1$ 个元素进行采样，在宽度方向间隔 $d2 - 1$ 个元素进行采样。深度、高度和宽度上取值范围分别为 [1, D]、[1, H] 和 [1, W]。默认值：1。
- group** (int，可选) - 将过滤器拆分为组，*in_channels* 和 *out_channels* 必须可被 *group* 整除。当 *group* 大于 1 时，暂不支持 Ascend 平台。默认值：1。
- output_padding** (Union(int, tuple[int]))，可选 - 输出的深度、高度和宽度方向上填充的数量。数据类型为 int，或包含 3 个整数的 tuple。如果 *output_padding* 是一个整数，则深度、高度和宽度方向的填充都等于 *output_padding*；如果 *output_padding* 是包含三个整数的 tuple，则深度、高度和宽度方向的填充分别等于填充 *output_padding*[0]、*output_padding*[1] 和 *output_padding*[2]。值要大于等于 0，默认值：0。
- has_bias** (bool，可选) - Conv3dTranspose 层是否添加偏置参数。默认值：False。
- weight_init** (Union[Tensor, str, Initializer, numbers.Number])，可选 - 权重参数的初始化方法。它可以是 Tensor、str、Initializer 或 numbers.Number。当使用 str 时，可选 "TruncatedNormal"，"Normal"，"Uniform"，"HeUniform" 和 "XavierUniform" 分布以及常量 "One" 和 "Zero" 分布的值，可接受别名

"xavier_uniform", "he_uniform", "ones" 和 "zeros"。上述字符串大小写均可。更多细节请参考 `Initializer` 的值。默认值: `None`，即权重使用 `HeUniform` 初始化。

- **bias_init** (`Union[Tensor, str, Initializer, numbers.Number]`, 可选) - 偏置参数的初始化方法。可以使用的初始化方法与 `weight_init` 相同。更多细节请参考 `Initializer` 的值。默认值: `None`，即偏差使用 `Uniform` 初始化。
- **data_format** (`str`, 可选) - 数据格式的可选值。目前仅支持 'NCDHW'。默认值: 'NCDHW'。
- **dtype** (`mindspore.dtype`, 可选) - `Parameters` 的 `dtype`。默认值: `mstype.float32`。

输入:

- **x** (`Tensor`) - shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ 的 `Tensor`。目前输入数据类型只支持 `float16` 和 `float32`。

输出:

`Tensor`, shape 为 $(N, C_{out}, D_{out}, H_{out}, W_{out})$ 。

`pad_mode` 为 "same" 时:

$$\begin{aligned} D_{out} &= \left\lfloor \frac{D_{in}}{\text{stride}[0]} + 1 \right\rfloor \\ H_{out} &= \left\lfloor \frac{H_{in}}{\text{stride}[1]} + 1 \right\rfloor \\ W_{out} &= \left\lfloor \frac{W_{in}}{\text{stride}[2]} + 1 \right\rfloor \end{aligned}$$

`pad_mode` 为 "valid" 时:

$$\begin{aligned} D_{out} &= \left\lfloor \frac{D_{in} - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1)}{\text{stride}[0]} + 1 \right\rfloor \\ H_{out} &= \left\lfloor \frac{H_{in} - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1)}{\text{stride}[1]} + 1 \right\rfloor \\ W_{out} &= \left\lfloor \frac{W_{in} - \text{dilation}[2] \times (\text{kernel\_size}[2] - 1)}{\text{stride}[2]} + 1 \right\rfloor \end{aligned}$$

`pad_mode` 为 "pad" 时:

$$\begin{aligned} D_{out} &= \left\lfloor \frac{D_{in} + \text{padding}[0] + \text{padding}[1] - (\text{dilation}[0] - 1) \times \text{kernel\_size}[0] - 1}{\text{stride}[0]} + 1 \right\rfloor \\ H_{out} &= \left\lfloor \frac{H_{in} + \text{padding}[2] + \text{padding}[3] - (\text{dilation}[1] - 1) \times \text{kernel\_size}[1] - 1}{\text{stride}[1]} + 1 \right\rfloor \\ W_{out} &= \left\lfloor \frac{W_{in} + \text{padding}[4] + \text{padding}[5] - (\text{dilation}[2] - 1) \times \text{kernel\_size}[2] - 1}{\text{stride}[2]} + 1 \right\rfloor \end{aligned}$$

异常:

- **TypeError** - `in_channels`、`out_channels` 或 `group` 不是 `int`。
- **TypeError** - `kernel_size`、`stride`、`padding`、`dilation` 或 `output_padding` 既不是 `int` 也不是 `tuple`。
- **TypeError** - 输入数据类型不是 `float16` 或 `float32`。
- **ValueError** - `in_channels`、`out_channels`、`kernel_size`、`stride` 或 `dilation` 小于 1。
- **ValueError** - `padding` 小于 0。
- **ValueError** - `pad_mode` 不是 "same", "valid" 或 "pad"。
- **ValueError** - `padding` 是长度不等于 6 的 `tuple`。
- **ValueError** - `pad_mode` 不等于 "pad" 且 `padding` 不等于 $(0, 0, 0, 0, 0, 0)$ 。
- **ValueError** - `data_format` 不是 "NCDHW"。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.ones([32, 16, 10, 32, 32]), mindspore.float32)
>>> conv3d_transpose = nn.Conv3dTranspose(in_channels=16, out_channels=3, kernel_size=(4, 6, 2),
...                                     pad_mode='pad')
>>> output = conv3d_transpose(x)
>>> print(output.shape)
(32, 3, 13, 37, 33)
```

3.4.7 mindspore.nn.Unfold

class mindspore.nn.Unfold(*ksizes, strides, rates, padding='valid'*)

从图像中提取滑窗的区域块。

输入为一个四维的 Tensor，数据格式为 (N, C, H, W)。

参数：

- **ksizes** (Union[tuple[int], list[int]]) - 滑窗大小，其格式为 [1, ksize_row, ksize_col, 1] 的 int 组成的 tuple 或 list。
- **strides** (Union[tuple[int], list[int]]) - 滑窗步长，其格式为 [1, stride_row, stride_col, 1] 的 int 组成的 tuple 或 list。
- **rates** (Union[tuple[int], list[int]]) - 滑窗元素之间的空洞个数，其格式为 [1, rate_row, rate_col, 1] 的 int 组成的 tuple 或 list。
- **padding** (str) - 填充模式，可选值有："same" 或 "valid" 的字符串，不区分大小写。默认值："valid"。
 - "same" - 指所提取的区域块的部分区域可以在原始图像之外，此部分填充为 0。
 - "valid" - 表示所取的区域块必须被原始图像所覆盖。

输入：

- **x** (Tensor) - 输入四维 Tensor，其 shape 为 [in_batch, in_depth, in_row, in_col]，其数据类型为 int。

输出：

Tensor，输出为四维 Tensor，数据类型与 x 相同，其 shape 为 (out_batch, out_depth, out_row, out_col)，且 out_batch 与 in_batch 相同。

- $out\_depth = ksize\_row * ksize\_col * in\_depth$
- $out\_row = (in\_row - (ksize\_row + (ksize\_row - 1) * (rate\_row - 1))) // stride\_row + 1$
- $out\_col = (in\_col - (ksize\_col + (ksize\_col - 1) * (rate\_col - 1))) // stride\_col + 1$

异常：

- **TypeError** - ksizes, strides 或 rates 既不是 tuple，也不是 list。
- **ValueError** - ksizes, strides 或 rates 的 shape 不是 (1, x_row, x_col, 1)。
- **ValueError** - ksizes, strides 或 rates 的第二个和第三个元素小于 1。

支持平台：

Ascend GPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> net = nn.Unfold(kernels=[1, 2, 2, 1], strides=[1, 2, 2, 1], rates=[1, 2, 2, 1])
>>> # As stated in the above code:
>>> # ksize_row = 2, ksize_col = 2, rate_row = 2, rate_col = 2, stride_row = 2, stride_col = 2.
>>> image = Tensor(np.ones([2, 3, 6, 6]), dtype=mindspore.float16)
>>> # in_batch = 2, in_depth = 3, in_row = 6, in_col = 6.
>>> # Substituting the formula to get:
>>> # out_batch = in_batch = 2
>>> # out_depth = 2 * 2 * 3 = 12
>>> # out_row = (6 - (2 + (2 - 1) * (2 - 1))) // 2 + 1 = 2
>>> # out_col = (6 - (2 + (2 - 1) * (2 - 1))) // 2 + 1 = 2
>>> output = net(image)
>>> print(output.shape)
(2, 12, 2, 2)
```

3.5 循环神经网络层

接口名	概述	支持平台
<code>mindspore.nn.RNN</code>	循环神经网络 (RNN) 层, 将具有 tanh 或 ReLU 非线性的 RNN 层应用到输入。	Ascend GPU CPU
<code>mindspore.nn.RNNCell</code>	循环神经网络单元, 激活函数是 tanh 或 relu。	Ascend GPU CPU
<code>mindspore.nn.GRU</code>	GRU (Gate Recurrent Unit) 称为门控循环单元网络, 是循环神经网络 (Recurrent Neural Network, RNN) 的一种。	Ascend GPU CPU
<code>mindspore.nn.GRUCell</code>	GRU (Gate Recurrent Unit) 称为门控循环单元。	Ascend GPU CPU
<code>mindspore.nn.LSTM</code>	长短期记忆 (LSTM) 网络, 根据输入序列和给定的初始状态计算输出序列和最终状态。	Ascend GPU CPU
<code>mindspore.nn.LSTMCell</code>	长短期记忆网络单元 (LSTMCell)。	Ascend GPU CPU

3.5.1 mindspore.nn.RNN

`class mindspore.nn.RNN(*args, **kwargs)`
循环神经网络 (RNN) 层, 将具有 tanh 或 ReLU 非线性的 RNN 层应用到输入。
对输入序列中的每个元素, 每层的计算公式如下:

$$h_t = activation(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

这里的 h_t 是在 t 时刻的隐藏状态, x_t 是在 t 时刻的输入, $h_{(t-1)}$ 是上一层在 $t - 1$ 时刻的隐藏状态, 或初始隐藏状态, W_{ih} 是每层 RNN 输入 (input gate) 计算的 weight, b_{ih} 是对应的 bias, W_{hh} 是每层 RNN 隐藏状态 (hidden state) 计算的 weight, b_{hh} 是对应的 bias。

参数:

- `input_size` (int) - 输入层输入的特征向量维度。
- `hidden_size` (int) - 隐藏层输出的特征向量维度。

- **num_layers** (int) - 堆叠 RNN 的层数。默认值: 1。
- **nonlinearity** (str) - 用于选择非线性激活函数。取值可为'tanh' 或'relu'。默认值: 'tanh'。
- **has_bias** (bool) - Cell 是否有偏置项 b_{ih} 和 b_{hh} 。默认值: True。
- **batch_first** (bool) - 指定输入 x 的第一个维度是否为 batch_size。默认值: False。
- **dropout** (float) - 指的是除第一层外每层输入时的 Dropout 概率。Dropout 的范围为 [0.0, 1.0)。默认值: 0.0。
- **bidirectional** (bool) - 指定是否为双向 RNN, 如果 bidirectional=True, 则 num_directions=2, 否则为 1。默认值: False。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: mstype.float32。

输入:

- **x** (Tensor) - 数据类型为 mindspore.float32 或 mindspore.float16, shape 为 (*seq_len, batch_size, input_size*) 或 (*batch_size, seq_len, input_size*) 的 Tensor。
- **hx** (Tensor) - 数据类型为 mindspore.float32 或 mindspore.float16, shape 为 (*num_directions * num_layers, batch_size, hidden_size*) 的 Tensor。
- **seq_length** (Tensor) - 输入 batch 的序列长度, Tensor 的 shape 为 (*batch_size*)。此输入指明真实的序列长度, 以避免使用填充后的元素计算隐藏状态, 影响最后的输出。当 x 被填充元素时, 建议使用此输入。默认值: None。

输出:

Tuple, 包含 (*output, hx_n*) 的 tuple。

- **output** (Tensor) - shape 为 (*seq_len, batch_size, num_directions * hidden_size*) 或 (*batch_size, seq_len, num_directions * hidden_size*) 的 Tensor。
- **hx_n** (Tensor) - shape 为 (*num_directions * num_layers, batch_size, hidden_size*) 的 Tensor。

异常:

- **TypeError** - *input_size*, *hidden_size* 或 *num_layers* 不是 int。
- **TypeError** - *has_bias*, *batch_first* 或 *bidirectional* 不是 bool。
- **TypeError** - *dropout* 不是 float。
- **ValueError** - *dropout* 不在 [0.0, 1.0) 范围内。
- **ValueError** - *nonlinearity* 不在 ['tanh', 'relu'] 中。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.RNN(10, 16, 2, has_bias=True, batch_first=True, bidirectional=False)
>>> x = ms.Tensor(np.ones([3, 5, 10]).astype(np.float32))
>>> h0 = ms.Tensor(np.ones([1 * 2, 3, 16]).astype(np.float32))
>>> output, hn = net(x, h0)
>>> print(output.shape)
(3, 5, 16)
```

3.5.2 mindspore.nn.RNNCell

class mindspore.nn.RNNCell (*input_size: int, hidden_size: int, has_bias: bool = True, nonlinearity: str = 'tanh', dtype=mstype.float32*)

循环神经网络单元，激活函数是 tanh 或 relu。

$$h_t = \tanh(W_{ih}x_t + b_{ih} + W_{hh}h_{(t-1)} + b_{hh})$$

其中 h_t 是在 t 时刻的隐藏状态， x_t 是在 t 时刻的输入， $h_{(t-1)}$ 是在 $t-1$ 时刻的隐藏状态，或初始隐藏状态。

参数：

- **input_size** (int) - 输入层输入的特征向量维度。
- **hidden_size** (int) - 隐藏层输出的特征向量维度。
- **has_bias** (bool) - Cell 是否有偏置项 b_{ih} 和 b_{hh} 。默认值：True。
- **nonlinearity** (str) - 用于选择非线性激活函数。取值可以是'tanh'或'relu'。默认值：'tanh'。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **x** (Tensor) - 输入 Tensor，其 shape 为 (*batch_size, input_size*)。
- **hx** (Tensor) - 输入 Tensor，其数据类型为 mindspore.float32 及 shape 为 (*batch_size, hidden_size*)。

输出：

- **hx'** (Tensor) - shape 为 (*batch_size, hidden_size*) 的 Tensor。

异常：

- **TypeError** - *input_size* 或 *hidden_size* 不是 int 或不大于 0。
- **TypeError** - *has_bias* 不是 bool。
- **ValueError** - *nonlinearity* 不在 ['tanh', 'relu'] 中。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.RNNCell(10, 16)
>>> x = ms.Tensor(np.ones([5, 3, 10]).astype(np.float32))
>>> hx = ms.Tensor(np.ones([3, 16]).astype(np.float32))
>>> output = []
>>> for i in range(5):
...     hx = net(x[i], hx)
...     output.append(hx)
>>> print(output[0].shape)
(3, 16)
```

3.5.3 mindspore.nn.GRU

class mindspore.nn.GRU(*args, **kwargs)

GRU (Gate Recurrent Unit) 称为门控循环单元网络, 是循环神经网络 (Recurrent Neural Network, RNN) 的一种。根据输出序列和给定的初始状态计算输出序列和最终状态。

应用 GRU 层到输入中。

GRU 网络模型中有两个门。一个是更新门, 另一个是重置门。将两个连续的时间节点表示为 $t-1$ 和 t 。给定一个在时刻 t 的输入 x_t , 一个隐藏状态 h_{t-1} , 在时刻 t 的更新门和重置门使用门控机制计算。更新门 z_t 用于控制前一时刻的状态信息被带入到当前状态中的程度。重置门 r_t 控制前一状态有多少信息被写入到当前候选集 n_t 上。完整的公式如下。

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\ h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)} \end{aligned}$$

其中 σ 是 sigmoid 激活函数, $*$ 是乘积。 W, b 是公式中输出和输入之间的可学习权重。例如, W_{ir}, b_{ir} 是用于将输入 x 转换为 r 的权重和偏置。详见论文 [Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation](#)。

说明: 当 GRU 运行在 Ascend 上时, hidden size 仅支持 16 的倍数。

参数:

- **input_size** (int) - 输入的大小。
- **hidden_size** (int) - 隐藏状态大小。
- **num_layers** (int) - 网络层数。默认值: 1。
- **has_bias** (bool) - cell 是否有偏置项 b_{in} 和 b_{hn} 。默认值: True。
- **batch_first** (bool) - 指定输入 x 的第一个维度是否为 batch_size。默认值: False。
- **dropout** (float) - 指的是除第一层外每层输入时的 Dropout 概率。默认值: 0.0。Dropout 的范围为 [0.0, 1.0)。
- **bidirectional** (bool) - 是否为双向 GRU。如果 bidirectional=True, 则 num_directions=2, 为双向 GRU。否则为 1, 单向 GRU。默认值: False。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: mstype.float32。

输入:

- **x** (Tensor) - 数据类型为 mindspore.float32、shape 为 $(seq\_len, batch\_size, input\_size)$ 或 $(batch\_size, seq\_len, input\_size)$ 的 Tensor。
- **hx** (Tensor) - 数据类型为 mindspore.float32、shape 为 $(num\_directions * num\_layers, batch\_size, hidden\_size)$ 的 Tensor。
- **seq_length** (Tensor) - 输入 batch 中每个序列的长度。shape 为 $(batch\_size)$ 的 Tensor。默认值: None。此输入指示填充前的真实序列长度, 避免填充元素被用于计算隐藏状态而影响最终输出。当 x 含填充元素时, 建议使用此输入。

输出:

Tuple, 包含 $(output, h_n)$ 的 tuple。

- **output** (Tensor) - shape 为 $(seq\_len, batch\_size, num\_directions * hidden\_size)$ 或 $(batch\_size, seq\_len, num\_directions * hidden\_size)$ 的 Tensor。

- **hx_n** (Tensor) - shape 为 $(num\_directions * num\_layers, batch\_size, hidden\_size)$ 的 Tensor。

异常:

- **TypeError** - *input_size* , *hidden_size* 或 *num_layers* 不是整数。
- **TypeError** - *has_bias* , *batch_first* 或 *bidirectional* 不是 bool。
- **TypeError** - *dropout* 既不是浮点数也不是整数。
- **ValueError** - *dropout* 不在 [0.0, 1.0) 范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.GRU(10, 16, 2, has_bias=True, batch_first=True, bidirectional=False)
>>> x = ms.Tensor(np.ones([3, 5, 10]).astype(np.float32))
>>> h0 = ms.Tensor(np.ones([1 * 2, 3, 16]).astype(np.float32))
>>> output, hn = net(x, h0)
>>> print(output.shape)
(3, 5, 16)
```

3.5.4 mindspore.nn.GRUCell

class mindspore.nn.GRUCell (*input_size: int, hidden_size: int, has_bias: bool = True, dtype=mstype.float32*)

GRU (Gate Recurrent Unit) 称为门控循环单元。

$$\begin{aligned} r &= \sigma(W_{ir}x + b_{ir} + W_{hr}h + b_{hr}) \\ z &= \sigma(W_{iz}x + b_{iz} + W_{hz}h + b_{hz}) \\ n &= \tanh(W_{in}x + b_{in} + r * (W_{hn}h + b_{hn})) \\ h' &= (1 - z) * n + z * h \end{aligned}$$

这里 σ 是 sigmoid 激活函数, $*$ 是乘积。 W, b 是公式中输出和输入之间的可学习权重, h 是隐藏层状态 (hidden state), r 是重置门 (reset gate), z 是更新门 (update gate), n 是第 n 层。例如, W_{ir}, b_{ir} 是用于将输入 x 转换为 r 的权重和偏置。详见论文 [Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation](#)。

参数:

- **input_size** (int) - 输入的大小。
- **hidden_size** (int) - 隐藏状态大小。
- **has_bias** (bool) - cell 是否有偏置项 b_{in} 和 b_{hn} 。默认值: True。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: mstype.float32。

输入:

- **x** (Tensor) - shape 为 $(batch\_size, input\_size)$ 的 Tensor。
- **hx** (Tensor) - 数据类型为 mindspore.float32, shape 为 $(batch\_size, hidden\_size)$ 的 Tensor。

输出:

- **hx'** (Tensor) - shape 为 $(batch\_size, hidden\_size)$ 的 Tensor。

异常:

- **TypeError** - *input_size*、*hidden_size* 不是 int。
- **TypeError** - *has_bias* 不是 bool 值。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.GRUCell(10, 16)
>>> x = ms.Tensor(np.ones([5, 3, 10]).astype(np.float32))
>>> hx = ms.Tensor(np.ones([3, 16]).astype(np.float32))
>>> output = []
>>> for i in range(5):
...     hx = net(x[i], hx)
...     output.append(hx)
>>> print(output[0].shape)
(3, 16)
```

3.5.5 mindspore.nn.LSTM

class mindspore.nn.LSTM(*args, **kwargs)

长短期记忆（LSTM）网络，根据输入序列和给定的初始状态计算输出序列和最终状态。

在 LSTM 模型中，有两条管道连接两个连续的 Cell，一条是 Cell 状态管道，另一条是隐藏状态管道。将两个连续的时间节点表示为 $t-1$ 和 t 。指定在 t 时刻输入 x_t ，在 $t-1$ 时刻的隐藏状态 h_{t-1} 和 Cell 状态 c_{t-1} 。在 t 时刻的 Cell 状态和隐藏状态使用门控机制计算得到。输入门 i_t 计算出候选值。遗忘门 f_t 决定是否让 h_{t-1} 学到的信息通过或部分通过。输出门 o_t 决定哪些信息输出。候选 Cell 状态 $\tilde{c}_t$ 是用当前输入计算的。最后，使用遗忘门、输入门、输出门计算得到当前时刻的 Cell 状态 c_t 和隐藏状态 h_t 。完整的公式如下。

$$\begin{aligned} i_t &= \sigma(W_{ix}x_t + b_{ix} + W_{ih}h_{(t-1)} + b_{ih}) \\ f_t &= \sigma(W_{fx}x_t + b_{fx} + W_{fh}h_{(t-1)} + b_{fh}) \\ \tilde{c}_t &= \tanh(W_{cx}x_t + b_{cx} + W_{ch}h_{(t-1)} + b_{ch}) \\ o_t &= \sigma(W_{ox}x_t + b_{ox} + W_{oh}h_{(t-1)} + b_{oh}) \\ c_t &= f_t * c_{(t-1)} + i_t * \tilde{c}_t \\ h_t &= o_t * \tanh(c_t) \end{aligned}$$

其中 σ 是 sigmoid 激活函数， $*$ 是乘积。 W, b 是公式中输出和输入之间的可学习权重。例如， W_{ix}, b_{ix} 是用于从输入 x 转换为 i 的权重和偏置。

详细信息可见论文 [LONG SHORT-TERM MEMORY](#) 和 [Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling](#)。

LSTM 隐藏了整个循环神经网络在序列时间步 (Time step) 上的循环，送入输入序列、初始状态，即可获得每个时间步的隐藏状态 (hidden state) 拼接而成的矩阵，以及最后一个时间步对应的隐状态。我们使用最后的一个时间步的隐藏状态作为输入句子的编码特征，送入下一层。公式为：

$$h_{0:n}, (h_n, c_n) = LSTM(x_{0:n}, (h_0, c_0))$$

参数：

- **input_size** (int) - 输入的大小。
- **hidden_size** (int) - 隐藏状态大小。

- **num_layers** (int) - 网络层数。默认值: 1。
- **has_bias** (bool) - Cell 是否有偏置项 b_{ih} 和 b_{fh} 。默认值: True。
- **batch_first** (bool) - 指定输入 x 的第一个维度是否为 **batch_size**。默认值: False。
- **dropout** (float, int) - 指的是除第一层外每层输入时的 dropout 概率。默认值: 0。dropout 的范围为 [0.0, 1.0)。
- **bidirectional** (bool) - 是否为双向 LSTM。如果 **bidirectional** 是 True, **directions** 数量为 2, 否则 **directions** 数量为 1。默认值: False。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: `mstype.float32`。

输入:

- **x** (Tensor) - shape 为 $(seq\_len, batch\_size, input\_size)$ 或 $(batch\_size, seq\_len, input\_size)$ 的 Tensor。
- **hx** (tuple) - 两个 Tensor(h_0, c_0) 的元组, 数据类型为 `mindspore.float32` 或 `mindspore.float16`, shape 为 $(num\_directions * num\_layers, batch\_size, hidden\_size)$ 。
- **seq_length** (Tensor) - 输入 batch 的序列长度。Tensor 的 shape 为 $(batch\_size)$ 。默认: None。这里输入指明真实的序列长度, 以避免使用填充后的元素计算隐藏状态, 影响最后的输出。推荐这种输入方法。

输出:

Tuple, 包含 $(output, (h_n, c_n))$ 的元组。

- **output** (Tensor) - shape 为 $(seq\_len, batch\_size, num\_directions * hidden\_size)$ 的 Tensor。
- **hx_n** (tuple) - 两个 Tensor (h_n, c_n) 的元组, shape 都是 $(num\_directions * num\_layers, batch\_size, hidden\_size)$ 。

异常:

- **TypeError** - *input_size*, *hidden_size* 或 *num_layers* 不是 int。
- **TypeError** - *has_bias*, *batch_first* 或 *bidirectional* 不是 bool。
- **TypeError** - *dropout* 既不是 float 也不是 int。
- **ValueError** - *dropout* 不在 [0.0, 1.0) 范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.LSTM(10, 16, 2, has_bias=True, batch_first=True, bidirectional=False)
>>> x = ms.Tensor(np.ones([3, 5, 10]).astype(np.float32))
>>> h0 = ms.Tensor(np.ones([1 * 2, 3, 16]).astype(np.float32))
>>> c0 = ms.Tensor(np.ones([1 * 2, 3, 16]).astype(np.float32))
>>> output, (hn, cn) = net(x, (h0, c0))
>>> print(output.shape)
(3, 5, 16)
```

3.5.6 mindspore.nn.LSTMCell

class mindspore.nn.LSTMCell (input_size: int, hidden_size: int, has_bias: bool = True, dtype=mstype.float32)
长短期记忆网络单元 (LSTMCell)。

公式如下：

$$\begin{aligned} i_t &= \sigma(W_{ix}x_t + b_{ix} + W_{ih}h_{(t-1)} + b_{ih}) \\ f_t &= \sigma(W_{fx}x_t + b_{fx} + W_{fh}h_{(t-1)} + b_{fh}) \\ \tilde{c}_t &= \tanh(W_{cx}x_t + b_{cx} + W_{ch}h_{(t-1)} + b_{ch}) \\ o_t &= \sigma(W_{ox}x_t + b_{ox} + W_{oh}h_{(t-1)} + b_{oh}) \\ c_t &= f_t * c_{(t-1)} + i_t * \tilde{c}_t \\ h_t &= o_t * \tanh(c_t) \end{aligned}$$

其中 σ 是 sigmoid 函数, $*$ 是乘积。 W, b 是公式中输出和输入之间的可学习权重。例如, W_{ix}, b_{ix} 是表示从输入 x 转换为 i 的权重和偏置。详见论文 [LONG SHORT-TERM MEMORY](#) 和 [Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling](#)。

nn 层封装的 LSTMCell 可以简化为如下公式：

$$h', c' = LSTMCell(x, (h_0, c_0))$$

参数：

- **input_size** (int) - 输入的大小。
- **hidden_size** (int) - 隐藏状态大小。
- **has_bias** (bool) - cell 是否有偏置 b_{ih} 和 b_{hh} 。默认值：True。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **x** (Tensor) - shape 为 $(batch\_size, input\_size)$ 的 Tensor。
- **hx** (tuple) - 两个 Tensor(h_0, c_0) 的元组, 其数据类型为 mindspore.float32, shape 为 $(batch\_size, hidden\_size)$ 。

输出：

- **hx'** (Tensor) - 两个 Tensor(h', c') 的元组, 其 shape 为 $(batch\_size, hidden\_size)$ 。

异常：

- **TypeError** - input_size, hidden_size 不是整数。
- **TypeError** - has_bias 不是 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.LSTMCell(10, 16)
>>> x = ms.Tensor(np.ones([5, 3, 10]).astype(np.float32))
>>> h = ms.Tensor(np.ones([3, 16]).astype(np.float32))
>>> c = ms.Tensor(np.ones([3, 16]).astype(np.float32))
>>> output = []
>>> for i in range(5):
```

(下页继续)

(续上页)

```
...     hx = net(x[i], (h, c))
...     output.append(hx)
>>> print(output[0][0].shape)
(3, 16)
```

3.6 Transformer 层

接口名	概述	支持平台
<code>mindspore.nn.MultiheadAttention</code>	论文 Attention Is All You Need 中描述了多头注意力的实现。	Ascend GPU CPU
<code>mindspore.nn.TransformerEncoderLayer</code>	Transformer 的编码器层。	Ascend GPU CPU
<code>mindspore.nn.TransformerDecoderLayer</code>	Transformer 的解码器层。	Ascend GPU CPU
<code>mindspore.nn.TransformerEncoder</code>	Transformer 编码器模块，多层 <code>mindspore.nn.TransformerEncoderLayer</code> 的堆叠，包括 <code>MultiheadAttention</code> 层和 <code>FeedForward</code> 层。	Ascend GPU CPU
<code>mindspore.nn.TransformerDecoder</code>	Transformer 的解码器模块，多层 <code>mindspore.nn.TransformerDecoderLayer</code> 的堆叠，包括 <code>Self Attention</code> 层、 <code>MultiheadAttention</code> 层和 <code>FeedForward</code> 层。	Ascend GPU CPU
<code>mindspore.nn.Transformer</code>	Transformer 模块，包括编码器和解码器。	Ascend GPU CPU

3.6.1 mindspore.nn.MultiheadAttention

```
class mindspore.nn.MultiheadAttention (embed_dim, num_heads, dropout=0.0, has_bias=True, add_bias_kv=False,
                                       add_zero_attn=False, kdim=None, vdim=None, batch_first=False,
                                       dtype=mstype.float32)
```

论文 [Attention Is All You Need](#) 中描述了多头注意力的实现。给定 query 向量、key 向量和 value 向量，注意力计算流程如下：

$$MultiHeadAttention(query, key, value) = Concat(head_1, \dots, head_h)W^O$$

其中， $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$ ， W^O 、 W_i^Q 、 W_i^K 、 W_i^V 是权重矩阵。注意：默认输入、输出投影层中带有偏置参数。

如果 query、key 和 value 相同，则上述即为自注意力机制的计算过程。

参数：

- **embed_dim** (int) - 模型的总维数。
- **num_heads** (int) - 并行注意力头的数量。`num_heads` 需要能够整除 `embed_dim`（每个头的维数为 `embed_dim // num_heads`）。
- **dropout** (float, 可选) - 应用到输入 `attn_output_weights` 上的随机丢弃比例。默认值：0.0。
- **has_bias** (bool, 可选) - 是否给输入、输出投射层添加偏置。默认值：True。
- **add_bias_kv** (bool, 可选) - 是否给 key、value 序列的零维添加偏置。默认值：False。
- **add_zero_attn** (bool, 可选) - 是否给 key、value 序列的一维添加 0。默认值：False。

- **kdim** (int, 可选) - key 的总特征数。默认值: None (即 $kdim=embed\_dim$)。
- **vdim** (int, 可选) - value 的总特征数。默认值: None (即 $vdim=embed\_dim$)。
- **batch_first** (bool, 可选) - 如果为 True, 则输入输出 Tensor 的 shape 为 $(batch, seq, feature)$, 否则 shape 为 $(seq, batch, feature)$ 。默认值: False。
- **dtype** (*mindspore.dtype*, 可选) - Parameter 的数据类型。默认值: `mstype.float32`。

输入:

- **query** (Tensor) - Query 矩阵。当输入非 Batch 数据时, shape 为: (L, E_q) 。当输入 Batch 数据, 参数 $batch\_first=False$ 时, shape 为 (L, N, E_q) , 当 $batch\_first=True$ 时, shape 为 (N, L, E_q) 。其中, L 为目标序列的长度, N 为 batch size, E_q 为 Query 矩阵的维数 $embed\_dim$ 。数据类型: float16、float32 或者 float64。注意力机制通过 Query 与 Key-Value 运算以生成最终输出。
- **key** (Tensor) - Key 矩阵。当输入非 Batch 数据时, shape 为: (S, E_k) 。当输入 Batch 数据, 参数 $batch\_first=False$ 时, shape 为 (S, N, E_k) , 当 $batch\_first=True$ 时, shape 为 (N, S, E_k) 。其中, S 为源序列的长度, N 为 batch size, E_k 为 Key 矩阵的维数 $kdim$ 。数据类型: float16、float32 或者 float64。
- **value** (Tensor) - Value 矩阵。当输入非 Batch 数据时, shape 为: (S, E_v) 。当输入 Batch 数据, 参数 $batch\_first=False$ 时, shape 为 (S, N, E_v) , 当 $batch\_first=True$ 时, shape 为 (N, S, E_v) 。其中, S 为源序列的长度, N 为 batch size, E_v 为 Value 矩阵的维数 $vdim$ 。数据类型: float16、float32 或者 float64。
- **key_padding_mask** (Tensor, 可选) - 如果指定此值, 则表示 shape 为 (N, S) 的掩码将被用于 key。当输入非 Batch 数据时, shape 为: (S) 。支持 Bool 和 float 类型。如果输入 Tensor 为 Bool 类型, 则 key 中对应为 True 的位置将在 Attention 计算时被忽略。如果输入 Tensor 为 float 类型, 则将直接与 key 相加。float 支持数据类型: float16、float32 或者 float64。默认值: None。
- **need_weights** (bool, 可选) - 是否需要返回 *attn_output_weights*, 如果为 True, 则输出包含 *attn_output_weights*。默认值: True。
- **attn_mask** (Tensor, 可选) - 如果指定此值, 则表示 shape 为 (L, S) 或 $(N \cdot num\_heads, L, S)$ 的掩码将被用于 Attention 计算。其中 N 为 batch size, L 为目标序列长度, S 为源序列长度。如果输入为二维矩阵, 则将自动沿 batch 维广播至三维矩阵。若为三维矩阵, 则允许沿 batch 维使用不同的掩码。如果输入 Tensor 为 Bool 类型, 则值为 True 对应位置允许被注意力计算。如果输入 Tensor 为 float 类型, 则将直接与注意力权重相加。float 支持数据类型: float16、float32 或者 float64。默认值: None。
- **average_attn_weights** (bool, 可选) - 如果为 True, 则返回值 *attn_weights* 为注意力头的平均值。如果为 False, 则 *attn_weights* 分别返回每个注意力头的值。本参数仅在 $need\_weights=True$ 时生效。默认值: True。

输出:

Tuple, 表示一个包含 (*attn_output*, *attn_output_weights*) 的元组。

- **attn_output** - 注意力机制的输出。当输入非 Batch 数据时, shape 为: (L, E) 。当输入 Batch 数据, 参数 $batch\_first=False$ 时, shape 为 (L, N, E) , 当 $batch\_first=True$ 时, shape 为 (N, L, E) 。其中, L 为目标序列的长度, N 为 batch size, E 为模型的总维数 $embed\_dim$ 。
- **attn_output_weights** - 仅当 $need\_weights=True$ 时返回。如果 $average\_attn\_weights=True$, 则返回值 *attn_weights* 为注意力头的平均值。当输入非 Batch 数据时, shape 为: (L, S) , 当输入 Batch 数据时, shape 为 (N, L, S) 。其中 N 为 batch size, L 为目标序列的长度, S 为源序列长度。如果 $average\_attn\_weights=False$, 分别返回每个注意力头的值。当输入非 Batch 数据时, shape 为: (num_heads, L, S) , 当输入 Batch 数据时, shape 为 (N, num_heads, L, S) 。

异常:

- **ValueError** - 如果 $embed\_dim$ 不能被 num_heads 整除。
- **TypeError** - 如果 $key\_padding\_mask$ 不是 bool 或 float 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> embed_dim, num_heads = 128, 8
>>> seq_length, batch_size = 10, 8
>>> query = ms.Tensor(np.random.randn(seq_length, batch_size, embed_dim), ms.float32)
>>> key = ms.Tensor(np.random.randn(seq_length, batch_size, embed_dim), ms.float32)
>>> value = ms.Tensor(np.random.randn(seq_length, batch_size, embed_dim), ms.float32)
>>> multihead_attn = ms.nn.MultiheadAttention(embed_dim, num_heads)
>>> attn_output, attn_output_weights = multihead_attn(query, key, value)
>>> print(attn_output.shape)
(10, 8, 128)
```

3.6.2 mindspore.nn.TransformerEncoderLayer

```
class mindspore.nn.TransformerEncoderLayer (d_model: int, nhead: int, dim_feedforward: int = 2048, dropout:
float = 0.1, activation: Union[str, Cell, callable] = 'relu',
layer_norm_eps: float = 1e-5, batch_first: bool = False, norm_first:
bool = False, dtype=mstype.float32)
```

Transformer 的编码器层。Transformer 编码器的单层实现，包括 MultiheadAttention 层和 FeedForward 层。

参数:

- **d_model** (int) - 输入的特征数。
- **nhead** (int) - MultiheadAttention 模块中注意力头的数量。
- **dim_feedforward** (int) - FeedForward 层的维数。默认值: 2048。
- **dropout** (float) - 随机丢弃比例。默认值: 0.1。
- **activation** (Union[str, callable, Cell]) - 中间层的激活函数，可以输入字符串 ("relu"、"gelu")、函数接口 (`mindspore.ops.relu()`、`mindspore.ops.gelu()`) 或激活函数层实例 (`mindspore.nn.ReLU`、`mindspore.nn.GELU`)。默认值: 'relu'。
- **layer_norm_eps** (float) - LayerNorm 层的 eps 值，默认值: 1e-5。
- **batch_first** (bool) - 如果为 True 则输入输出 shape 为 (batch, seq, feature)，反之，shape 为 (seq, batch, feature)。默认值: False。
- **norm_first** (bool) - 如果为 True，则 LayerNorm 层位于 MultiheadAttention 层和 FeedForward 层之前，反之，位于其后。默认值: False。
- **dtype** (`mindspore.dtype`) - Parameter 的数据类型。默认值: `mstype.float32`。

输入:

- **src** (Tensor) - 源序列。如果源序列没有 batch，shape 是 (S, E)；否则如果 `batch_first=False`，则 shape 为 (S, N, E)，如果 `batch_first=True`，则 shape 为 (N, S, E)。(S) 是源序列的长度，(N) 是 batch 个数，(E) 是特性个数。数据类型: float16、float32 或者 float64。
- **src_mask** (Tensor, 可选) - 源序列的掩码矩阵。shape 是 (S, S) 或 (N * nhead, S, S)。数据类型: float16、float32、float64 或者 bool。默认值: None。
- **src_key_padding_mask** (Tensor, 可选) - 源序列 Key 矩阵的掩码矩阵。如果目标序列没有 batch，shape 是 (S)，否则 shape 为 (N, S)。数据类型: float16、float32、float64 或者 bool。默认值: None。

输出:

Tensor。Tensor 的 shape 和 dtype 与 *src* 一致。

异常：

- **ValueError** - 如果 *activation* 不是 str、callable 或 Cell 的实例。
- **ValueError** - 如果 *activation* 不是 `mindspore.nn.ReLU`、`mindspore.nn.GELU` 的子类、`mindspore.ops.relu()` 或 `mindspore.ops.gelu()`、“relu”或“gelu”。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> encoder_layer = ms.nn.TransformerEncoderLayer(d_model=512, nhead=8)
>>> src = ms.Tensor(np.random.rand(10, 32, 512), ms.float32)
>>> out = encoder_layer(src)
>>> print(out.shape)
(10, 32, 512)
>>> # Alternatively, when batch_first=True:
>>> encoder_layer = ms.nn.TransformerEncoderLayer(d_model=512, nhead=8, batch_first=True)
>>> src = ms.Tensor(np.random.rand(32, 10, 512), ms.float32)
>>> out = encoder_layer(src)
>>> print(out.shape)
(32, 10, 512)
```

3.6.3 mindspore.nn.TransformerDecoderLayer

```
class mindspore.nn.TransformerDecoderLayer (d_model: int, nhead: int, dim_feedforward: int = 2048, dropout:
float = 0.1, activation: Union[str, Cell, callable] = 'relu',
layer_norm_eps: float = 1e-5, batch_first: bool = False, norm_first:
bool = False, dtype=mstype.float32)
```

Transformer 的解码器层。Transformer 解码器的单层实现，包括 Self Attention 层、MultiheadAttention 层和 FeedForward 层。

参数：

- **d_model** (int) - 输入的特征数。
- **nhead** (int) - MultiheadAttention 模块中注意力头的数量。
- **dim_feedforward** (int) - FeedForward 层的维数。默认值：2048。
- **dropout** (float) - 随机丢弃比例。默认值：0.1。
- **activation** (Union[str, callable, Cell]) - 中间层的激活函数，可以输入字符串（"relu"、"gelu"）、函数接口（`mindspore.ops.relu()`、`mindspore.ops.gelu()`）或激活函数层实例（`mindspore.nn.ReLU`、`mindspore.nn.GELU`）。默认值：'relu'。
- **layer_norm_eps** (float) - LayerNorm 层的 eps 值，默认值：1e-5。
- **batch_first** (bool) - 如果为 True 则输入输出 shape 为 (batch, seq, feature)，反之，shape 为 (seq, batch, feature)。默认值：False。
- **norm_first** (bool) - 如果为 True，则 LayerNorm 层位于 Self Attention 层、MultiheadAttention 层和 FeedForward 层之前，反之，位于其后。默认值：False。
- **dtype** (`mindspore.dtype`) - Parameter 的数据类型。默认值：mstype.float32。

输入：

- **tgt** (Tensor) - 目标序列。如果目标序列没有 batch, shape 是 (T, E) ；否则如果 batch_first=False, 则 shape 为 (T, N, E) , 如果 batch_first=True, 则 shape 为 (N, T, E) 。 (T) 是目标序列的长度。数据类型: float16、float32 或者 float64。
- **memory** (Tensor) - TransformerEncoder 的最后一层输出序列。数据类型: float16、float32 或者 float64。
- **tgt_mask** (Tensor, 可选) - 目标序列的掩码矩阵。shape 是 (T, T) 或 $(N * nhead, T, T)$ 。数据类型: float16、float32、float64 或者 bool。默认值: None。
- **memory_mask** (Tensor, 可选) - memory 序列的掩码矩阵。shape 是 (T, S) 。数据类型: float16、float32、float64 或者布尔。默认值: None。
- **tgt_key_padding_mask** (Tensor, 可选) - 目标序列 Key 矩阵的掩码矩阵。如果目标序列没有 batch, shape 是 (T) ；否则 shape 为 (N, T) 。数据类型: float16、float32、float64 或者 bool。默认值: None
- **memory_key_padding_mask** (Tensor, 可选) - memory 序列 Key 矩阵的掩码矩阵 ∂ 。如果目标序列没有 batch, shape 是 (S) ；否则 shape 为 (N, S) 。数据类型: float16、float32、float64 或者布尔。默认值: None。

输出：

Tensor。Tensor 的 shape 和 dtype 与 *tgt* 一致。

异常：

- **ValueError** - 如果 *activation* 不是 str、callable 或 Cell 的实例。
- **ValueError** - 如果 *activation* 不是 `mindspore.nn.ReLU`、`mindspore.nn.GELU` 的子类、`mindspore.ops.relu()`、`mindspore.ops.gelu()`、"relu" 或 "gelu"。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> decoder_layer = ms.nn.TransformerDecoderLayer(d_model=512, nhead=8)
>>> memory = ms.Tensor(np.random.rand(10, 32, 512), ms.float32)
>>> tgt = ms.Tensor(np.random.rand(20, 32, 512), ms.float32)
>>> out = decoder_layer(tgt, memory)
>>> print(out.shape)
(20, 32, 512)
>>> # Alternatively, when `batch_first` is ``True``:
>>> decoder_layer = ms.nn.TransformerDecoderLayer(d_model=512, nhead=8, batch_first=True)
>>> memory = ms.Tensor(np.random.rand(32, 10, 512), ms.float32)
>>> tgt = ms.Tensor(np.random.rand(32, 20, 512), ms.float32)
>>> out = decoder_layer(tgt, memory)
>>> print(out.shape)
(32, 20, 512)
```

3.6.4 mindspore.nn.TransformerEncoder

class mindspore.nn.**TransformerEncoder** (*encoder_layer, num_layers, norm=None*)

Transformer 编码器模块，多层 `mindspore.nn.TransformerEncoderLayer` 的堆叠，包括 MultiheadAttention 层和 FeedForward 层。可以使用此模块构造 BERT(<https://arxiv.org/abs/1810.04805>) 模型。

参数：

- **encoder_layer** (Cell) - `mindspore.nn.TransformerEncoderLayer` 的实例。
- **num_layers** (int) - 编码器层数。
- **norm** (Cell, 可选) - 自定义 LayerNorm 层。默认值：None。

输入：

- **src** (Tensor) - 源序列。如果源序列没有 batch，shape 是 (S, E) ；否则如果 `mindspore.nn.TransformerEncoderLayer` 中 `batch_first=False`，则 shape 为 (S, N, E) ，如果 `batch_first=True`，则 shape 为 (N, S, E) 。 (S) 是源序列的长度， (N) 是 batch 个数， (E) 是特性个数。数据类型：float16、float32 或者 float64。
- **src_mask** (Tensor, 可选) - 源序列的掩码矩阵。shape 是 (S, S) 或 $(N * nhead, S, S)$ 。其中 `nhead` 是 `mindspore.nn.TransformerEncoderLayer` 中的 `nhead` 参数。数据类型：float16、float32、float64 或者 bool。默认值：None。
- **src_key_padding_mask** (Tensor, 可选) - 源序列 Key 矩阵的掩码矩阵。如果目标序列没有 batch，shape 是 (S) ，否则 shape 为 (N, S) 。数据类型：float16、float32、float64 或者 bool。默认值：None。

输出：

Tensor。Tensor 的 shape 和 dtype 与 `src` 一致。

异常：

- **AssertionError** - 如果 `src_key_padding_mask` 不是 bool 或 float 类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> encoder_layer = ms.nn.TransformerEncoderLayer(d_model=512, nhead=8)
>>> transformer_encoder = ms.nn.TransformerEncoder(encoder_layer, num_layers=6)
>>> src = ms.Tensor(np.random.rand(10, 32, 512), ms.float32)
>>> out = transformer_encoder(src)
>>> print(out.shape)
(10, 32, 512)
```

3.6.5 mindspore.nn.TransformerDecoder

class mindspore.nn.**TransformerDecoder** (*decoder_layer, num_layers, norm=None*)

Transformer 的解码器模块，多层 `mindspore.nn.TransformerDecoderLayer` 的堆叠，包括 Self Attention 层、MultiheadAttention 层和 FeedForward 层。

参数：

- **decoder_layer** (Cell) - `mindspore.nn.TransformerDecoderLayer` 的实例。
- **num_layers** (int) - 解码器层数。

- **norm** (Cell, 可选) - 层标准化模块。默认值: None。

输入:

- **tgt** (Tensor) - 目标序列。如果目标序列没有 batch, 则 shape 为 (T, E) 。否则, 如果 `mindspore.nn.TransformerDecoderLayer` 中 `batch_first=False`, 则 shape 为 (T, N, E) ; 如果 `batch_first=True`, 则 shape 为 (N, T, E) 。其中, (T) 是目标序列的长度, (N) 是 batch 个数, (E) 是特性个数。数据类型为 float16、float32 或者 float64。
- **memory** (Tensor) - TransformerEncoder 的最后一层输出序列。数据类型: float16、float32 或者 float64。
- **tgt_mask** (Tensor, 可选) - 目标序列的掩码矩阵。shape 为 (T, T) 或 $(N * nhead, T, T)$ 。其中 `nhead` 是 `mindspore.nn.TransformerDecoderLayer` 中的 `nhead` 参数。数据类型为 float16、float32、float64 或者 bool。默认值: None。
- **memory_mask** (Tensor, 可选) - memory 序列的掩码矩阵。shape 为 (T, S) 。数据类型为 float16、float32、float64 或者 bool。默认值: None。
- **tgt_key_padding_mask** (Tensor, 可选) - 目标序列 Key 矩阵的掩码矩阵。shape 为 (T) 。数据类型为 float16、float32、float64 或者 bool。默认值: None。
- **memory_key_padding_mask** (Tensor, 可选) - memory 序列 Key 矩阵的掩码矩阵。如果目标序列没有 batch, shape 为 (S) ; 否则 shape 为 (N, S) 。数据类型为 float16、float32、float64 或者 bool。默认值: None。

输出:

Tensor。Tensor 的 shape 和 dtype 与 `tgt` 一致。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> decoder_layer = ms.nn.TransformerDecoderLayer(d_model=512, nhead=8)
>>> transformer_decoder = ms.nn.TransformerDecoder(decoder_layer, num_layers=6)
>>> memory = ms.Tensor(np.random.rand(10, 32, 512), ms.float32)
>>> tgt = ms.Tensor(np.random.rand(20, 32, 512), ms.float32)
>>> out = transformer_decoder(tgt, memory)
>>> print(out.shape)
(20, 32, 512)
```

3.6.6 mindspore.nn.Transformer

```
class mindspore.nn.Transformer(d_model: int = 512, nhead: int = 8, num_encoder_layers: int = 6, num_decoder_layers:
    int = 6, dim_feedforward: int = 2048, dropout: float = 0.1, activation: Union[str, Cell,
    callable] = 'relu', custom_encoder: Optional[Cell] = None, custom_decoder:
    Optional[Cell] = None, layer_norm_eps: float = 1e-5, batch_first: bool = False,
    norm_first: bool = False, dtype=mstype.float32)
```

Transformer 模块, 包括编码器和解码器。本模块与原论文的实现不同, 原论文在 LayerNorm 前使用了残差模块。且默认的隐藏层激活函数为 `gelu`。详情可见 [Attention is all you need](#)。

参数:

- **d_model** (int) - Encoder 或 Decoder 输入的特征数。默认值: 512。
- **nhead** (int) - MultiheadAttention 模块中注意力头的数量。默认值: 8。
- **num_encoder_layers** (int) - Encoder 的层数。默认值: 6。

- **num_decoder_layers** (int) - Decoder 的层数。默认值：6。
- **dim_feedforward** (int) - FeedForward 层的维数。默认值：2048。
- **dropout** (float) - 随机丢弃比例。默认值：0.1。
- **activation** (Union[str, callable, Cell]) - 中间层的激活函数，可以输入字符串 ("relu"、"gelu")、函数接口 (`mindspore.ops.relu()`、`mindspore.ops.gelu()`) 或激活函数层实例 (`mindspore.nn.ReLU`、`mindspore.nn.GELU`)。默认值：'relu'。
- **custom_encoder** (Cell) - 自定义 Encoder 层。默认值：None。
- **custom_decoder** (Cell) - 自定义 Decoder 层。默认值：None。
- **layer_norm_eps** (float) - LayerNorm 层的 eps 值，默认值：1e-5。
- **batch_first** (bool) - 如果为 True 则输入输出 shape 为 (batch, seq, feature)，反之，shape 为 (seq, batch, feature)。默认值：False。
- **norm_first** (bool) - 如果为 True，则 LayerNorm 层位于 MultiheadAttention 层和 FeedForward 层之前，反之，位于其后。默认值：False。
- **dtype** (`mindspore.dtype`) - Parameter 的数据类型。默认值：mstype.float32。

输入：

- **src** (Tensor) - 源序列。如果源序列没有 batch，shape 是 (S, E)；否则如果 `batch_first=False`，则 shape 为 (S, N, E)，如果 `batch_first=True`，则 shape 为 (N, S, E)。(S) 是源序列的长度，(N) 是 batch 个数，(E) 是特性个数。数据类型：float16、float32 或者 float64。
- **tgt** (Tensor) - 目标序列。如果目标序列没有 batch，shape 是 (T, E)；否则如果 `batch_first=False`，则 shape 为 (T, N, E)，如果 `batch_first=True`，则 shape 为 (N, T, E)。(T) 是目标序列的长度。数据类型：float16、float32 或者 float64。
- **src_mask** (Tensor, 可选) - 源序列的掩码矩阵。shape 是 (S, S) 或 (N * nhead, S, S)。数据类型：float16、float32、float64 或者 bool。默认值：None。
- **tgt_mask** (Tensor, 可选) - 目标序列的掩码矩阵。shape 是 (T, T) 或 (N * nhead, T, T)。数据类型：float16、float32、float64 或者 bool。默认值：None。
- **memory_mask** (Tensor, 可选) - memory 序列的掩码矩阵。shape 是 (T, S)。数据类型：float16、float32、float64 或者 bool。默认值：None。
- **src_key_padding_mask** (Tensor, 可选) - 源序列 Key 矩阵的掩码矩阵。如果目标序列没有 batch，shape 是 (S)，否则 shape 为 (N, S)。数据类型：float16、float32、float64 或者 bool。默认值：None。
- **tgt_key_padding_mask** (Tensor, 可选) - 目标序列 Key 矩阵的掩码矩阵。如果目标序列没有 batch，shape 是 (T)，否则 shape 为 (N, S)。数据类型：float16、float32、float64 或者 bool。默认值：None。
- **memory_key_padding_mask** (Tensor, 可选) - memory 序列 Key 矩阵的掩码矩阵。如果目标序列没有 batch，shape 是 (S)，否则 shape 为 (N, S)。数据类型：float16、float32、float64 或者 bool。默认值：None。

输出：

Tensor。如果源序列没有 batch，shape 是 (T, E)；否则如果 `batch_first=False`，则 shape 为 (T, N, E)，如果 `batch_first=True`，则 shape 为 (N, T, E)。

异常：

- **ValueError** - 如果 `src` 和 `tgt` 的 batch size 不相等。
- **ValueError** - 如果 `src` 和 `tgt` 的特征数量必须和 `d_model` 的一致。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> transformer_model = ms.nn.Transformer(nhead=16, num_encoder_layers=12)
>>> src = ms.Tensor(np.random.rand(10, 32, 512), ms.float32)
>>> tgt = ms.Tensor(np.random.rand(20, 32, 512), ms.float32)
>>> out = transformer_model(src, tgt)
>>> print(out.shape)
(20, 32, 512)
```

3.7 嵌入层

接口名	概述	支持平台
<code>mindspore.nn.Embedding</code>	嵌入层。	Ascend GPU CPU
<code>mindspore.nn.EmbeddingLookup</code>	嵌入查找层。	Ascend GPU CPU
<code>mindspore.nn.MultiFieldEmbeddingLookup</code>	根据指定的索引和字段 ID，返回输入 Tensor 的切片。	Ascend GPU

3.7.1 mindspore.nn.Embedding

`class mindspore.nn.Embedding (vocab_size, embedding_size, use_one_hot=False, embedding_table='normal', dtype=mstype.float32, padding_idx=None)`

嵌入层。

用于存储词向量并使用索引进行检索，根据输入 Tensor 中的 id，从 `embedding_table` 中查询对应的 `embedding` 向量。当输入为 id 组成的序列时，输出为对应 `embedding` 向量构成的矩阵。

说明：当 `use_one_hot` 等于 `True` 时，`x` 的类型必须是 `mindspore.int32`。

参数：

- `vocab_size` (int) - 词典的大小。
- `embedding_size` (int) - 每个嵌入向量的大小。
- `use_one_hot` (bool) - 指定是否使用 one-hot 形式。默认值： `False` 。
- `embedding_table` (Union[`Tensor`, `str`, `Initializer`, `numbers.Number`]) - `embedding_table` 的初始化方法。当指定为字符串，字符串取值请参见类 `mindspore.common.initializer` 。默认值： `"normal"` 。
- `dtype` (`mindspore.dtype`) - `x` 的数据类型。默认值： `mstype.float32` 。
- `padding_idx` (int, `None`) - 将 `padding_idx` 对应索引所输出的嵌入向量用零填充。默认值： `None` 。该功能已停用。

输入：

- `x` (`Tensor`) - `Tensor` 的 `shape` 为 `(batch_size, x_length)` ， 其元素为整数，并且元素数目必须小于等于 `vocab_size` ， 否则相应的嵌入向量将为零。该数据类型可以是 `int32` 或 `int64`。

输出：

`Tensor` 的 `shape` (`batch_size, x_length, embedding_size`) 。

异常:

- **TypeError** - 如果 `vocab_size` 或者 `embedding_size` 不是整数。
- **TypeError** - 如果 `use_one_hot` 不是布尔值。
- **ValueError** - 如果 `padding_idx` 是一个不在 `[0, vocab_size]` 范围内的整数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> net = nn.Embedding(20000, 768, True)
>>> x = Tensor(np.ones([8, 128]), mindspore.int32)
>>> # Maps the input word IDs to word embedding.
>>> output = net(x)
>>> result = output.shape
>>> print(result)
(8, 128, 768)
```

3.7.2 mindspore.nn.EmbeddingLookup

class mindspore.nn.EmbeddingLookup (*vocab_size, embedding_size, param_init='normal', target='CPU', slice_mode='batch_slice', manual_shapes=None, max_norm=None, sparse=True, vocab_cache_size=0, dtype=mstype.float32*)

嵌入查找层。

与嵌入层功能相同，主要用于自动并行或半自动并行时，存在大规模嵌入层的异构并行场景。

说明：当 `target` 设置为 `'CPU'` 时，此模块将使用 `ops.EmbeddingLookup().set_device('CPU')`，在 `lookup` 表指定了 `'offset = 0'`。当 `target` 设置为 `'DEVICE'` 时，此模块将使用 `ops.Gather()`，在 `lookup` 表指定了 `'axis = 0'`。在字段切片模式下，必须指定 `manual_shapes`。此 `tuple` 包含 `vocab[i]` 元素，`vocab[i]` 是第 `i` 部分的行号。该模块不支持 `PyNative` 模式。

参数:

- **vocab_size** (int) - 嵌入词典的大小。
- **embedding_size** (int) - 每个嵌入向量的大小。
- **param_init** (Union[`Tensor`, `str`, `Initializer`, `numbers.Number`]) - `embedding_table` 的初始化方法。当指定为字符串，字符串取值请参见类 *Initializer*。默认值: `'normal'`。
- **target** (str) - 指定执行操作的 `'target'`。取值范围为 `['DEVICE', 'CPU']`。默认值: `'CPU'`。
- **slice_mode** (str) - `semi_auto_parallel` 或 `auto_parallel` 模式下的切片方式。默认值: `'batch_slice'`。
 - **batch_slice** (str) - EmbeddingLookup 算子会将输入的索引张量按批次 (`batch`) 进行划分，然后查找对应的嵌入向量。适用于每个样本都有相同数量索引的情况。
 - **field_slice** (str) - EmbeddingLookup 算子会将输入的索引张量按特征 (`field`) 进行划分，然后查找对应的嵌入向量。适用于每个样本索引数量可能不同但是特征维度相同的情况。
 - **table_row_slice** (str) - EmbeddingLookup 算子会将输入的索引张量看作一个二维表，并按行进行划分，然后查找对应的嵌入向量。

- **table_column_slice** (str) - EmbeddingLookup 算子会将输入的索引张量看作一个二维表, 并按列进行划分, 然后查找对应的嵌入向量。
- **manual_shapes** (tuple) - 字段切片模式下的伴随数组 (accompaniment array), 默认值: None。
- **max_norm** (Union[float, None]) - 最大剪切值。数据类型必须为 float16、float32 或 None。默认值: None。
- **sparse** (bool) - 使用稀疏模式。当'target' 设置为'CPU' 时, 'sparse' 必须为 True。默认值: True。
- **vocab_cache_size** (int) - 嵌入字典的缓存大小。默认值: 0。仅在训练模式和'DEVICE' 目标中有效。相应优化器的力矩参数也将设置为缓存大小。此外需注意, 它还会消耗'DEVICE' 内存, 因此建议合理设置参数值, 避免内存不足。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: *mstype.float32*。

输入:

- **input_indices** (Tensor) - shape 为 $(y_1, y_2, \dots, y_S)$ 的 Tensor。指定原始 Tensor 元素的索引。当取值超出 embedding_table 的范围时, 超出部分在输出中填充为 0。不支持负值, 如果为负值, 则结果未定义。在 semi auto parallel 或 auto parallel 模式下运行时, Input_indices 只能是此接口中的二维 Tensor。

输出:

Tensor, shape 为 $(z_1, z_2, \dots, z_N)$ 的 Tensor。

异常:

- **TypeError** - vocab_size、embedding_size 或 vocab_cache_size 不是整数。
- **TypeError** - sparse 不是 bool 或 manual_shapes 不是 tuple。
- **ValueError** - vocab_size 或 embedding_size 小于 1。
- **ValueError** - vocab_cache_size 小于 0。
- **ValueError** - target 既不是'CPU' 也不是'DEVICE'。
- **ValueError** - slice_mode 不是'batch_slice'、'field_slice'、'table_row_slice' 或'table_column_slice'。
- **ValueError** - sparse 为 False 且 target 为'CPU'。
- **ValueError** - slice_mode 为'field_slice' 且 manual_shapes 是 None。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input_indices = Tensor(np.array([[1, 0], [3, 2]]), mindspore.int32)
>>> result = nn.EmbeddingLookup(4, 2)(input_indices)
>>> print(result.shape)
(2, 2, 2)
```

3.7.3 mindspore.nn.MultiFieldEmbeddingLookup

```
class mindspore.nn.MultiFieldEmbeddingLookup (vocab_size, embedding_size, field_size, param_init='normal',
                                              target='CPU', slice_mode='batch_slice', feature_num_list=None,
                                              max_norm=None, sparse=True, operator='SUM',
                                              dtype=mstype.float32)
```

根据指定的索引和字段 ID，返回输入 Tensor 的切片。此操作支持同时使用 multi hot 和 one hot 查找嵌入。

说明：当'target' 设置为 'CPU' 时，此模块将使用 P.EmbeddingLookup().set_device('CPU') 指定'offset = 0' 的查找表。

当'target' 设置为 'DEVICE' 时，此模块将使用 P.Gather() 指定'axis = 0' 的查找表。

具有相同 *field_ids* 的向量将由'operator' 组合，例如'SUM'、'MAX' 和'MEAN'。确保填充 ID 的 *input_values* 为零，以便忽略它们。如果字段绝对权重之和为零，最终输出将为零。该类仅支持 ['table_row_slice', 'batch_slice', 'table_column_slice']。对于 Ascend 设备上的'MAX' 操作，存在 $batch\_size * (seq\_length + field\_size) < 3500$ 的约束。

参数：

- **vocab_size** (int) - 嵌入词典的大小。
- **embedding_size** (int) - 每个嵌入向量的大小。
- **field_size** (int) - 最终输出的字段大小。
- **param_init** (Union[Tensor, str, Initializer, numbers.Number]) - 嵌入 Tensor 的初始化方法。当指定字符串时，请参见 *Initializer* 类了解字符串的值。默认值：'normal'。
- **target** (str) - 指定执行操作的'target'。该值必须在 ['DEVICE', 'CPU'] 中。默认值：'CPU'。
- **slice_mode** (str) - semi_auto_parallel 或 auto_parallel 模式下的切片方式。默认值：'batch_slice'。
 - **batch_slice** (str) - EmbeddingLookup 算子会将输入的索引张量按批次 (batch) 进行划分，然后查找对应的嵌入向量。适用于每个样本都有相同数量索引的情况。
 - **field_slice** (str) - EmbeddingLookup 算子会将输入的索引张量按特征 (field) 进行划分，然后查找对应的嵌入向量。适用于每个样本索引数量可能不同但是特征维度相同的情况。
 - **table_row_slice** (str) - EmbeddingLookup 算子会将输入的索引张量看作一个二维表，并按行进行划分，然后查找对应的嵌入向量。
 - **table_column_slice** (str) - EmbeddingLookup 算子会将输入的索引张量看作一个二维表，并按列进行划分，然后查找对应的嵌入向量。
- **feature_num_list** (tuple) - 字段切片模式下的伴随数组 (accompaniment array)。目前该参数的功能还未实现。默认值：None。
- **max_norm** (Union[float, None]) - 最大剪切值。数据类型必须为 float16、float32。默认值：None。
- **sparse** (bool) - 使用稀疏模式。当'target' 设置为'CPU' 时，'sparse' 必须为 true。默认值：True。
- **operator** (str) - 字段特征的池化方法。支持 'SUM'、'MEAN' 和 'MAX'。默认值：'SUM'。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **input_indices** (Tensor) - 指定输入 Tensor 元素的索引，其 shape 为 (batch_size, seq_length)。数据类型为 int32、int64。
- **input_values** (Tensor) - 指定 *input_indices* 元素的权重。将检索出的向量乘以 *input_values*。其 shape 为 (batch_size, seq_length)。类型为 float32。
- **field_ids** (Tensor) - 指定 *input_indices* 元素的字段 ID，其 shape 为 (batch_size, seq_length)。类型为 int32。

输出：

Tensor, shape 为 (batch_size, field_size, embedding_size) 。类型为 float32。

异常：

- **TypeError** - vocab_size 、 embedding_size 或 field_size 不是 int。
- **TypeError** - sparse 不是 bool 或 feature_num_list 不是 tuple。
- **ValueError** - vocab_size 、 embedding_size 或 field_size 小于 1。
- **ValueError** - target 既不是 'CPU' 也不是 'DEVICE'。
- **ValueError** - slice_mode 不是 'batch_slice'、'field_slice'、'table_row_slice' 或 'table_column_slice'。
- **ValueError** - sparse 为 False, target 为 'CPU'。
- **ValueError** - slice_mode 为 'field_slice', feature_num_list 为 None。
- **ValueError** - operator 不是 'SUM'、'MAX' 或 'MEAN'。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> input_indices = Tensor([[2, 4, 6, 0, 0], [1, 3, 5, 0, 0]], mindspore.int32)
>>> input_values = Tensor([[1, 1, 1, 0, 0], [1, 1, 1, 0, 0]], mindspore.float32)
>>> field_ids = Tensor([[0, 1, 1, 0, 0], [0, 0, 1, 0, 0]], mindspore.int32)
>>> net = nn.MultiFieldEmbeddingLookup(10, 2, field_size=2, operator='SUM', target='DEVICE')
>>> out = net(input_indices, input_values, field_ids)
>>> print(out.shape)
(2, 2, 2)
```

3.8 非线性激活函数层

接口名	概述	支持平台
<code>mindspore.nn.CELU</code>	CELU 激活层 (CELU Activation Operator)。	Ascend GPU CPU
<code>mindspore.nn.ELU</code>	指数线性单元激活函数 (Exponential Linear Unit activation function)。	Ascend GPU CPU
<code>mindspore.nn.FastGelu</code>	快速高斯误差线性单元激活函数 (Fast Gaussian Error Linear Units activation function)。	Ascend GPU CPU
<code>mindspore.nn.GELU</code>	高斯误差线性单元激活函数 (Gaussian error linear unit activation function)。	Ascend GPU CPU
<code>mindspore.nn.GLU</code>	门线性单元函数 (Gated Linear Unit function)。	Ascend GPU CPU
<code>mindspore.nn.get_activation</code>	获取激活函数。	Ascend GPU CPU
<code>mindspore.nn.Hardtanh</code>	逐元素计算 Hardtanh 函数。	Ascend GPU CPU
<code>mindspore.nn.HShrink</code>	逐元素计算 Hard Shrink 激活函数。	Ascend GPU CPU
<code>mindspore.nn.HSigmoid</code>	逐元素计算 Hard Sigmoid 激活函数。	Ascend GPU CPU
<code>mindspore.nn.HSwish</code>	逐元素计算 Hard Swish 激活函数。	Ascend GPU CPU
<code>mindspore.nn.LeakyReLU</code>	逐元素计算 Leaky ReLU 激活函数。	Ascend GPU CPU

下页继续

表 8 – 续上页

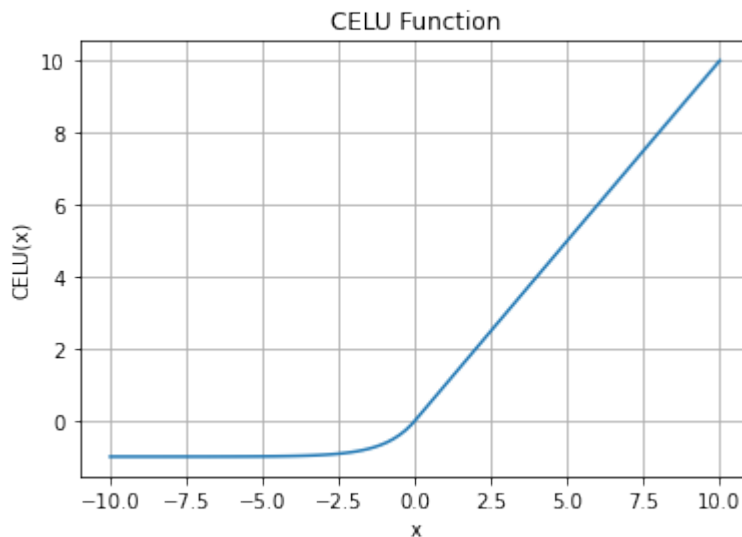
<code>mindspore.nn.LogSigmoid</code>	逐元素计算 Log Sigmoid 激活函数。	Ascend GPU CPU
<code>mindspore.nn.LogSoftmax</code>	按元素计算 Log Softmax 激活函数。	Ascend GPU CPU
<code>mindspore.nn.LRN</code>	局部响应归一化操作 LRN (Local Response Normalization)。	GPU CPU
<code>mindspore.nn.Mish</code>	逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。	Ascend GPU CPU
<code>mindspore.nn.Softsign</code>	逐元素计算 Softsign 激活函数。	Ascend GPU CPU
<code>mindspore.nn.PReLU</code>	逐元素计算 PReLU (PReLU Activation Operator) 激活函数。	Ascend GPU CPU
<code>mindspore.nn.ReLU</code>	逐元素计算 ReLU (Rectified Linear Unit activation function) 修正线性单元激活函数。	Ascend GPU CPU
<code>mindspore.nn.ReLU6</code>	逐元素计算 ReLU6 激活函数。	Ascend GPU CPU
<code>mindspore.nn.RReLU</code>	逐元素计算 RReLU (Randomized Leaky ReLU) 激活函数。	Ascend GPU CPU
<code>mindspore.nn.SeLU</code>	逐元素计算激活函数 SeLU (Scaled exponential Linear Unit)。	Ascend GPU CPU
<code>mindspore.nn.SiLU</code>	逐元素计算 SiLU 激活函数。	Ascend GPU CPU
<code>mindspore.nn.Sigmoid</code>	逐元素计算 Sigmoid 激活函数。	Ascend GPU CPU
<code>mindspore.nn.Softmin</code>	逐元素计算 Softmin 函数，它是二分类函数 <code>mindspore.nn.Sigmoid</code> 在多分类上的推广，目的是将多分类的结果以概率的形式展现出来。	Ascend GPU CPU
<code>mindspore.nn.Softmax</code>	逐元素计算 Softmax 激活函数，它是二分类函数 <code>mindspore.nn.Sigmoid</code> 在多分类上的推广，目的是将多分类的结果以概率的形式展现出来。	Ascend GPU CPU
<code>mindspore.nn.Softmax2d</code>	应用于 2D 特征数据的 Softmax 函数。	Ascend GPU CPU
<code>mindspore.nn.SoftShrink</code>	逐元素计算 SoftShrink 激活函数。	Ascend GPU CPU
<code>mindspore.nn.Tanh</code>	逐元素计算 Tanh (双曲正切值) 激活函数，返回一个新的 Tensor。	Ascend GPU CPU
<code>mindspore.nn.Tanhshrink</code>	逐元素计算 Tanhshrink 激活函数，返回一个新的 Tensor。	Ascend GPU CPU
<code>mindspore.nn.Threshold</code>	逐元素计算 Threshold 激活函数。	Ascend GPU CPU

3.8.1 mindspore.nn.CELU

class `mindspore.nn.CELU` (*alpha=1.0*)
CELU 激活层 (CELU Activation Operator)。
根据 Continuously Differentiable Exponential Linear Units 对输入 Tensor 逐元素计算。

$$\text{CELU}(x) = \max(0, x) + \min(0, \alpha * (\exp(x/\alpha) - 1))$$

更多详情，请查看：[CELU](#)。
CELU 函数图：



参数:

- **alpha** (float, 可选) - CELU 公式中的 α 值。默认值: 1.0。

输入:

- **x** (Tensor) - CELU 的输入。其数据类型为 float16 或 float32, shape 为 $(N,*)$, 其中 $*$ 表示任何数量的附加维度。

输出:

Tensor, 数据类型和 shape 与 x 相同。

异常:

- **TypeError** - 如果 α 不是 float。
- **ValueError** - 如果 α 的值为 0。
- **TypeError** - 如果输入 x 不是 Tensor。
- **TypeError** - 如果输入 x 的数据类型既不是 float16 也不是 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([-2.0, -1.0, 1.0, 2.0]), mindspore.float32)
>>> celu = nn.CELU()
>>> output = celu(x)
>>> print(output)
[-0.86466473 -0.63212055  1.          2.          ]
```

3.8.2 mindspore.nn.ELU

class mindspore.nn.ELU(alpha=1.0)

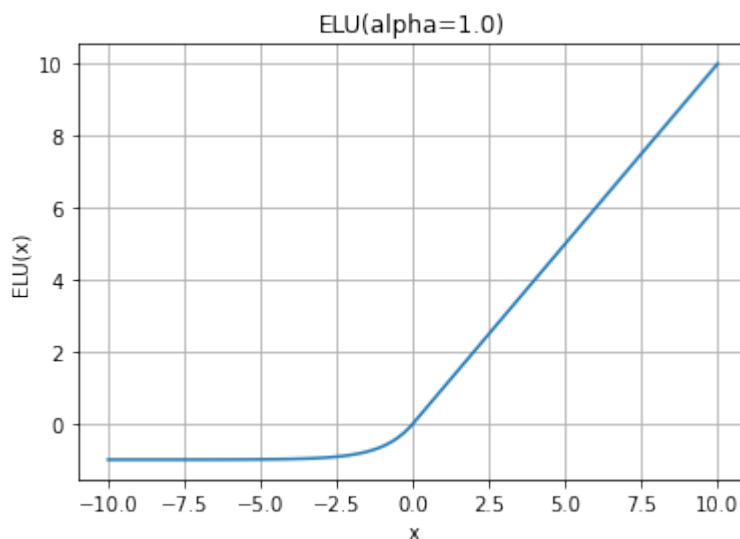
指数线性单元激活函数 (Exponential Linear Unit activation function)。

对输入的每个元素计算 ELU。该激活函数定义如下：

$$E_i = \begin{cases} x_i, & \text{if } x_i \geq 0; \\ \alpha * (\exp(x_i) - 1), & \text{otherwise.} \end{cases}$$

其中, x_i 表示输入的元素, α 表示 *alpha* 参数。

ELU 函数图：



参数：

- **alpha** (*float*) - ELU 的 *alpha* 值, 数据类型为浮点数, 目前仅支持 1.0。默认值: 1.0。

输入：

- **input_x** (*Tensor*) - 用于计算 ELU 的任意维度的 *Tensor*, 数据类型为 float16 或 float32。

输出：

Tensor, 数据类型和 shape 与 *input_x* 相同。

异常：

- **TypeError** - *alpha* 不是浮点数。
- **TypeError** - *input_x* 的数据类型既不是 float16, 也不是 float32。
- **ValueError** - *alpha* 不等于 1.0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
```

(下页继续)

(续上页)

```
>>> x = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float32)
>>> elu = nn.ELU()
>>> result = elu(x)
>>> print(result)
[-0.63212055 -0.86466473  0.  2.  1.]
```

3.8.3 mindspore.nn.FastGelu

class mindspore.nn.FastGelu

快速高斯误差线性单元激活函数 (Fast Gaussian Error Linear Units activation function)。

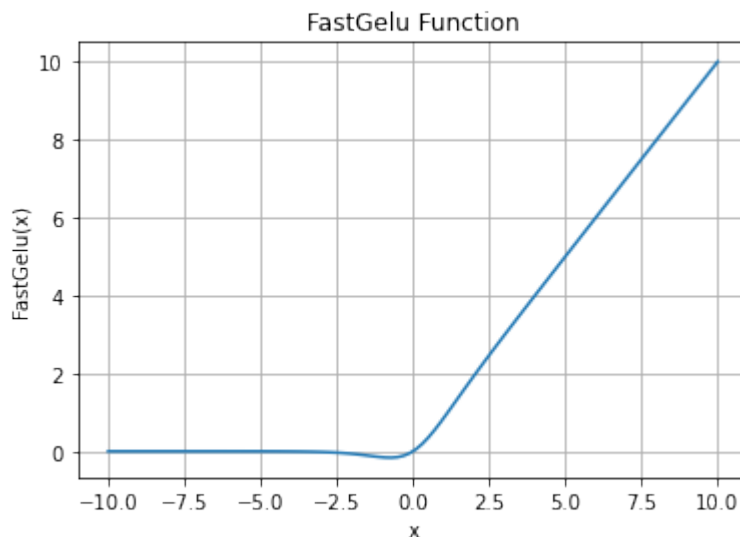
对输入的每个元素计算 FastGelu。输入是具有任何有效形状的张量。

FastGelu 定义如下：

$$\text{FastGelu}(x_i) = \frac{x_i}{1 + \exp(-1.702 * |x_i|)} * \exp(0.851 * (x_i - |x_i|))$$

其中 x_i 是输入的元素。

FastGelu 函数图：



输入：

- **x** (Tensor) - 用于计算 FastGelu 的 Tensor。数据类型为 float16 或 float32。shape 为 (N,*), 其中 * 表示任意的附加维度数。

输出：

Tensor, 具有与 x 相同的数据类型和 shape。

异常：

- **TypeError** - x 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：


```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]], mindspore.float32)
>>> fast_gelu = nn.FastGelu()
>>> output = fast_gelu(x)
>>> print(output)
[[-1.5418735e-01  3.9921875e+00 -9.7473649e-06]
 [ 1.9375000e+00 -1.0052517e-03  8.9824219e+00]]
```

3.8.4 mindspore.nn.GELU

class mindspore.nn.GELU (*approximate=True*)

高斯误差线性单元激活函数 (Gaussian error linear unit activation function)。

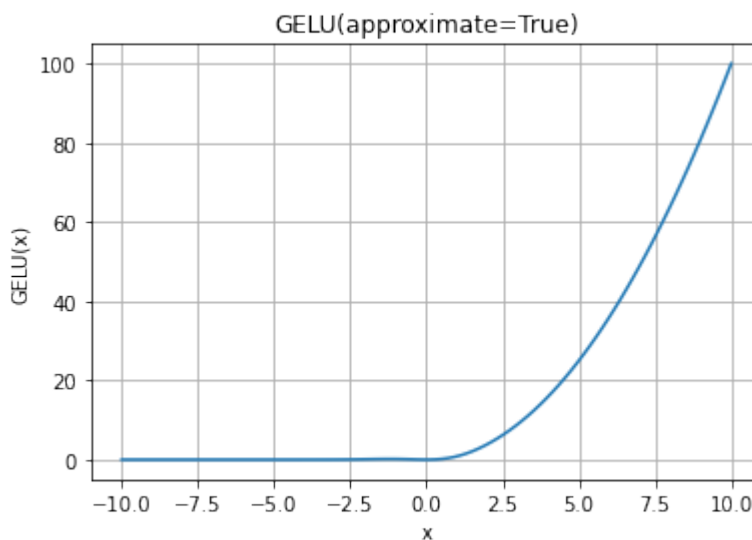
对输入的每个元素计算 GELU，输入可以是任意有效 shape 的 Tensor。

GELU 的定义如下：

$$GELU(x_i) = x_i * P(X < x_i),$$

其中 P 是标准高斯分布的累积分布函数， x_i 是输入的元素。

GELU 函数图：



参数：

- **approximate** (bool, 可选) - 是否启用 approximation。默认值：True。如果 approximate 的值为 True，则高斯误差线性激活函数为：

$$0.5 * x * (1 + \tanh(\sqrt{2/\pi} * (x + 0.044715 * x^3)))。$$

否则为： $x * P(X \leq x) = 0.5 * x * (1 + \operatorname{erf}(x/\sqrt{2}))$ ，其中 $P(X) \sim N(0, 1)$ 。

说明：

- 在计算 gelu 的输入梯度时，当输入为 inf，Ascend 与 GPU 在反向传播输出之间存在差异。
- 当输入 x 为 -inf 时，Ascend 的计算结果为 0，GPU 的计算结果为 nan。

- 当输入 x 为 inf 时, Ascend 的计算结果为梯度 dy , GPU 的计算结果为 nan 。
- 数学意义上, Ascend 的计算结果精度更高。

输入:

- x (Tensor) - 用于计算 GELU 的 Tensor。数据类型为 float16、float32 或 float64。shape 是 $(N, *)$, $*$ 表示任意的附加维度数。

输出:

Tensor, 具有与 x 相同的数据类型和 shape。

异常:

- **TypeError** - x 的数据类型不是 float16、float32 或 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> gelu = nn.GELU()
>>> output = gelu(x)
>>> print(output)
[[-1.5880802e-01  3.9999299e+00 -3.1077917e-21]
 [ 1.9545976e+00 -2.2918017e-07  9.0000000e+00]]
>>> gelu = nn.GELU(approximate=False)
>>> # CPU not support "approximate=False", using "approximate=True" instead
>>> output = gelu(x)
>>> print(output)
[[-1.5865526e-01  3.9998732e+00 -0.0000000e+00]
 [ 1.9544997e+00 -1.4901161e-06  9.0000000e+00]]
```

3.8.5 mindspore.nn.GLU

class mindspore.nn.GLU(*axis=-1*)

门线性单元函数 (Gated Linear Unit function)。

$$GLU(a, b) = a \otimes \sigma(b)$$

其中, a 表示输入 Tensor 的前一半元素, b 表示输入 Tensor 的另一半元素。这里 σ 为 sigmoid 函数, $\otimes$ 是 Hadamard 乘积。

参数:

- **axis** (int, 可选) - 指定分割轴。数据类型为整型, 默认值: -1 , 输入 x 的最后一维。

输入:

- x (Tensor) - Tensor 的 shape 为 $(*_1, N, *_2)$ 。* 表示任意数量的维度。

输出:

Tensor, 数据类型与输入 x 相同, shape 为 $(*_1, M, *_2)$, 其中 $M = N/2$ 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> m = ms.nn.GLU()
>>> input = ms.Tensor([[0.1, 0.2, 0.3, 0.4], [0.5, 0.6, 0.7, 0.8]])
>>> output = m(input)
>>> print(output)
[[0.05744425 0.11973753]
 [0.33409387 0.41398472]]
```

3.8.6 mindspore.nn.get_activation

`mindspore.nn.get_activation(name, prim_name=None)`

获取激活函数。

参数:

- **name** (str) - 激活函数名称。
- **prim_name** (Union[str, None]) - 算子名称, 默认值: None。

返回:

激活函数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.nn as nn
>>> sigmoid = nn.get_activation('sigmoid')
>>> print(sigmoid)
Sigmoid()
```

3.8.7 mindspore.nn.Hardtanh

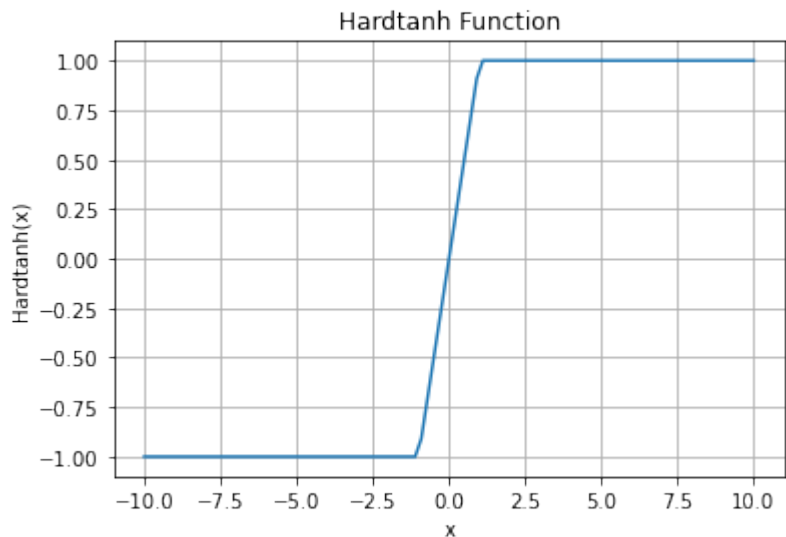
`class mindspore.nn.Hardtanh(min_val=-1.0, max_val=1.0)`

逐元素计算 Hardtanh 函数。Hardtanh 函数定义为:

$$\text{Hardtanh}(x) = \begin{cases} 1, & \text{if } x > 1; \\ -1, & \text{if } x < -1; \\ x, & \text{otherwise.} \end{cases}$$

线性区域范围 $[-1, 1]$ 可以使用 `min_val` 和 `max_val` 进行调整。

Hardtanh 函数图:



说明：在 Ascend 硬件上，float16 数据类型场景下会有偶现的精度误差较大的问题。

参数：

- **min_val** (Union[int, float]) - 线性区域范围的最小值。默认值：-1.0。
- **max_val** (Union[int, float]) - 线性区域范围的最大值。默认值：1.0。

输入：

- **x** (Tensor) - 数据类型为 float16 或 float32 的 Tensor。在 CPU 和 Ascend 平台上支持零到七维。在 GPU 平台上支持零到四维。

输出：

Tensor，数据类型和 shape 与 x 的相同。

异常：

- **TypeError** - x 不是 Tensor。
- **TypeError** - x 的数据类型既不是 float16 也不是 float32。
- **TypeError** - min_val 的数据类型既不是 int 也不是 float。
- **TypeError** - max_val 的数据类型既不是 int 也不是 float。
- **ValueError** - min_val 不小于 max_val 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> hardtanh = nn.Hardtanh(min_val=-1.0, max_val=1.0)
>>> output = hardtanh(x)
```

(下页继续)

(续上页)

```
>>> print(output)
[-1. -1.  0.  1.  1.]
```

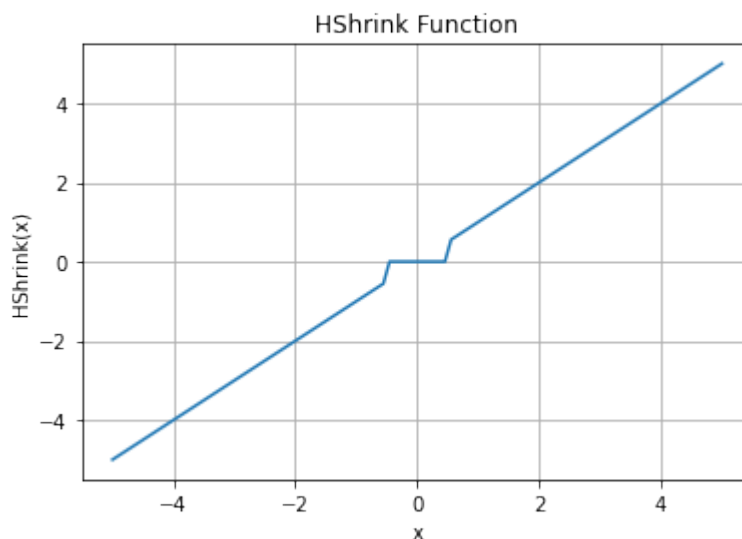
3.8.8 mindspore.nn.HShrink

class mindspore.nn.HShrink (lambd=0.5)

逐元素计算 Hard Shrink 激活函数。公式定义如下：

$$\text{HShrink}(x) = \begin{cases} x, & \text{if } x > \lambda \\ x, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

HShrink 函数图：



参数：

- **lambd** (number, 可选) - Hard Shrink 公式定义的阈值 λ 。默认值：0.5。

输入：

- **input** (Tensor) - Hard Shrink 的输入。支持数据类型：
 - Ascend: float16、float32、bfloat16。
 - CPU/GPU: float16、float32。

输出：

Tensor, shape 和数据类型与输入相同。

异常：

- **TypeError** - *lambd* 不是 float、int 或 bool。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的 dtype 不是 float16、float32 或 bfloat16。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.array([[0.5, 1, 2.0], [0.0533, 0.0776, -2.1233]]), mindspore.float32)
>>> hshrink = nn.HShrink()
>>> output = hshrink(input)
>>> print(output)
[[ 0.      1.      2.      ]
 [ 0.      0.     -2.1233]]
```

3.8.9 mindspore.nn.HSigmoid

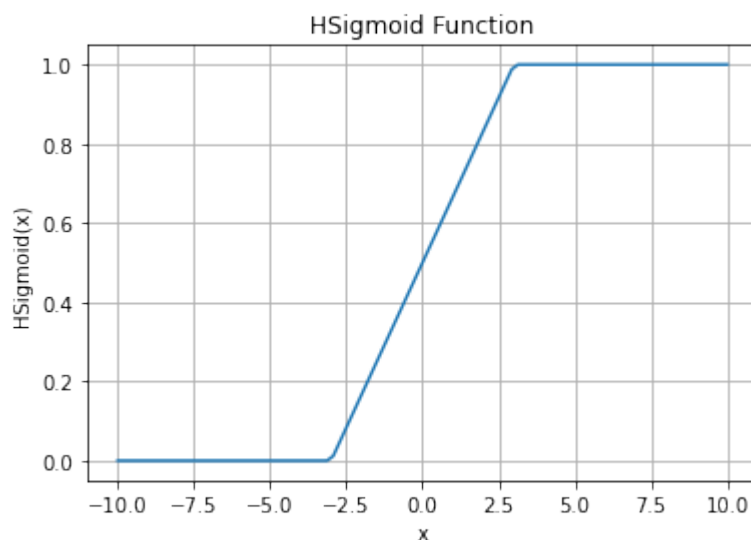
class mindspore.nn.HSigmoid

逐元素计算 Hard Sigmoid 激活函数。

Hard Sigmoid 定义为:

$$\text{HSigmoid}(input) = \begin{cases} 0, & \text{if } input \leq -3, \\ 1, & \text{if } input \geq +3, \\ input/6 + 1/2, & \text{otherwise} \end{cases}$$

HSigmoid 函数图:



输入:

- **input** (Tensor) - Hard Sigmoid 的输入。

输出:

Tensor, 数据类型和 shape 与 *input* 的相同。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> hsigmoid = nn.HSigmoid()
>>> result = hsigmoid(input)
>>> print(result)
[0.3333 0.1666 0.5      0.8335 0.6665]
```

3.8.10 mindspore.nn.HSwish

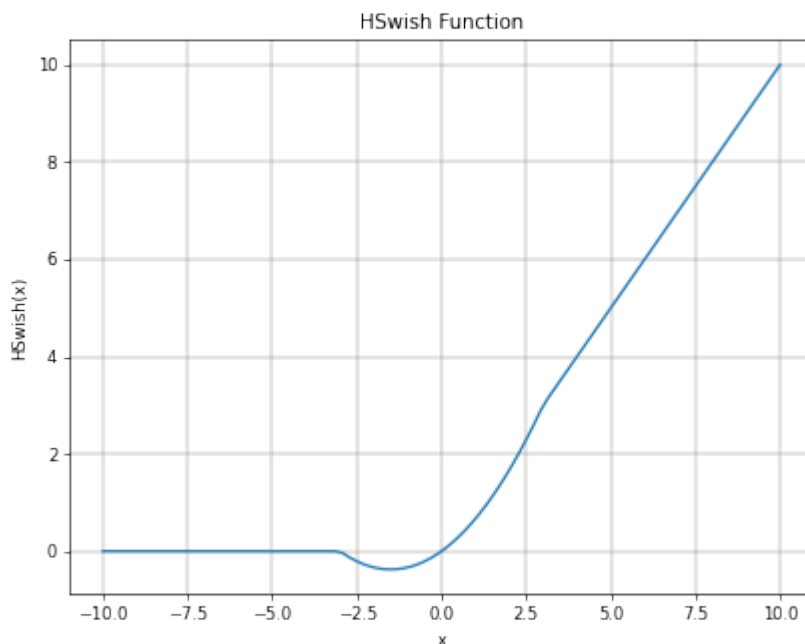
class mindspore.nn.HSwish

逐元素计算 Hard Swish 激活函数。

Hard Swish 定义如下:

$$\text{HSwish}(\text{input}) = \begin{cases} 0, & \text{if } \text{input} \leq -3, \\ \text{input}, & \text{if } \text{input} \geq +3, \\ \text{input} * (\text{input} + 3)/6, & \text{otherwise} \end{cases}$$

HSwish 函数图:



输入:

- **input** (Tensor) - Hard Swish 的输入。

输出:

Tensor, 具有与 *input* 相同的数据类型和 shape。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> hswish = nn.HSwish()
>>> result = hswish(input)
>>> print(result)
[-0.3333 -0.3333  0.      1.667  0.6665]
```

3.8.11 mindspore.nn.LeakyReLU

class mindspore.nn.LeakyReLU(*alpha=0.2*)

逐元素计算 Leaky ReLU 激活函数。

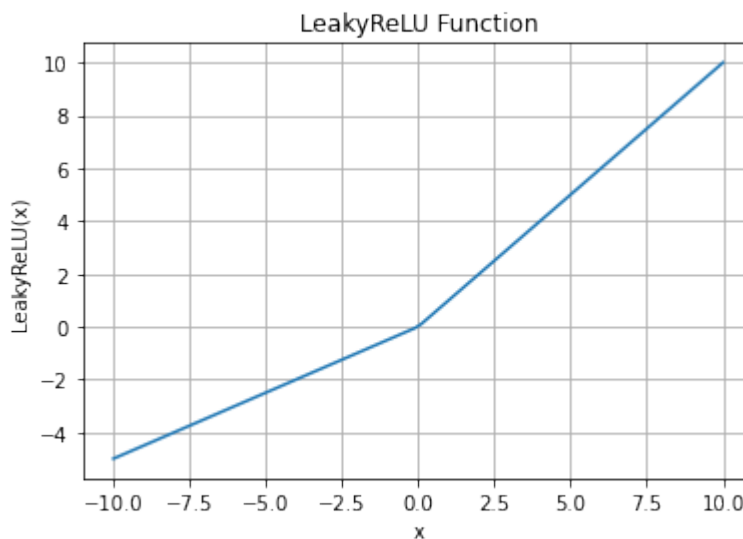
该激活函数定义如下:

$$\text{leaky\_relu}(x) = \begin{cases} x, & \text{if } x \geq 0; \\ \alpha * x, & \text{otherwise.} \end{cases}$$

其中, α 表示 *alpha* 参数。

更多细节详见 [Rectifier Nonlinearities Improve Neural Network Acoustic Models](#)。

LeakyReLU 函数图:



参数:

- **alpha** (*Union[int, float]*) - x 小于 0 时激活函数的斜率, 默认值: 0.2。

输入:

- **x** (*Tensor*) - 计算 LeakyReLU 的任意维度的 Tensor。

输出:

Tensor, 数据类型和 shape 与 x 相同。

异常:

- **TypeError** - α 不是浮点数或整数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> leaky_relu = nn.LeakyReLU()
>>> output = leaky_relu(x)
>>> print(output)
[[-0.2  4.  -1.6]
 [ 2.  -1.   9. ]]
```

3.8.12 mindspore.nn.LogSigmoid

class mindspore.nn.LogSigmoid

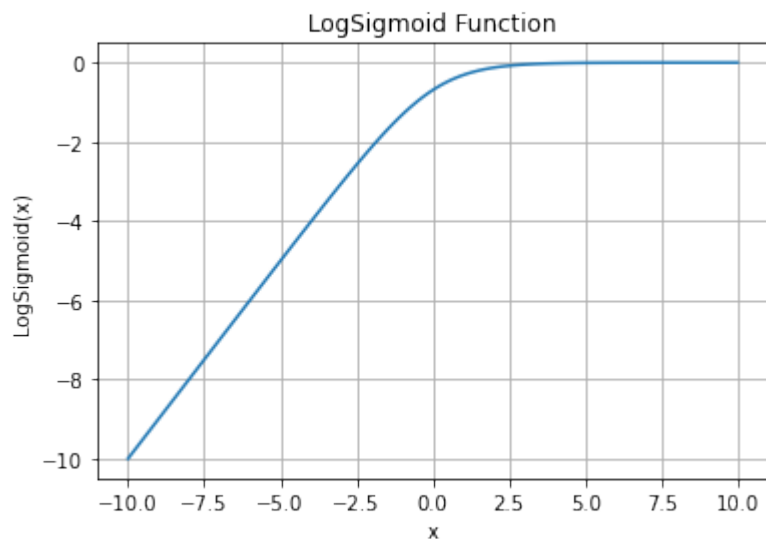
逐元素计算 Log Sigmoid 激活函数。输入是任意格式的 Tensor。

Log Sigmoid 定义为:

$$\text{logsigmoid}(x_i) = \log\left(\frac{1}{1 + \exp(-x_i)}\right),$$

其中, x_i 是输入 Tensor 的一个元素。

LogSigmoid 函数图:



输入：

- **x** (Tensor) - Log Sigmoid 的输入，数据类型为 float16 或 float32。shape 为 $(N, *)$ ，其中 * 表示任意的附加维度。

输出：

Tensor，数据类型和 shape 与 x 的相同。

异常：

- **TypeError** - x 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> net = nn.LogSigmoid()
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> output = net(x)
>>> print(output)
[-0.31326166 -0.12692806 -0.04858734]
```

3.8.13 mindspore.nn.LogSoftmax

class mindspore.nn.LogSoftmax (axis=-1)

按元素计算 Log Softmax 激活函数。

输入经 Softmax 函数、Log 函数转换后，值的范围在 $[-\inf, 0)$ 。

Log Softmax 定义如下：

$$\text{logsoftmax}(x_i) = \log \left(\frac{\exp(x_i)}{\sum_{j=0}^{n-1} \exp(x_j)} \right)$$

参数：

- **axis** (int) - Log Softmax 运算的 axis, -1 表示最后一个维度。默认值: -1。

输入:

- **x** (Tensor) - Log Softmax 的输入, 数据类型为 float16 或 float32。

输出:

Tensor, 数据类型和 shape 与 *x* 相同, 输出值的范围在 $[-\infty, 0)$ 。

异常:

- **TypeError** - *axis* 不是 int。
- **TypeError** - *x* 的数据类型既不是 float16 也不是 float32。
- **ValueError** - *axis* 不在 $[-\text{len}(x), \text{len}(x))$ 范围中。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]], mindspore.float32)
>>> log_softmax = nn.LogSoftmax()
>>> output = log_softmax(x)
>>> print(output)
[[-5.00672150e+00 -6.72150636e-03 -1.20067215e+01]
 [-7.00091219e+00 -1.40009127e+01 -9.12250078e-04]]
```

3.8.14 mindspore.nn.LRN

class mindspore.nn.LRN(*depth_radius=5, bias=1.0, alpha=1.0, beta=0.5, norm_region='ACROSS_CHANNELS'*)
局部响应归一化操作 LRN(Local Response Normalization)。

警告: LRN 在 Ascend 平台已废弃, 存在潜在精度问题。建议使用其他归一化方法, 如 `mindspore.nn.BatchNorm1d`、`mindspore.nn.BatchNorm2d`、`mindspore.nn.BatchNorm3d` 代替 LRN。

更多参考详见 `mindspore.ops.lrn()`。

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input_x = Tensor(np.array([[[[0.1], [0.2]],
...                             [[0.3], [0.4]]]]], mindspore.float32)
>>> output = nn.LRN()(input_x)
>>> print(output)
[[[0.09534626]
 [0.1825742 ]]]
```

(下页继续)

(续上页)

```
[[0.2860388 ]
 [0.3651484 ]]]
```

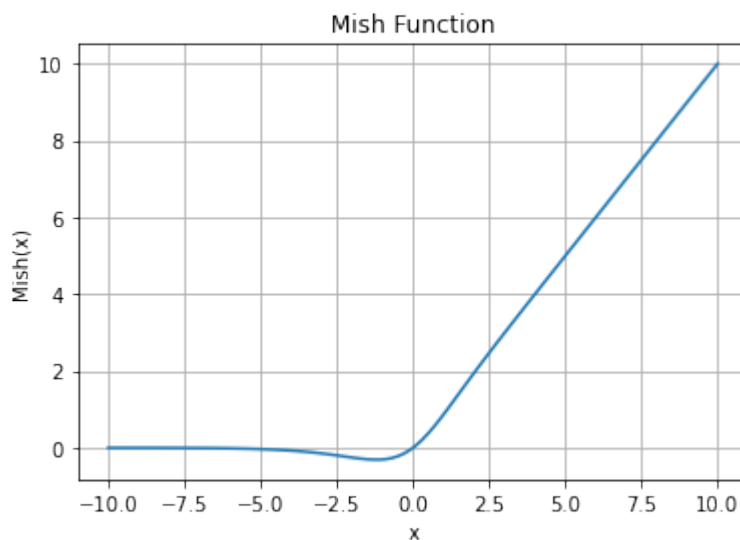
3.8.15 mindspore.nn.Mish

class mindspore.nn.Mish

逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。

更多参考详见 `mindspore.ops.mish()`。

Mish 激活函数图：



支持平台：

Ascend GPU CPU

样例：

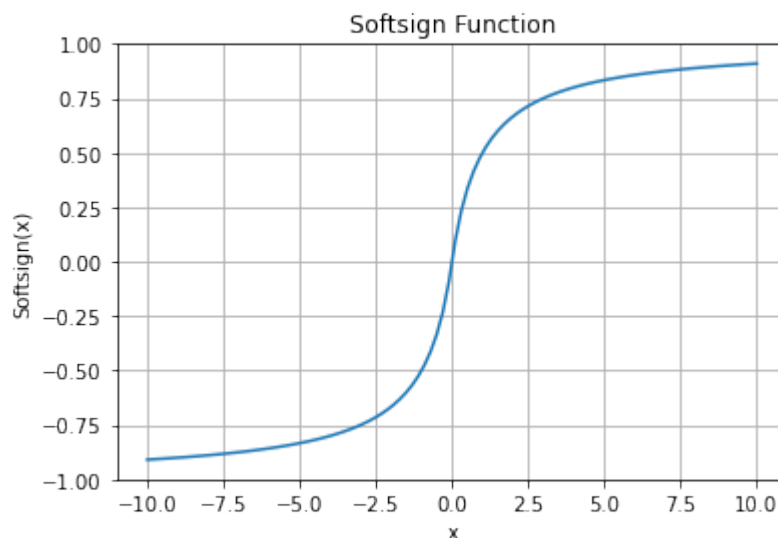
```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]), mindspore.float32)
>>> mish = nn.Mish()
>>> output = mish(x)
>>> print(output)
[[-3.03401530e-01  3.99741292e+00 -2.68321624e-03]
 [ 1.94395900e+00 -3.35762873e-02  9.00000000e+00]]
```

3.8.16 mindspore.nn.Softsign

class mindspore.nn.Softsign

逐元素计算 Softsign 激活函数。

Softsign 函数图：



更多参考详见 `mindspore.ops.softsign()`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([0, -1, 2, 30, -30]), mindspore.float32)
>>> softsign = nn.Softsign()
>>> output = softsign(x)
>>> print(output)
[ 0.          -0.5          0.6666667   0.9677419  -0.9677419]
```

3.8.17 mindspore.nn.PReLU

class mindspore.nn.PReLU (*channel=1, w=0.25*)

逐元素计算 PReLU (PReLU Activation Operator) 激活函数。

公式定义为：

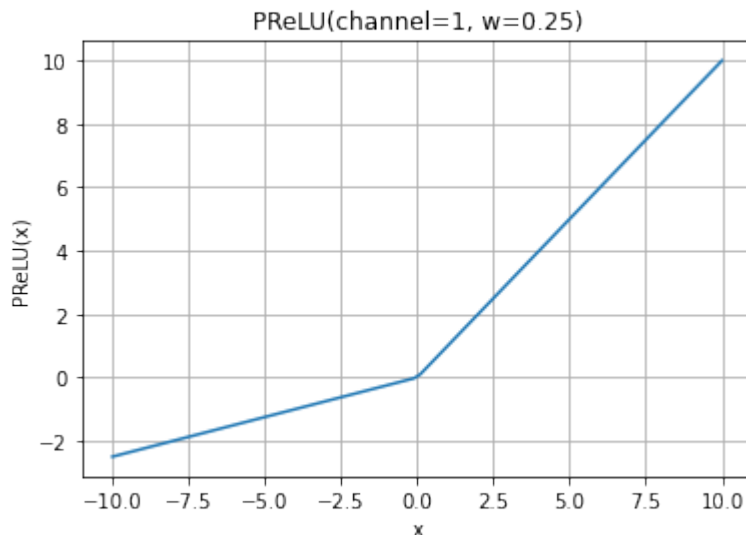
$$PReLU(x_i) = \max(0, x_i) + w * \min(0, x_i),$$

其中 x_i 是输入的 Tensor。

这里 w 是一个可学习的参数，默认初始值 0.25。

当带参数调用时每个通道上学习一个 w 。如果不带参数调用时，则将在所有通道中共享单个参数 w 。

PReLU 函数图：



参数:

- **channel** (int, 可选) - 可训练参数 w 的数量。它可以是 int, 值是 1, 或输入 Tensor x 的通道数。默认值: 1。
- **w** (Union[float, list, Tensor], 可选) - 参数的初始值。它可以是 float, float 组成的 list, 或与输入 Tensor x 具有相同数据类型的 Tensor。默认值: 0.25。

输入:

- **x** (Tensor) - PReLU 的输入 Tensor, 其 shape 为 $(N, *)$, 其中 $*$ 表示任意的额外维度, 数据类型为 float16 或 float32。

输出:

Tensor, 数据类型和 shape 与 x 相同。

异常:

- **TypeError** - *channel* 不是 int。
- **TypeError** - w 既不是 float, 也不是 list[float] 或 Tensor[float]。
- **TypeError** - x 的数据类型既不是 float16 也不是 float32。
- **ValueError** - x 是 Ascend 上的 0-D 或 1-D Tensor。
- **ValueError** - *channel* 小于 1。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[[[0.1, 0.6], [0.9, 0.9]]]]), mindspore.float32)
>>> prelu = nn.PReLU()
>>> output = prelu(x)
>>> print(output)
[[[0.1 0.6]
 [0.9 0.9]]]
```

3.8.18 mindspore.nn.ReLU

class mindspore.nn.ReLU

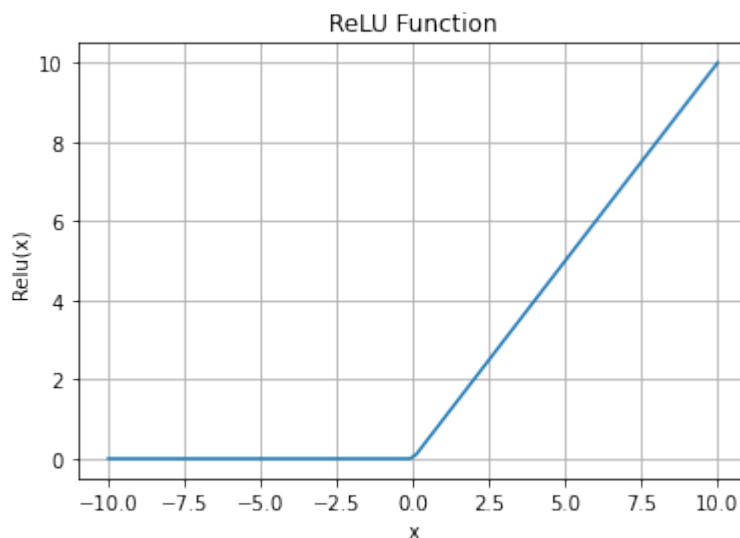
逐元素计算 ReLU (Rectified Linear Unit activation function) 修正线性单元激活函数。

$$\text{ReLU}(\text{input}) = (\text{input})^+ = \max(0, \text{input}),$$

逐元素求 $\max(0, \text{input})$ 。

说明：负数输出值会被修改为 0，正数输出不受影响。

ReLU 激活函数图：



输入：

- **input** (Tensor) - 用于计算 ReLU 的任意维度的 Tensor。

输出：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *input* 的数据类型不支持。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> input = Tensor(np.array([-1, 2, -3, 2, -1]), mindspore.float16)
>>> relu = nn.ReLU()
>>> output = relu(input)
>>> print(output)
[0. 2. 0. 2. 0.]
```

3.8.19 mindspore.nn.ReLU6

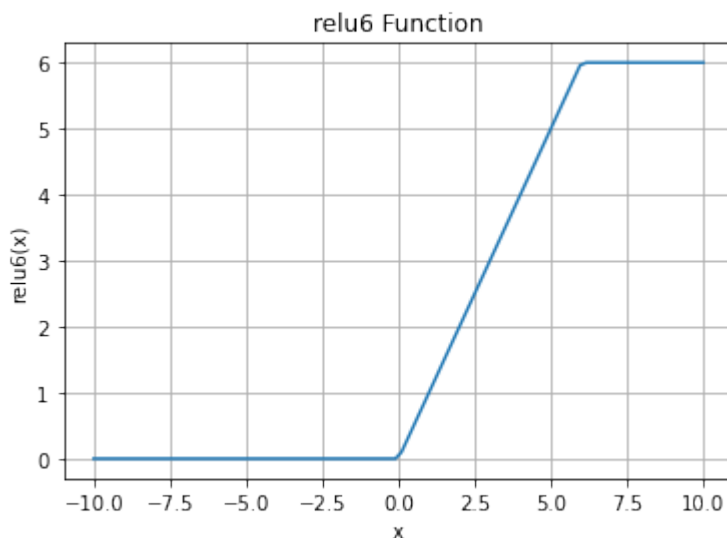
class mindspore.nn.ReLU6

逐元素计算 ReLU6 激活函数。

ReLU6 类似于 ReLU，不同之处在于设置了上限，其上限为 6，如果输入大于 6，输出会被限制为 6。公式如下：

$$Y = \min(\max(0, x), 6)$$

ReLU6 函数图：



输入：

- **x (Tensor)** - ReLU6 的输入，是具任何有效形状的张量，其数据类型为 float16 或 float32。

输出：

Tensor，数据类型与 x 相同。

异常：

- **TypeError** - x 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> relu6 = nn.ReLU6()
>>> output = relu6(x)
>>> print(output)
[0. 0. 0. 2. 1.]
```


3.8.20 mindspore.nn.RReLU

class mindspore.nn.RReLU(lower=1/8, upper=1/3)

逐元素计算 RReLU (Randomized Leaky ReLU) 激活函数。

该激活函数定义如下：

$$\text{RReLU}(x_{ji}) = \begin{cases} x_{ji}, & \text{if } x_{ji} \geq 0; \\ \alpha_{ji} * x_{ji}, & \text{otherwise.} \end{cases}$$

其中, $\alpha_{ji} \sim U(l, u), l \leq u$ 。

更多细节详见 [Empirical Evaluation of Rectified Activations in Convolution Network](#)。

参数：

- **lower** (Union[int, float]) - $x < 0$ 时激活函数的斜率的下界，默认值：1 / 8。
- **upper** (Union[int, float]) - $x < 0$ 时激活函数的斜率的上界，默认值：1 / 3。

输入：

- **x** (Tensor) - 计算 RReLU 的任意维度的 Tensor。

输出：

Tensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - lower 不是浮点数或整数。
- **TypeError** - upper 不是浮点数或整数。
- **TypeError** - x 不是 Tensor。
- **TypeError** - x 内的数据类型不是 mindspore.float16 或 mindspore.float32。
- **ValueError** - lower 大于 upper。

支持平台：

Ascend GPU CPU

样例：

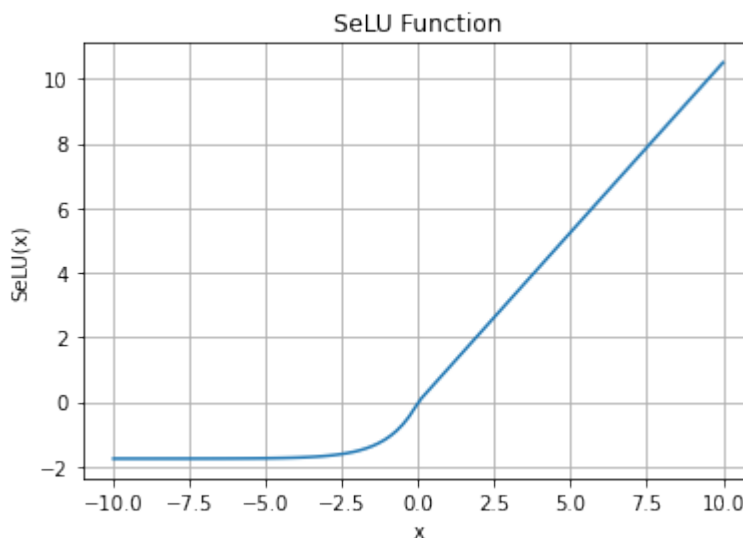
```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[ -1.0,  4.0], [ 2.0,  0.]]), mindspore.float32)
>>> r_relu = nn.RReLU()
>>> output = r_relu(x)
>>> print(output)
[[-0.31465699  4.          ]
 [ 2.          0.          ]]
```

3.8.21 mindspore.nn.SeLU

class mindspore.nn.SeLU

逐元素计算激活函数 SeLU (Scaled exponential Linear Unit)。

SeLU 函数图:



更多参考详见 `mindspore.ops.selu()`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input_x = Tensor(np.array([[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]), mindspore.float32)
>>> selu = nn.SeLU()
>>> output = selu(input_x)
>>> print(output)
[[-1.1113307  4.202804 -1.7575096]
 [ 2.101402 -1.7462534  9.456309 ]]
```

3.8.22 mindspore.nn.SiLU

class mindspore.nn.SiLU

逐元素计算 SiLU 激活函数。有时也被称作 Swish 函数。

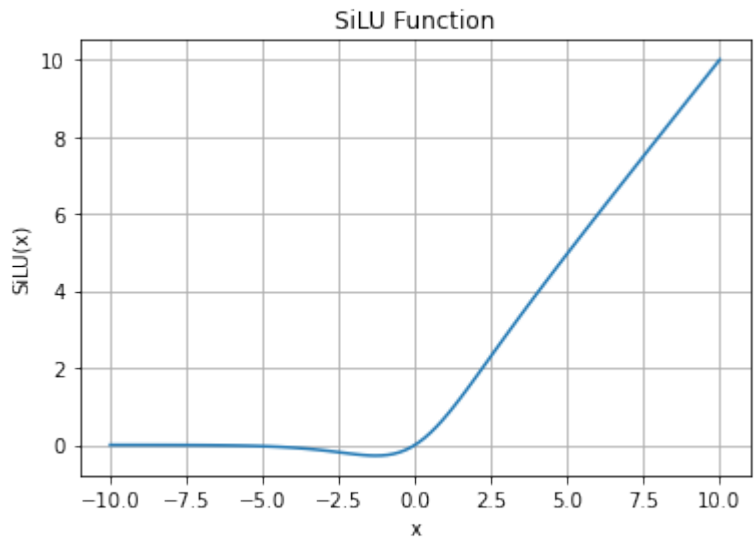
SiLU 函数定义为:

$$\text{SiLU}(x) = x * \sigma(x),$$

其中 x_i 是输入的元素, $\sigma(x)$ 是 Sigmoid 函数。

$$\text{sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)},$$

SiLU 函数图：



输入：

- **input** (Tensor) - *input* 即为上述公式中的 x 。数据类型为 float16 或 float32，维度任意。

输出：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.array([-1, 2, -3, 2, -1]), mindspore.float16)
>>> silu = nn.SiLU()
>>> output = silu(input)
>>> print(output)
[-0.269  1.762 -0.1423  1.762 -0.269]
```

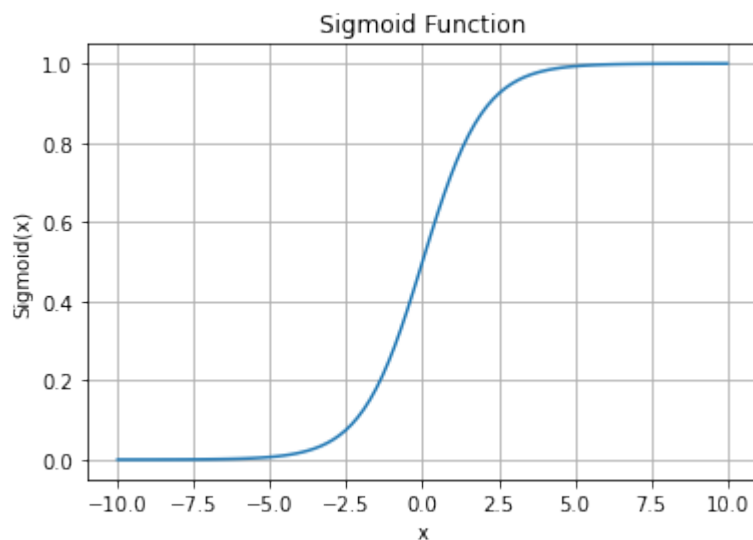
3.8.23 mindspore.nn.Sigmoid

class mindspore.nn.Sigmoid
逐元素计算 Sigmoid 激活函数。
Sigmoid 函数定义为：

$$\text{sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)},$$

其中 x_i 是 x 的一个元素。

Sigmoid 函数图：



输入：

- **input** (Tensor) - *input* 即为上述公式中的 x 。数据类型为 float16、float32、float64、complex64 或 complex128 的 Sigmoid 输入。任意维度的 Tensor。

输出：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> sigmoid = nn.Sigmoid()
>>> output = sigmoid(input)
>>> print(output)
[0.2688  0.11914 0.5      0.881   0.7305 ]
```

3.8.24 mindspore.nn.Softmin

class mindspore.nn.Softmin (axis=-1)

逐元素计算 Softmin 函数，它是二分类函数 `mindspore.nn.Sigmoid` 在多分类上的推广，目的是将多分类的结果以概率的形式展现出来。

对输入 Tensor 在轴 `axis` 上的元素计算其指数函数值，然后归一化到 `[0, 1]` 范围，总和为 1。

Softmin 定义为：

$$\text{softmin}(x_i) = \frac{\exp(-x_i)}{\sum_{j=0}^{n-1} \exp(-x_j)},$$

其中， x_i 是输入 Tensor 在轴 `axis` 上的第 i 个元素。

参数：

- **axis** (Union[int, tuple[int]], 可选) - 指定 Softmin 运算的轴 `axis`，假设输入 x 的维度为 `x.ndim`，则 `axis` 的范围为 `[-x.ndim, x.ndim)`，-1 表示最后一个维度。默认值：-1。CPU 环境下，`axis` 只支持 int 类型。

输入：

- **x** (Tensor) - 用于计算 Softmin 函数的 Tensor，数据类型为 float16 或 float32。

输出：

Tensor，数据类型和 shape 与 x 相同，取值范围为 `[0, 1]`。

异常：

- **TypeError** - `axis` 既不是 int 也不是 tuple。
- **TypeError** - x 的数据类型既不是 float16 也不是 float32。
- **ValueError** - `axis` 是长度小于 1 的 tuple。
- **ValueError** - `axis` 是一个 tuple，其元素不都在 `[-x.ndim, x.ndim)` 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # axis = -1 (default), and the sum of return value is 1.0.
>>> x = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> softmin = nn.Softmin()
>>> output = softmin(x)
>>> print(output)
[0.2341  0.636  0.0862  0.01165  0.03168 ]
```

3.8.25 mindspore.nn.Softmax

class mindspore.nn.Softmax(axis=-1)

逐元素计算 Softmax 激活函数，它是二分类函数 `mindspore.nn.Sigmoid` 在多分类上的推广，目的是将多分类的结果以概率的形式展现出来。

对输入 Tensor 在轴 *axis* 上的元素计算其指数函数值，然后归一化到 [0, 1] 范围，总和为 1。

Softmax 定义为：

$$\text{softmax}(\text{input}_i) = \frac{\exp(\text{input}_i)}{\sum_{j=0}^{n-1} \exp(\text{input}_j)},$$

其中， input_i 是输入 Tensor 在轴 *axis* 上的第 *i* 个元素。

参数：

- **axis** (int, 可选) - 指定 Softmax 运算的轴 *axis*，假设输入 *input* 的维度为 `input.ndim`，则 *axis* 的范围为 `[-input.ndim, input.ndim)`，-1 表示最后一个维度。默认值：-1。

输入：

- **input** (Tensor) - 用于计算 Softmax 函数的 Tensor。

输出：

Tensor，数据类型和 shape 与 *input* 相同，取值范围为 [0, 1]。

异常：

- **TypeError** - *axis* 既不是 int 也不是 tuple。
- **ValueError** - *axis* 是长度小于 1 的 tuple。
- **ValueError** - *axis* 是一个 tuple，其元素不都在 `[-input.ndim, input.ndim)` 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # axis = -1(default), and the sum of return value is 1.0.
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> softmax = nn.Softmax()
>>> output = softmax(input)
>>> print(output)
[0.03168 0.01166 0.0861 0.636 0.2341 ]
```

3.8.26 mindspore.nn.Softmax2d

class mindspore.nn.Softmax2d

应用于 2D 特征数据的 Softmax 函数。

将 Softmax 应用于具有 shape (C, H, W) 的输入 Tensor 的每个位置。

输入：

- **x** (Tensor) - Tensor 的 shape $(N, C_{in}, H_{in}, W_{in})$ 或者 (C_{in}, H_{in}, W_{in}) 。数据类型可以是 float16 或者 float32。

输出：

Tensor，数据类型和 shape 与 x 相同，取值范围为 $[0, 1]$ 。

异常：

- **TypeError** - x 的数据类型既不是 float16 也不是 float32。
- **ValueError** - 数据格式不是 “NCHW” 或者 “CHW”。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[[[0.1, 0.2]], [[0.3, 0.4]], [[0.6, 0.5]]]]), mindspore.float32)
>>> softmax2d = nn.Softmax2d()
>>> output = softmax2d(x)
>>> print(output)
[[[0.25838965 0.28001308]]
 [[0.31559783 0.34200877]]
 [[0.42601252 0.37797815]]]
```

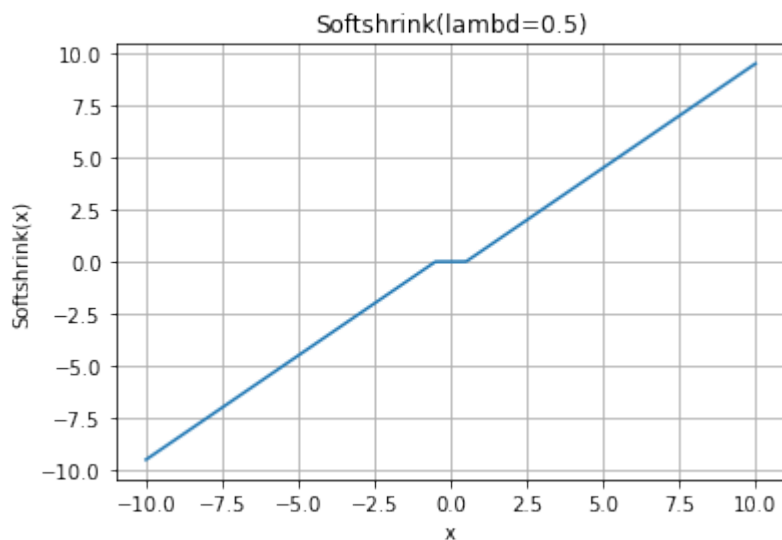
3.8.27 mindspore.nn.SoftShrink

class mindspore.nn.SoftShrink (*lambd=0.5*)

逐元素计算 SoftShrink 激活函数。

$$\text{SoftShrink}(x) = \begin{cases} x - \lambda, & \text{if } x > \lambda \\ x + \lambda, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

SoftShrink 函数图：



参数：

- **lambd** (number, 可选) - SoftShrink 公式中的 λ ，必须不小于零。默认值：0.5。

输入：

- **input** (Tensor) - SoftShrink 的输入，任意维度的 Tensor，数据类型为 float16 或 float32。
 - Ascend: float16、float32、bfloat16。
 - CPU/GPU: float16、float32。

输出：

Tensor，shape 和数据类型与 *input* 相同。

异常：

- **TypeError** - *lambd* 不是 float、int 或 bool。
- **TypeError** - *input* 不是 tensor。
- **TypeError** - *input* 的数据类型 float16、float32 或 bfloat16。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.array([[ 0.5297,  0.7871,  1.1754], [ 0.7836,  0.6218, -1.1542]]),
↳mindspore.float16)
>>> softshrink = nn.SoftShrink()
>>> output = softshrink(input)
>>> print(output)
[[ 0.02979  0.287   0.676  ]
 [ 0.2837   0.1216 -0.6543  ]]
```

3.8.28 mindspore.nn.Tanh

class mindspore.nn.Tanh

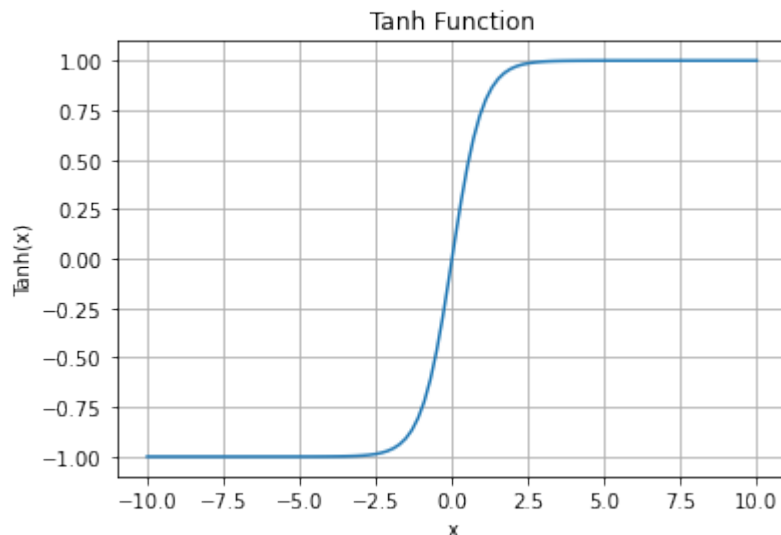
逐元素计算 Tanh（双曲正切值）激活函数，返回一个新的 Tensor。

Tanh 函数定义为：

$$\tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 Tensor 的元素。

Tanh 函数图：



输入：

- **x** (Tensor) - 任意维度的 Tensor，数据类型为 float16 或 float32 的输入。

输出：

Tensor，数据类型和 shape 与 x 的相同。

异常：

- **TypeError** - x 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([1, 2, 3, 2, 1]), mindspore.float16)
>>> tanh = nn.Tanh()
>>> output = tanh(x)
>>> print(output)
[0.7617 0.964 0.995 0.964 0.7617]
```

3.8.29 mindspore.nn.Tanhshrink

class mindspore.nn.Tanhshrink

逐元素计算 Tanhshrink 激活函数，返回一个新的 Tensor。

Tanhshrink 函数定义为：

$$\text{tanhshrink}(x_i) = x_i - \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = x_i - \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 Tensor 的元素。

输入：

- **x** (Tensor) - 任意维度的 Tensor。

输出：

Tensor，其 shape 与 x 的相同。

异常：

- **TypeError** - x 不是一个 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([1, 2, 3, 2, 1]), ms.float16)
>>> tanhshrink = nn.Tanhshrink()
>>> output = tanhshrink(x)
>>> print(output)
[0.2383 1.036 2.004 1.036 0.2383]
```

3.8.30 mindspore.nn.Threshold

class mindspore.nn.Threshold(*threshold, value*)

逐元素计算 Threshold 激活函数。

Threshold 定义为：

$$y = \begin{cases} x, & \text{if } x > \text{threshold} \\ \text{value}, & \text{otherwise} \end{cases}$$

参数：

- **threshold** (Union[int, float]) - 阈值。
- **value** (Union[int, float]) - 输入 Tensor 中元素小于阈值时的填充值。

输入：

- **input_x** (Tensor) - 输入 Tensor，数据类型为 float16 或 float32。

输出：

Tensor，数据类型和 shape 与 *input_x* 的相同。

异常：

- **TypeError** - *threshold* 不是浮点数或整数。
- **TypeError** - *value* 不是浮点数或整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> m = nn.Threshold(0.1, 20)
>>> inputs = Tensor([0.1, 0.2, 0.3], mindspore.float32)
>>> outputs = m(inputs)
>>> print(outputs)
[ 20.0      0.2      0.3]
```

3.9 线性层

接口名	概述	支持平台
<code>mindspore.nn.Dense</code>	全连接层。	Ascend GPU CPU
<code>mindspore.nn.BiDense</code>	双线性全连接层。	Ascend GPU CPU

3.9.1 mindspore.nn.Dense

class mindspore.nn.Dense (*in_channels*, *out_channels*, *weight_init*=None, *bias_init*=None, *has_bias*=True, *activation*=None, *dtype*=mstype.float32)

全连接层。

适用于输入的密集连接层。公式如下：

$$\text{outputs} = \text{activation}(X * \text{kernel} + \text{bias}),$$

其中 X 是输入 Tensor，`activation` 是激活函数，`kernel` 是一个权重矩阵，其数据类型与 X 相同，`bias` 是一个偏置向量，其数据类型与 X 相同（仅当 `has_bias` 为 `True` 时）。

警告：在 Ascend 硬件平台下，设置 PYNATIVE 或 KBK 模式时，如果 `bias` 为 `False`， x 不可以大于 6D。

参数：

- in_channels** (int) - Dense 层输入 Tensor 的空间维度。
- out_channels** (int) - Dense 层输出 Tensor 的空间维度。
- weight_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 权重参数的初始化方法。数据类型与 x 相同。str 的值引用自函数 `initializer`。默认值：None，权重使用 HeUniform 初始化。
- bias_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 偏置参数的初始化方法。数据类型与 x 相同。str 的值引用自函数 `initializer`。默认值：None，偏差使用 Uniform 初始化。
- has_bias** (bool, 可选) - 是否使用偏置向量 `bias`。默认值：True。
- activation** (Union[str, Cell, Primitive, None], 可选) - 应用于全连接层输出的激活函数。可指定激活函数名，如 `'relu'`，或具体激活函数，如 `mindspore.nn.ReLU`。默认值：None。
- dtype** (`mindspore.dtype`, 可选) - Parameter 的数据类型。默认值：`mstype.float32`。当 `weight_init` 是 Tensor 时，Parameter 的数据类型与 `weight_init` 的数据类型一致，其他情况 Parameter 的数据类型跟 `dtype` 一致，`bias_init` 同理。

输入：

- x** (Tensor) - shape 为 $(*, in\_channels)$ 的 Tensor。参数中的 `in_channels` 应等于输入中的 `in_channels`。

输出：

shape 为 $(*, out\_channels)$ 的 Tensor。

异常：

- **TypeError** - *in_channels* 或 *out_channels* 不是整数。
- **TypeError** - *has_bias* 不是 bool 值。
- **TypeError** - *activation* 不是 str、Cell、Primitive 或者 None。
- **ValueError** - *weight_init* 的 shape 长度不等于 2, *weight_init* 的 shape[0] 不等于 *out_channels*, 或者 *weight_init* 的 shape[1] 不等于 *in_channels*。
- **ValueError** - *bias_init* 的 shape 长度不等于 1 或 *bias_init* 的 shape[0] 不等于 *out_channels*。
- **RuntimeError** - 在 Ascend 硬件平台下, 设置 PYNATIVE 或 KBK 模式时, *bias* 为 False 且 *x* 大于 6D。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[180, 234, 154], [244, 48, 247]]), mindspore.float32)
>>> net = nn.Dense(3, 4)
>>> output = net(x)
>>> print(output.shape)
(2, 4)
```

3.9.2 mindspore.nn.BiDense

class mindspore.nn.BiDense (*in1_channels, in2_channels, out_channels, weight_init=None, bias_init=None, has_bias=True, dtype=mstype.float32*)

双线性全连接层。

两个输入的密集连接层。公式如下：

$$y = x_1^T A x_2 + b,$$

其中 x_1 是第一个输入 Tensor, x_2 是第二个输入 Tensor, A 是一个权重矩阵, 其数据类型与 x_* 相同, b 是一个偏置向量, 其数据类型与 x_* 相同 (仅当 *has_bias* 为 True 时)。

参数：

- **in1_channels** (int) - BiDense 层第一个输入 Tensor 的空间维度。
- **in2_channels** (int) - BiDense 层第二个输入 Tensor 的空间维度。
- **out_channels** (int) - BiDense 层输出 Tensor 的空间维度。
- **weight_init** (Union[Tensor, str, Initializer, numbers.Number]) - 权重参数的初始化方法。str 的值引用自函数 *initializer*。默认值：None。
- **bias_init** (Union[Tensor, str, Initializer, numbers.Number]) - 偏置参数的初始化方法。str 的值引用自函数 *initializer*。默认值：None。
- **has_bias** (bool) - 是否使用偏置向量 *bias*。默认值：True。

- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: `mstype.float32`。

形状:

- **input1** - $(*, H_{in1})$, 其中 $H_{in1} = \text{in1\_channels}$, * 为任意维度. input1 除最后一维外的维度需与其他输入保持一致。
- **input2** - $(*, H_{in2})$, 其中 $H_{in2} = \text{in2\_channels}$, * 为任意维度. input2 除最后一维外的维度需与其他输入保持一致。
- **output** - $(*, H_{out})$, 其中 $H_{out} = \text{out\_channels}$, * 为任意维度. output 除最后一维外的维度需与所有输入保持一致。

数据类型:

- **input1** (Tensor) - 数据类型必须为 float16 或 float32, 且与 input2 一致。
- **input2** (Tensor) - 数据类型必须为 float16 或 float32, 且与 input1 一致。
- **output** (Tensor) - 数据类型与输入保持一致。

权重:

- **weight** (Parameter) - 权重参数, shape 为 $(\text{out\_channels}, \text{in1\_channels}, \text{in2\_channels})$ 。当 *weight_init* 设为 None 时, 其初始化值服从 $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, 其中 $k = \frac{1}{\text{in1\_channels}}$ 。
- **bias** (Parameter) - 偏置参数, shape 为 (out_channels) 。当 *has_bias* 设为 True 且 *bias_init* 设为 None 时, 其初始化值服从 $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, 其中 $k = \frac{1}{\text{in1\_channels}}$ 。

异常:

- **TypeError** - *in_channels* 或 *out_channels* 不是整数。
- **TypeError** - *has_bias* 不是 bool 值。
- **ValueError** - *weight_init* 的 shape 长度不等于 3, *weight_init* 的 *shape*[0] 不等于 *out_channels*, 或者 *weight_init* 的 *shape*[1] 不等于 *in1_channels*, 或者 *weight_init* 的 *shape*[2] 不等于 *in2_channels*。
- **ValueError** - *bias_init* 的 shape 长度不等于 1 或 *bias_init* 的 *shape*[0] 不等于 *out_channels*。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x1 = Tensor(np.random.randn(128, 20), mindspore.float32)
>>> x2 = Tensor(np.random.randn(128, 30), mindspore.float32)
>>> net = nn.BiDense(20, 30, 40)
>>> output = net(x1, x2)
>>> print(output.shape)
(128, 40)
```

3.10 Dropout 层

接口名	概述	支持平台
<code>mindspore.nn.Dropout</code>	随机丢弃层。	Ascend GPU CPU
<code>mindspore.nn.Dropout1d</code>	在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零（对于 shape 为 (N, C, L) 的三维 Tensor，其通道特征图指的是后一维 L 的一维特征图）。	Ascend GPU CPU
<code>mindspore.nn.Dropout2d</code>	在训练期间，服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道置为零（对于 shape 为 $NCHW$ 的四维 Tensor，其通道特征图指的是后两维 HW 的二维特征图）。	Ascend GPU CPU
<code>mindspore.nn.Dropout3d</code>	在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零（对于 shape 为 $NCDHW$ 的五维 Tensor，其通道特征图指的是后三维 DHW 的三维特征图）。	Ascend GPU CPU

3.10.1 mindspore.nn.Dropout

class mindspore.nn.Dropout (*keep_prob=0.5, p=None, dtype=mstype.float32*)
随机丢弃层。

Dropout 是一种正则化手段，通过阻止神经元节点间的相关性来减少过拟合。该操作根据丢弃概率 p ，在训练过程中随机将一些神经元输出设置为 0。并且训练过程中返回值会乘以 $\frac{1}{1-p}$ 。在推理过程中，此层返回与 x 相同的 Tensor。

论文 [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#) 中提出了该技术，并证明其能有效地减少过度拟合，防止神经元共适应。更多详细信息，请参见 [Improving neural networks by preventing co-adaptation of feature detectors](#)。

说明:

- 训练过程中每步对同一通道（或神经元）独立进行丢弃。
- `keep_prob` 参数会在未来版本删除，请使用 `p` 参数代替它。`p` 表示输入 Tensor 中元素设置成 0 的概率。
- `dtype` 参数会在未来版本删除。不建议使用这个参数。

参数:

- **keep_prob** (float) - 废弃。输入神经元保留率，数值范围介于 (0, 1] 之间。例如，`keep_prob=0.9`，删除 10% 的神经元。默认值：0.5。
- **p** (Union[float, int, None]) - 输入神经元丢弃率，数值范围介于 [0, 1) 之间。例如，`p=0.9`，删除 90% 的神经元。默认值：None。
- **dtype** (`mindspore.dtype`) - `input` 的数据类型。默认值：`mstype.float32`。

输入:

- **x** (Tensor) - Dropout 的输入。数据类型必须为 float16 或 float32。`x` 的 shape 不能小于 1。

输出:

Tensor，输出为 Tensor，其 shape 与 `x` shape 相同。

异常:

- **TypeError** - *keep_prob* 不是浮点数。
- **TypeError** - *p* 数据类型不是 float 或 int。
- **TypeError** - *x* 的 dtype 既不是 float16 也不是 float32。
- **ValueError** - *keep_prob* 不在范围 (0, 1] 之间。
- **ValueError** - *p* 不在范围 [0, 1) 之间。
- **ValueError** - *x* 的 shape 长度小于 1。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.ones([2, 2, 3]), mindspore.float32)
>>> net = nn.Dropout(p=0.2)
>>> net.set_train()
>>> output = net(x)
>>> print(output.shape)
(2, 2, 3)
```

3.10.2 mindspore.nn.Dropout1d

class mindspore.nn.Dropout1d (*p=0.5*)

在训练期间，以服从伯努利分布的概率 *p* 随机将输入 Tensor 的某些通道归零（对于 shape 为 (*N, C, L*) 的三维 Tensor，其通道特征图指的是后一维 *L* 的一维特征图）。例如，在批处理输入中 *i_th* 批，*j_th* 通道的 *input[i, j]* 1D Tensor 是一个待处理数据。每个通道将会独立依据伯努利分布概率 *p* 来确定是否被清零。

论文 [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#) 中提出了该技术，并证明其能有效地减少过度拟合，防止神经元共适应。更多详细信息，请参见 [Improving neural networks by preventing co-adaptation of feature detectors](#)。

Dropout1d 可以提高通道特征映射之间的独立性。

参数：

- **p** (float, 可选) - 通道的丢弃概率，介于 0 和 1 之间，例如 *p* = 0.8，意味着 80% 的清零概率。默认值：0.5。

输入：

- **x** (Tensor) - 一个 shape 为 (*N, C, L*) 或 (*C, L*) 的 3D 或 2D Tensor，其中 *N* 是批处理大小，*C* 是通道数，*L* 是特征长度。其数据类型应为 int8、int16、int32、int64、float16、float32 或 float64。

输出：

Tensor，具有与输入 *x* 相同的 shape 和数据类型。

异常：

- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *p* 的数据类型不是 float。
- **ValueError** - *p* 值不在 [0.0, 1.0] 之间。
- **ValueError** - *x* 的维度不是 2D 或 3D。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> op = ms.nn.Dropout1d(p=0.6)
>>> op.training = True
>>> a = ms.Tensor(np.ones((3, 3)), ms.float32)
>>> output = op(a)
```

3.10.3 mindspore.nn.Dropout2d

class mindspore.nn.Dropout2d($p=0.5$)

在训练期间，服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道置为零（对于 shape 为 $NCHW$ 的四维 Tensor，其通道特征图指的是后两维 HW 的二维特征图）。例如，对于批处理输入中第 i 批样本的第 j 个通道，是一个待处理的二维 Tensor $input[i, j]$ 。在每次前向传播时，每个通道将会独立地以概率 p 来确定是否被置为零。

Dropout2d 可以提高通道特征映射之间的独立性。

更多参考详见 `mindspore.ops.dropout2d()`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> dropout = nn.Dropout2d(p=0.5)
>>> x = Tensor(np.ones([2, 1, 2, 3]), mindspore.float32)
>>> output = dropout(x)
>>> print(output.shape)
(2, 1, 2, 3)
```

3.10.4 mindspore.nn.Dropout3d

class mindspore.nn.Dropout3d($p=0.5$)

在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零（对于 shape 为 $NCDHW$ 的五维 Tensor，其通道特征图指的是后三维 DHW 的三维特征图）。例如，在批处理输入中 i_th 批， j_th 通道的 $input[i, j]$ 3D Tensor 是一个待处理数据。每个通道将会独立依据伯努利分布概率 p 来确定是否被清零。

Dropout3d 可以提高通道特征映射之间的独立性。

更多参考详见 `mindspore.ops.dropout3d()`。

支持平台：

Ascend GPU CPU

样例：


```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> dropout = nn.Dropout3d(p=0.5)
>>> x = Tensor(np.ones([2, 1, 2, 1, 2]), mindspore.float32)
>>> output = dropout(x)
>>> print(output.shape)
(2, 1, 2, 1, 2)
```

3.11 归一化层

接口名	概述	支持平台
<code>mindspore.nn.BatchNorm1d</code>	在二维或三维输入（mini-batch 一维输入或二维输入）上应用批归一化（Batch Normalization Layer），避免内部协变量偏移。	Ascend GPU CPU
<code>mindspore.nn.BatchNorm2d</code>	在四维输入（具有额外通道维度的小批量二维输入）上应用批归一化处理（Batch Normalization Layer），以避免内部协变量偏移。	Ascend GPU CPU
<code>mindspore.nn.BatchNorm3d</code>	对输入的五维数据进行批归一化（Batch Normalization Layer）。	Ascend GPU CPU
<code>mindspore.nn.GroupNorm</code>	在 mini-batch 输入上进行组归一化。	Ascend GPU CPU
<code>mindspore.nn.InstanceNorm1d</code>	该层在三维输入（带有额外通道维度的 mini-batch 一维输入）上应用实例归一化。	GPU
<code>mindspore.nn.InstanceNorm2d</code>	该层在四维输入（带有额外通道维度的 mini-batch 二维输入）上应用实例归一化。	GPU
<code>mindspore.nn.InstanceNorm3d</code>	该层在五维输入（带有额外通道维度的 mini-batch 三维输入）上应用实例归一化。	GPU
<code>mindspore.nn.LayerNorm</code>	在 mini-batch 输入上应用层归一化（Layer Normalization）。	Ascend GPU CPU
<code>mindspore.nn.SyncBatchNorm</code>	在 N 维输入上进行跨设备同步批归一化（Batch Normalization，BN）。	Ascend

3.11.1 mindspore.nn.BatchNorm1d

```
class mindspore.nn.BatchNorm1d(num_features, eps=1e-5, momentum=0.9, affine=True, gamma_init='ones',
                                beta_init='zeros', moving_mean_init='zeros', moving_var_init='ones',
                                use_batch_statistics=None, data_format='NCHW', dtype=mstype.float32)
```

在二维或三维输入（mini-batch 一维输入或二维输入）上应用批归一化（Batch Normalization Layer），避免内部协变量偏移。归一化在卷积网络中被广泛的应用。请见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。

使用 mini-batch 数据和学习参数进行训练，计算公式如下。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

说明：BatchNorm 的实现在图模式和 PyNative 模式下是不同的，因此不建议在网络初始化后更改其模式。

参数：

- **num_features** (int) - 特征数量或输入 x 中的通道数量 C 。
- **eps** (float) - ϵ 加在分母上的值, 以确保数值稳定。默认值: $1e-5$ 。
- **momentum** (float) - 动态均值和动态方差所使用的动量。默认值: 0.9 。
- **affine** (bool) - bool 类型。设置为 True 时, 可学习到 γ 和 β 值。默认值: True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number]) - γ 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number]) - β 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'zeros'。
- **moving_mean_init** (Union[Tensor, str, Initializer, numbers.Number]) - 动态平均值的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'zeros'。
- **moving_var_init** (Union[Tensor, str, Initializer, numbers.Number]) - 动态方差的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'ones'。
- **use_batch_statistics** (bool) - 如果为 True, 则使用当前批次数据的平均值和方差值。如果为 False, 则使用指定的平均值和方差值。如果为 None, 训练时, 将使用当前批次数据的均值和方差, 并更新动态均值和方差, 验证过程将直接使用动态均值和方差。默认值: None。
- **data_format** (str) - 数据格式可为 'NHWC' 或 'NCHW'。默认值: 'NCHW'。
- **dtype** (`mindspore.dtype`) - Parameters 的 dtype。默认值: `mstype.float32`。

输入:

- **x** (Tensor) - 输入 shape 为 (N, C) 或 (N, C, L) 的 Tensor, 其中 N 为 batch, C 为特征数量或通道数量, L 为序列长度。支持数据类型: float16、float32。

输出:

Tensor, 归一化后的 Tensor, shape 为 (N, C) 或 (N, C, L) 。

异常:

- **TypeError** - `num_features` 不是整数。
- **TypeError** - `eps` 不是浮点数。
- **ValueError** - `num_features` 小于 1。
- **ValueError** - `momentum` 不在范围 $[0, 1]$ 内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> net = ms.nn.BatchNorm1d(num_features=4)
>>> x = ms.Tensor(np.array([[0.7, 0.5, 0.5, 0.6],
...                          [0.5, 0.4, 0.6, 0.9]]).astype(np.float32))
>>> output = net(x)
>>> print(output)
[[ 0.6999965  0.4999975  0.4999975  0.59999704 ]
 [ 0.4999975  0.399998  0.59999704  0.89999545 ]]
```

3.11.2 mindspore.nn.BatchNorm2d

```
class mindspore.nn.BatchNorm2d(num_features, eps=1e-5, momentum=0.9, affine=True, gamma_init='ones',
                                beta_init='zeros', moving_mean_init='zeros', moving_var_init='ones',
                                use_batch_statistics=None, data_format='NCHW', dtype=mstype.float32)
```

在四维输入（具有额外通道维度的小批量二维输入）上应用批归一化处理（Batch Normalization Layer），以避免内部协变量偏移。批归一化广泛应用于卷积网络中。详情见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。使用 mini-batch 数据和学习参数进行训练，参数详见以下公式：

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

说明：BatchNorm 的实现在图模式和 PyNative 模式下是不同的，因此不建议在网络初始化后更改其模式。

需要注意的是，更新 *moving_mean* 和 *moving_var* 的公式为：

$$\begin{aligned} \text{moving\_mean} &= \text{moving\_mean} * \text{momentum} + (1 - \text{momentum}) * \text{current\_mean} \\ \text{moving\_var} &= \text{moving\_var} * \text{momentum} + (1 - \text{momentum}) * \text{current\_var} \end{aligned}$$

其中，*moving_mean* 是更新后的均值，*moving_var* 是更新后的方差， $\cdot^2$ 是每一批的数据的观测值（分别为均值和方差）。

参数：

- **num_features** (int) - 通道数量。即输入 Tensor 的 shape (N, C, H, W) 中的 C 。
- **eps** (float, 可选) - ϵ 加在分母上的值，以确保数值稳定。默认值：1e-5。
- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。默认值：0.9。
- **affine** (bool, 可选) - bool 类型。设置为 True 时，可学习 γ 和 β 值。默认值：True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - γ 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`，包括 'zeros'、'ones' 等。默认值：'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - β 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`，包括 'zeros'、'ones' 等。默认值：'zeros'。
- **moving_mean_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 动态平均值的初始化方法。str 的值引用自函数 `mindspore.common.initializer`，包括 'zeros'、'ones' 等。默认值：'zeros'。
- **moving_var_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 动态方差的初始化方法。str 的值引用自函数 `mindspore.common.initializer`，包括 'zeros'、'ones' 等。默认值：'ones'。
- **use_batch_statistics** (bool, 可选) - 默认值：None。
 - 如果为 True，则使用当前批处理数据的平均值和方差值，并跟踪运行平均值和运行方差。
 - 如果为 False，则使用指定值的平均值和方差值，不跟踪统计值。
 - 如果为 None，则根据训练和验证模式自动设置 *use_batch_statistics* 为 True 或 False。在训练时，*use_batch_statistics* 会设置为 True。在验证时，*use_batch_statistics* 会自动设置为 False。
- **data_format** (str, 可选) - 数据格式可为 'NHWC' 或 'NCHW'。默认值：'NCHW'。
- **dtype** (`mindspore.dtype`, 可选) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **x** (Tensor) - 输入 shape 为 (N, C, H, W) 的 Tensor。支持数据类型：float16、float32。

输出：

Tensor, 归一化后的 Tensor, shape 为 (N, C, H, W) 。

异常：

- **TypeError** - `num_features` 不是 int。
- **TypeError** - `eps` 不是 float。
- **ValueError** - `num_features` 小于 1。
- **ValueError** - `momentum` 不在范围 $[0, 1]$ 内。
- **ValueError - `data_format` 既不是 'NHWC' 也不是 'NCHW'。**

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> net = ms.nn.BatchNorm2d(num_features=3)
>>> x = ms.Tensor(np.ones([1, 3, 2, 2]).astype(np.float32))
>>> output = net(x)
>>> print(output)
[[[ [ 0.999995 0.999995 ]
    [ 0.999995 0.999995 ]]
  [ [ 0.999995 0.999995 ]
    [ 0.999995 0.999995 ]]
  [ [ 0.999995 0.999995 ]
    [ 0.999995 0.999995 ]]]]
```

3.11.3 mindspore.nn.BatchNorm3d

class mindspore.nn.BatchNorm3d(*num_features*, *eps=1e-5*, *momentum=0.9*, *affine=True*, *gamma_init='ones'*, *beta_init='zeros'*, *moving_mean_init='zeros'*, *moving_var_init='ones'*, *use_batch_statistics=None*, *dtype=mstype.float32*)

对输入的五维数据进行批归一化 (Batch Normalization Layer)。

在五维输入（带有附加通道维度的 mini-batch 三维输入）上应用批归一化 (Batch Normalization Layer)，避免内部协变量偏移。归一化在卷积网络中得到了广泛的应用。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

说明：BatchNorm 的实现在图模式和 PyNative 模式下是不同的，因此不建议在网络初始化后更改其模式。

需要注意的是，更新 `running_mean` 和 `running_var` 的公式为 $\hat{x}_{\text{new}} = (1 - \text{momentum}) \times x_t + \text{momentum} \times \hat{x}$ ，其中 $\hat{x}$ 是估计的统计量， x_t 是新的观察值。

参数：

- **num_features** (int) - 指定输入 Tensor 的通道数量。输入 Tensor 的 size 为 (N, C, D, H, W) 。
- **eps** (float) - 加在分母上的值，以确保数值稳定。默认值： $1e-5$ 。

- **momentum** (float) - 动态均值和动态方差所使用的动量。默认值: 0.9。
- **affine** (bool) - bool 类型。设置为 True 时, 可以学习 γ 和 β 。默认值: True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number]) - γ 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number]) - β 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'zeros'。
- **moving_mean_init** (Union[Tensor, str, Initializer, numbers.Number]) - 动态均值和动态方差所使用的动量。平均值的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'zeros'。
- **moving_var_init** (Union[Tensor, str, Initializer, numbers.Number]) - 动态均值和动态方差所使用的动量。方差的初始化方法。str 的值引用自函数 `mindspore.common.initializer`, 包括 'zeros'、'ones' 等。默认值: 'ones'。
- **use_batch_statistics** (bool) - 如果为 True, 则使用当前批次数据的平均值和方差值。如果为 False, 则使用指定的平均值和方差值。如果为 None, 训练时, 将使用当前批次数据的均值和方差, 并更新动态均值和方差, 验证过程将直接使用动态均值和方差。默认值: None。
- **dtype** (`mindspore.dtype`) - Parameters 的 dtype。默认值: `mstype.float32`。

输入:

- **x** (Tensor) - 输入 shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor。支持数据类型: float16、float32。

输出:

Tensor, 归一化后的 Tensor, shape 为 $(N, C_{out}, D_{out}, H_{out}, W_{out})$ 。

异常:

- **TypeError** - `num_features` 不是整数。
- **TypeError** - `eps` 不是浮点数。
- **ValueError** - `num_features` 小于 1。
- **ValueError** - `momentum` 不在范围 [0, 1] 内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> net = ms.nn.BatchNorm3d(num_features=3)
>>> x = ms.Tensor(np.ones([16, 3, 10, 32, 32]).astype(np.float32))
>>> output = net(x)
>>> print(output.shape)
(16, 3, 10, 32, 32)
```

3.11.4 mindspore.nn.GroupNorm

class mindspore.nn.GroupNorm (num_groups, num_channels, eps=1e-05, affine=True, gamma_init='ones', beta_init='zeros', dtype=mstype.float32)

在 mini-batch 输入上进行组归一化。

Group Normalization 广泛用于递归神经网络中。适用单个训练用例的 mini-batch 输入归一化，详见论文 [Group Normalization](#)。

Group Normalization 把通道划分为组，然后计算每一组之内的均值和方差，以进行归一化。其中 γ 是通过训练学习得到的 scale 值， β 是通过训练学习得到的 shift 值。

公式如下，

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

参数：

- **num_groups** (int) - 沿通道维度待划分的组数。
- **num_channels** (int) - 输入的通道数。
- **eps** (float) - 添加到分母中的值，以确保数值稳定。默认值：1e-05。
- **affine** (bool) - 指定是否需要添加仿射变换参数。当设置为 True 时，给该层添加可学习的仿射变换参数 gamma 与 beta。默认值：True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number]) - gamma 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`，包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值：'ones'。如果 gamma_init 是 Tensor，则 shape 必须为 (num_channels)。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number]) - beta 参数的初始化方法。str 的值引用自函数 `mindspore.common.initializer`，包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值：'zeros'。如果 gamma_init 是 Tensor，则 shape 必须为 (num_channels)。
- **dtype** (`mindspore.dtype`) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **x** (Tensor) - shape 为 (N, C, *) 的特征输入，其中 * 表示任意的附加维度。

输出：

Tensor，标准化和缩放的偏移 Tensor，具有与 x 相同的 shape 和数据类型。

异常：

- **TypeError** - num_groups 或 num_channels 不是 int。
- **TypeError** - eps 不是 float。
- **TypeError** - affine 不是 bool。
- **ValueError** - num_groups 或 num_channels 小于 1。
- **ValueError** - num_channels 未被 num_groups 整除。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> group_norm_op = ms.nn.GroupNorm(2, 2)
>>> x = ms.Tensor(np.ones([1, 2, 4, 4], np.float32))
>>> output = group_norm_op(x)
>>> print(output)
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]]
```

3.11.5 mindspore.nn.InstanceNorm1d

class mindspore.nn.InstanceNorm1d (*num_features*, *eps*=1e-5, *momentum*=0.1, *affine*=True, *gamma_init*='ones', *beta_init*='zeros', *dtype*=mstype.float32)

该层在三维输入（带有额外通道维度的 mini-batch 一维输入）上应用实例归一化。详见论文 [Instance Normalization: The Missing Ingredient for Fast Stylization](#)。使用 mini-batch 数据和学习参数进行训练，参数见如下公式。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中 γ 和 β 是可学习的参数向量，如果 *affine* 为 True，则大小为 *num_features*。通过偏置估计函数计算标准偏差。此层使用从训练和验证模式的输入数据计算得到的实例数据。

InstanceNorm1d 和 BatchNorm1d 类似。不同之处在于 InstanceNorm1d 应用于 RGB 图像等通道数据的每个通道，而 BatchNorm1d 通常应用于批处理。

说明：需要注意的是，更新滑动平均和滑动方差的公式为 $\hat{x}_{\text{new}} = (1 - \text{momentum}) \times x_t + \text{momentum} \times \hat{x}$ ，其中 $\hat{x}$ 是估计的统计量， x_t 是新的观察值。

参数：

- **num_features** (int) - 通道数量，输入 Tensor shape (*N*, *C*, *L*) 中的 *C*。
- **eps** (float) - 添加到分母中的值，以确保数值稳定。默认值：1e-5。
- **momentum** (float) - 动态均值和动态方差所使用的动量。默认值：0.1。
- **affine** (bool) - bool 类型。设置为 True 时，可以学习 gamma 和 beta 参数。默认值：True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number]) - gamma 参数的初始化方法。str 的值引用自函数 *initializer*，包括 'zeros'、'ones' 等。使用 Tensor 作为初始化参数时，shape 必须为 (*C*)。默认值：'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number]) - beta 参数的初始化方法。str 的值引用自函数 *initializer*，包括 'zeros'、'ones' 等。使用 Tensor 作为初始化参数时，shape 必须为 (*C*)。默认值：'zeros'。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **x** (Tensor) - shape 为 (*N*, *C*, *L*) 的 Tensor。数据类型为 float16 或 float32。

输出：

Tensor，归一化，缩放，偏移后的 Tensor，其 shape 为 (N, C, L) 。类型和 shape 与 x 相同。

异常：

- **TypeError** - `num_features` 不是整数。
- **TypeError** - `eps` 的类型不是 float。
- **TypeError** - `momentum` 的类型不是 float。
- **TypeError** - `affine` 不是 bool。
- **TypeError** - `gamma_init` / `beta_init` 的类型不相同，或者初始化的元素类型不是 float32。
- **ValueError** - `num_features` 小于 1。
- **ValueError** - `momentum` 不在范围 $[0, 1]$ 内。
- **ValueError** - `gamma_init` / `beta_init` 的 shape 不为 (C) 。
- **KeyError** - `gamma_init` / `beta_init` 中的任何一个不是 str，并且不存在继承自 `Initializer` 的同名类。

支持平台：

GPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.InstanceNorm1d(3)
>>> x = ms.Tensor(np.ones([2, 3, 5]), ms.float32)
>>> output = net(x)
>>> print(output.shape)
(2, 3, 5)
```

3.11.6 mindspore.nn.InstanceNorm2d

class mindspore.nn.InstanceNorm2d(`num_features`, `eps=1e-5`, `momentum=0.1`, `affine=True`, `gamma_init='ones'`, `beta_init='zeros'`, `dtype=mstype.float32`)

该层在四维输入（带有额外通道维度的 mini-batch 二维输入）上应用实例归一化。详见论文 [Instance Normalization: The Missing Ingredient for Fast Stylization](#)。使用 mini-batch 数据和学习参数进行训练，参数见如下公式。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中 γ 和 β 是可学习的参数向量，如果 `affine` 为 `True`，则大小为 `num_features`。通过偏置估计函数计算标准偏差。

此层使用从训练和验证模式的输入数据计算得到的实例数据。

`InstanceNorm2d` 和 `BatchNorm2d` 类似。不同之处在于 `InstanceNorm2d` 应用于 RGB 图像等通道数据的每个通道，而 `BatchNorm2d` 通常应用于批处理。

说明：需要注意的是，更新滑动平均和滑动方差的公式为 $\hat{x}_{\text{new}} = (1 - \text{momentum}) \times x_t + \text{momentum} \times \hat{x}$ ，其中 $\hat{x}$ 是估计的统计量， x_t 是新的观察值。

参数：

- **num_features** (int) - 通道数量, 输入 Tensor shape (N, C, H, W) 中的 C 。
- **eps** (float) - 添加到分母中的值, 以确保数值稳定。默认值: $1e-5$ 。
- **momentum** (float) - 动态均值和动态方差所使用的动量。默认值: 0.1 。
- **affine** (bool) - bool 类型。设置为 True 时, 可以学习 **gamma** 和 **beta** 参数。默认值: True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number]) - **gamma** 参数的初始化方法。**str** 的值引用自函数 *initializer*, 包括 'zeros'、'ones' 等。使用 Tensor 作为初始化参数时, **shape** 必须为 (C) 。默认值: 'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number]) - **beta** 参数的初始化方法。**str** 的值引用自函数 *initializer*, 包括 'zeros'、'ones' 等。使用 Tensor 作为初始化参数时, **shape** 必须为 (C) 。默认值: 'zeros'。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值: *mstype.float32*。

输入:

- **x** (Tensor) - shape 为 (N, C, H, W) 的 Tensor。数据类型为 float16 或 float32。

输出:

Tensor, 归一化, 缩放, 偏移后的 Tensor, 其 shape 为 (N, C, H, W) 。类型和 shape 与 *x* 相同。

异常:

- **TypeError** - *num_features* 不是整数。
- **TypeError** - *eps* 的类型不是 float。
- **TypeError** - *momentum* 的类型不是 float。
- **TypeError** - *affine* 不是 bool。
- **TypeError** - *gamma_init* / *beta_init* 的类型不相同, 或者初始化的元素类型不是 float32。
- **ValueError** - *num_features* 小于 1。
- **ValueError** - *momentum* 不在范围 $[0, 1]$ 内。
- **ValueError** - *gamma_init* / *beta_init* 的 shape 不为 (C) 。
- **KeyError** - *gamma_init* / *beta_init* 中的任何一个不是 str, 并且不存在继承自 *Initializer* 的同名类。

支持平台:

GPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.InstanceNorm2d(3)
>>> x = ms.Tensor(np.ones([2, 3, 2, 2]), ms.float32)
>>> output = net(x)
>>> print(output.shape)
(2, 3, 2, 2)
```

3.11.7 mindspore.nn.InstanceNorm3d

class mindspore.nn.InstanceNorm3d(*num_features*, *eps*=1e-5, *momentum*=0.1, *affine*=True, *gamma_init*='ones', *beta_init*='zeros', *dtype*=mstype.float32)

该层在五维输入（带有额外通道维度的 mini-batch 三维输入）上应用实例归一化。详见论文 [Instance Normalization: The Missing Ingredient for Fast Stylization](#)。使用 mini-batch 数据和学习参数进行训练，参数见如下公式。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中 γ 和 β 是可学习的参数向量，如果 *affine* 为 True，则大小为 *num_features*。通过偏置估计函数计算标准偏差。此层使用从训练和验证模式的输入数据计算得到的实例数据。

InstanceNorm3d 和 BatchNorm3d 类似。不同之处在于 InstanceNorm3d 应用于 RGB 图像等通道数据的每个通道，而 BatchNorm3d 通常应用于批处理。

说明：需要注意的是，更新滑动平均和滑动方差的公式为 $\hat{x}_{\text{new}} = (1 - \text{momentum}) \times x_t + \text{momentum} \times \hat{x}$ ，其中 $\hat{x}$ 是估计的统计量， x_t 是新的观察值。

参数：

- **num_features** (int) - 通道数量，输入 Tensor shape (*N, C, D, H, W*) 中的 *C*。
- **eps** (float) - 添加到分母中的值，以确保数值稳定。默认值：1e-5。
- **momentum** (float) - 动态均值和动态方差所使用的动量。默认值：0.1。
- **affine** (bool) - bool 类型。设置为 True 时，可以学习 *gamma* 和 *beta* 参数。默认值：True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number]) - *gamma* 参数的初始化方法。str 的值引用自函数 *initializer*，包括 'zeros'、'ones' 等。使用 Tensor 作为初始化参数时，shape 必须为 (*C*)。默认值：'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number]) - *beta* 参数的初始化方法。str 的值引用自函数 *initializer*，包括 'zeros'、'ones' 等。使用 Tensor 作为初始化参数时，shape 必须为 (*C*)。默认值：'zeros'。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **x** (Tensor) - shape 为 (*N, C, D, H, W*) 的 Tensor。数据类型为 float16 或 float32。

输出：

Tensor，归一化，缩放，偏移后的 Tensor，其 shape 为 (*N, C, D, H, W*)。类型和 shape 与 *x* 相同。

异常：

- **TypeError** - *num_features* 不是整数。
- **TypeError** - *eps* 的类型不是 float。
- **TypeError** - *momentum* 的类型不是 float。
- **TypeError** - *affine* 不是 bool。
- **TypeError** - *gamma_init* / *beta_init* 的类型不相同，或者初始化的元素类型不是 float32。
- **ValueError** - *num_features* 小于 1。
- **ValueError** - *momentum* 不在范围 [0, 1] 内。

- **ValueError** - *gamma_init* / *beta_init* 的 shape 不为 (C)。
- **KeyError** - *gamma_init* / *beta_init* 中的任何一个不是 str，并且不存在继承自 *Initializer* 的同义类。

支持平台：

GPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> net = ms.nn.InstanceNorm3d(3)
>>> x = ms.Tensor(np.ones([2, 3, 5, 2, 2]), ms.float32)
>>> output = net(x)
>>> print(output.shape)
(2, 3, 5, 2, 2)
```

3.11.8 mindspore.nn.LayerNorm

class mindspore.nn.**LayerNorm** (*normalized_shape*, *begin_norm_axis=-1*, *begin_params_axis=-1*, *gamma_init='ones'*, *beta_init='zeros'*, *epsilon=1e-7*, *dtype=mstype.float32*)

在 mini-batch 输入上应用层归一化 (Layer Normalization)。

层归一化在递归神经网络中被广泛的应用。适用单个训练用例的 mini-batch 输入上应用归一化，详见论文 [Layer Normalization](#)。

与批归一化 (Batch Normalization) 不同，层归一化在训练和测试时执行完全相同的计算。应用于所有通道和像素，即使 *batch_size*=1 也适用。其中 γ 是通过训练学习出的 scale 值， β 是通过训练学习出的 shift 值。公式如下：

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

参数：

- **normalized_shape** (Union(tuple[int], list[int])) - 沿轴 *begin_norm_axis* ... *R-1* 执行归一化。其中 *R* 为输入 *x* 的维度大小。
- **begin_norm_axis** (int) - 归一化开始计算的轴，取值范围是 [-1, *R*)。默认值：-1。
- **begin_params_axis** (int) - 指定输入参数 (γ, β) 需进行层归一化的开始轴，取值范围是 [-1, *R*)。默认值：-1。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number]) - γ 参数的初始化方法。str 的值引用自函数 *initializer*，包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值：'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number]) - β 参数的初始化方法。str 的值引用自函数 *initializer*，包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值：'zeros'。
- **epsilon** (float) - 添加到分母中的值 (ϵ)，以确保数值稳定。默认值：1e-7。
- **dtype** (*mindspore.dtype*) - Parameters 的 dtype。默认值：mstype.float32。

输入：

- **x** (Tensor) - *x* 的 shape 为 $(x_1, x_2, \dots, x_R)$ ，*input_shape*[*begin_norm_axis*:] 等于 *normalized_shape*。

输出：

Tensor，归一化后的 Tensor，shape 和数据类型与 *x* 相同。

异常：

- **TypeError** - *normalized_shape* 既不是 list 也不是 tuple。
- **TypeError** - *begin_norm_axis* 或 *begin_params_axis* 不是 int。
- **TypeError** - *epsilon* 不是 float。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> x = ms.Tensor(np.ones([20, 5, 10, 10]), ms.float32)
>>> shape1 = x.shape[1:]
>>> m = ms.nn.LayerNorm(shape1, begin_norm_axis=1, begin_params_axis=1)
>>> output = m(x).shape
>>> print(output)
(20, 5, 10, 10)
```

3.11.9 mindspore.nn.SyncBatchNorm

class mindspore.nn.SyncBatchNorm(*num_features*, *eps*=1e-5, *momentum*=0.9, *affine*=True, *gamma_init*='ones', *beta_init*='zeros', *moving_mean_init*='zeros', *moving_var_init*='ones', *use_batch_statistics*=None, *process_groups*=None, *dtype*=mstype.float32)

在 N 维输入上进行跨设备同步批归一化 (Batch Normalization, BN)。

同步 BN 是跨设备的。BN 的实现仅对每个设备中的数据进行归一化。同步 BN 将归一化组内的输入。描述见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。使用 mini-batch 数据和和学习参数进行训练，参数见如下公式。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

说明：目前，SyncBatchNorm 仅支持二维和四维输入。 γ 是通过训练学习出的 scale 值， β 是通过训练学习出的 shift 值。

参数：

- **num_features** (int) - 指定输入 Tensor 的通道数量，输入 Tensor 的 size 为 (N, C, H, W) 。
- **eps** (float, 可选) - ϵ 添加到分母中的值，以确保数值稳定。默认值：1e-5。
- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。默认值：0.9。
- **affine** (bool, 可选) - bool 类型。设置为 True 时， γ 和 β 为可学习参数。设置为 False 时， γ 和 β 为不可学习参数。默认值：True。
- **gamma_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - γ 参数的初始化方法。str 的值引用自函数 *mindspore.common.initializer*，包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值：'ones'。
- **beta_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - β 参数的初始化方法。str 的值引用自函数 *mindspore.common.initializer*，包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值：'zeros'。

- **moving_mean_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 动态平均值的初始化方法。str 的值引用自函数 `mindspore.common.initializer` , 包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值: 'zeros'。
- **moving_var_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 动态方差的初始化方法。str 的值引用自函数 `mindspore.common.initializer` , 包括 'zeros'、'ones'、'xavier_uniform'、'he_uniform' 等。默认值: 'ones'。
- **use_batch_statistics** (bool, 可选) - 如果为 True , 则使用当前批次数据的平均值和方差值。如果为 False , 则使用指定的平均值和方差值。如果为 None , 则训练过程将使用当前批次数据的均值和方差, 并跟踪动态均值和动态方差, 验证过程将使用动态均值和动态方差。默认值: None。
- **process_groups** (list, 可选) - 将设备划分为不同的同步组的列表, 包含 N 个列表。每个列表都包含需要在同一组中同步的 rank ID, 其数据类型为整数且数值范围必须为 [0, rank_size) 并且各不相同。如果为 None , 表示跨所有设备同步。默认值: None。
- **dtype** (`mindspore.dtype`, 可选) - Parameters 的 dtype。默认值: `mstype.float32`。

输入:

- **x** (Tensor) - shape 为 $(N, C_{in}, H_{in}, W_{in})$ 的 Tensor。

输出:

Tensor, 归一化后的 Tensor, shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。

异常:

- **TypeError** - `num_features` 不是 int。
- **TypeError** - `eps` 不是 float。
- **TypeError** - `process_groups` 不是 list。
- **ValueError** - `num_features` 小于 1。
- **ValueError** - `momentum` 不在范围 [0, 1] 内。
- **ValueError** - `process_groups` 中的 rank ID 不在 [0, rank_size) 范围内。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在多卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> ms.reset_auto_parallel_context()
>>> ms.set_auto_parallel_context(parallel_mode=ms.ParallelMode.DATA_PARALLEL)
>>> sync_bn_op = ms.nn.SyncBatchNorm(num_features=3, process_groups=[[0, 1], [2, 3]])
>>> x = ms.Tensor(np.ones([1, 3, 2, 2]), ms.float32)
```

(下页继续)

(续上页)

```
>>> output = sync_bn_op(x)
>>> print(output)
[[[ 0.999995 0.999995 ]
 [ 0.999995 0.999995 ]]
 [[ 0.999995 0.999995 ]
 [ 0.999995 0.999995 ]]
 [[ 0.999995 0.999995 ]
 [ 0.999995 0.999995 ]]]]
```

3.12 池化层

接口名	概述	支持平台
<code>mindspore.nn.AdaptiveAvgPool1d</code>	在一个输入 Tensor 上应用 1D 自适应平均池化运算，可视为组成一个 1D 输入平面。	Ascend GPU CPU
<code>mindspore.nn.AdaptiveAvgPool2d</code>	对输入 Tensor ，提供二维的自适应平均池化操作。	Ascend GPU CPU
<code>mindspore.nn.AdaptiveAvgPool3d</code>	对输入 Tensor ，提供三维的自适应平均池化操作。	Ascend GPU CPU
<code>mindspore.nn.AdaptiveMaxPool1d</code>	在一个输入 Tensor 上应用 1D 自适应最大池化运算，可被视为组成一个 1D 输入平面。	Ascend GPU CPU
<code>mindspore.nn.AdaptiveMaxPool2d</code>	对输入 Tensor ，提供二维自适应最大池化操作。	Ascend GPU CPU
<code>mindspore.nn.AdaptiveMaxPool3d</code>	对输入 Tensor 执行三维自适应最大池化操作。	GPU CPU
<code>mindspore.nn.AvgPool1d</code>	在一个输入 Tensor 上应用 1D 平均池化运算，可被视为组成一个 1D 输入平面。	Ascend GPU CPU
<code>mindspore.nn.AvgPool2d</code>	在输入 Tensor 上应用 2D 平均池化运算，可视为二维输入平面的组合。	Ascend GPU CPU
<code>mindspore.nn.AvgPool3d</code>	在一个输入 Tensor 上应用 3D 平均池化运算，输入 Tensor 可看成是由一系列 3D 平面组成的。	Ascend GPU CPU
<code>mindspore.nn.FractionalMaxPool3d</code>	在输入上应用三维分数最大池化。	GPU CPU
<code>mindspore.nn.LPPool1d</code>	在一个输入 Tensor 上应用 1D LP 池化运算，可被视为组成一个 1D 输入平面。	Ascend GPU CPU
<code>mindspore.nn.LPPool2d</code>	在一个输入 Tensor 上应用 2D LP 池化运算，可被视为组成一个 2D 输入平面。	Ascend GPU CPU
<code>mindspore.nn.MaxPool1d</code>	在一个输入 Tensor 上应用 1D 最大池化运算，该 Tensor 可被视为一维平面的组合。	Ascend GPU CPU
<code>mindspore.nn.MaxPool2d</code>	在一个输入 Tensor 上应用 2D 最大池化运算，可被视为组成一个 2D 平面。	Ascend GPU CPU
<code>mindspore.nn.MaxPool3d</code>	在一个输入 Tensor 上应用 3D 最大池化运算，输入 Tensor 可看成是由一系列 3D 平面组成的。	Ascend GPU CPU
<code>mindspore.nn.MaxUnpool1d</code>	计算 <code>mindspore.nn.MaxPool1d</code> 的逆过程。	GPU CPU
<code>mindspore.nn.MaxUnpool2d</code>	<code>mindspore.nn.MaxPool2d</code> 的逆过程。	GPU CPU
<code>mindspore.nn.MaxUnpool3d</code>	<code>mindspore.nn.MaxPool3d</code> 的逆过程。	GPU CPU

3.12.1 mindspore.nn.AdaptiveAvgPool1d

class mindspore.nn.AdaptiveAvgPool1d(output_size)

在一个输入 Tensor 上应用 1D 自适应平均池化运算，可视为组成一个 1D 输入平面。

通常，输入的 shape 为 (N_{in}, C_{in}, L_{in}) ，AdaptiveAvgPool1d 在 L_{in} 维度上计算区域平均值。输出的 shape 为 $(N_{in}, C_{in}, L_{out})$ ，其中， L_{out} 为 output_size。

说明： L_{in} 必须能被 output_size 整除。

参数：

- **output_size** (int) - 目标输出大小 L_{out} 。

输入：

- **input** (Tensor) - shape 为 (N, C_{in}, L_{in}) 的 Tensor，数据类型为 float16 或 float32。

输出：

Tensor，其 shape 为 (N, C_{in}, L_{out}) ，数据类型与 input 相同。

异常：

- **TypeError** - output_size 不是 int。
- **TypeError** - input 不是 float16 或 float32。
- **ValueError** - output_size 小于 1。
- **ValueError** - input 的 shape 长度不等于 3。
- **ValueError** - input 的最后一个维度小于 output_size。
- **ValueError** - input 的最后一个维度不能被 output_size 整除。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> pool = ms.nn.AdaptiveAvgPool1d(output_size=2)
>>> input = ms.Tensor(np.random.randint(0, 10, [1, 3, 6]), ms.float32)
>>> output = pool(input)
>>> result = output.shape
>>> print(result)
(1, 3, 2)
```

3.12.2 mindspore.nn.AdaptiveAvgPool2d

class mindspore.nn.AdaptiveAvgPool2d(*output_size*)

对输入 Tensor，提供二维的自适应平均池化操作。也就是说，对于输入任何尺寸，指定输出的尺寸都为 $H * W$ 。但是输入和输出特征的数目不会变化。

输入和输出数据格式可以是“NCHW”和“CHW”。N 是批处理大小，C 是通道数，H 是特征高度，W 是特征宽度。运算如下：

$$\begin{aligned} h_{start} &= \text{floor}(i * H_{in} / H_{out}) \\ h_{end} &= \text{ceil}((i + 1) * H_{in} / H_{out}) \\ w_{start} &= \text{floor}(j * W_{in} / W_{out}) \\ w_{end} &= \text{ceil}((j + 1) * W_{in} / W_{out}) \\ \text{Output}(i, j) &= \frac{\sum \text{Input}[h_{start}:h_{end}, w_{start}:w_{end}]}{(h_{end} - h_{start}) * (w_{end} - w_{start})} \end{aligned}$$

参数：

- **output_size** (Union[int, tuple]) - 输出特征图的尺寸为 $H * W$ 。可以是 int 类型的 H 和 W 组成的 tuple，也可以为一个 int 值，代表相同 H 和 W，或 None，如果是 None，则意味着输出大小与输入相同。

输入：

- **input** (Tensor) - AdaptiveAvgPool2d 的输入，为三维或四维的 Tensor，数据类型为 float16、float32 或者 float64。

输出：

Tensor，输出 shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。

异常：

- **ValueError** - 如果 *output_size* 是 tuple，并且 *output_size* 的长度不是 2。
- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *input* 的数据类型不是 float16、float32 或者 float64。
- **ValueError** - 如果 *input* 的维度小于或等于 *output_size* 的维度。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> pool = ms.nn.AdaptiveAvgPool2d(2)
>>> input_x = ms.Tensor(np.array([[ [1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                               [ [1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                               [ [1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]]), ms.
↳float32)
>>> output = pool(input_x)
>>> result = output.shape
>>> print(result)
(3, 2, 2)
```


3.12.3 mindspore.nn.AdaptiveAvgPool3d

class mindspore.nn.AdaptiveAvgPool3d(output_size)

对输入 Tensor，提供三维的自适应平均池化操作。也就是说对于输入任何尺寸，指定输出的尺寸都为 (D, H, W) 。但是输入和输出特征的数目不会变化。

假设输入 *input* 最后三维大小分别为 (inD, inH, inW) ，则输出的最后三维大小分别为 $(outD, outH, outW)$ 。运算如下：

$$\begin{aligned} \forall \quad od \in [0, outD - 1], oh \in [0, outH - 1], ow \in [0, outW - 1] \\ output[od, oh, ow] = \\ \quad mean(input[istartD : iendD + 1, istartH : iendH + 1, istartW : iendW + 1]) \\ \text{where,} \\ istartD = \left\lceil \frac{od * inD}{outD} \right\rceil \\ iendD = \left\lfloor \frac{(od+1) * inD}{outD} \right\rfloor \\ istartH = \left\lceil \frac{oh * inH}{outH} \right\rceil \\ iendH = \left\lfloor \frac{(oh+1) * inH}{outH} \right\rfloor \\ istartW = \left\lceil \frac{ow * inW}{outW} \right\rceil \\ iendW = \left\lfloor \frac{(ow+1) * inW}{outW} \right\rfloor \end{aligned}$$

参数：

- **output_size** (Union[int, tuple]) - 指定输出特征图的尺寸，可以是 tuple (D, H, W) ，也可以是一个 int 值 D 来表示输出尺寸为 (D, D, D) 。 D, H 和 W 可以是 int 值或者 None，其中 None 表示输出大小与对应的输入大小相同。

输入：

- **input** (Tensor) - AdaptiveAvgPool3d 的输入，是 4D 或者 5D 的 Tensor。数据类型：float16，float32 或者 float64。

输出：

Tensor，与输入 *input* 的数据类型相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的数据类型不是 float16、float32 或者 float64。
- **ValueError** - *input* 维度不是 4D 或者 5D。
- **ValueError** - *output_size* 的值不是正数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> # case 1: output_size=(3, 3, 4)
>>> output_size=(3, 3, 4)
>>> input_x_val = np.random.randn(4, 3, 5, 6, 7)
>>> input_x = ms.Tensor(input_x_val, ms.float32)
>>> net = ms.nn.AdaptiveAvgPool3d(output_size)
>>> output = net(input_x)
>>> print(output.shape)
(4, 3, 3, 3, 4)
```

(下页继续)

(续上页)

```

>>> # case 2: output_size=4
>>> output_size=5
>>> input_x_val = np.random.randn(2, 3, 8, 6, 12)
>>> input_x = ms.Tensor(input_x_val, ms.float32)
>>> net = ms.nn.AdaptiveAvgPool3d(output_size)
>>> output = net(input_x)
>>> print(output.shape)
(2, 3, 5, 5, 5)
>>> # case 3: output_size=(None, 4, 5)
>>> output_size=(None, 4, 5)
>>> input_x_val = np.random.randn(4, 1, 9, 10, 8)
>>> input_x = ms.Tensor(input_x_val, ms.float32)
>>> net = ms.nn.AdaptiveAvgPool3d(output_size)
>>> output = net(input_x)
>>> print(output.shape)
(4, 1, 9, 4, 5)

```

3.12.4 mindspore.nn.AdaptiveMaxPool1d

class mindspore.nn.AdaptiveMaxPool1d(output_size)

在一个输入 Tensor 上应用 1D 自适应最大池化运算，可被视为组成一个 1D 输入平面。

通常，输入的 shape 为 (N_{in}, C_{in}, L_{in}) ，AdaptiveMaxPool1d 在 L_{in} 维度上计算区域最大值。输出的 shape 为 $(N_{in}, C_{in}, L_{out})$ ，其中， L_{out} 为 output_size。

说明： L_{in} 必须能被 output_size 整除。

参数：

- **output_size** (int) - 目标输出大小 L_{out} 。

输入：

- **x** (Tensor) - shape 为 (N, C_{in}, L_{in}) 的 Tensor，数据类型为 float16 或 float32。

输出：

Tensor，其 shape 为 (N, C_{in}, L_{out}) ，数据类型与 x 相同。

异常：

- **TypeError** - x 不是 float16 或 float32。
- **TypeError** - output_size 不是 int。
- **ValueError** - output_size 小于 1。
- **ValueError** - x 的最后一个维度小于 output_size。
- **ValueError** - x 的最后一个维度不能被 output_size 整除。
- **ValueError** - x 的 shape 长度不等于 3。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> pool = ms.nn.AdaptiveMaxPool1d(output_size=3)
>>> x = ms.Tensor(np.random.randint(0, 10, [1, 3, 6]), ms.float32)
>>> output = pool(x)
>>> result = output.shape
>>> print(result)
(1, 3, 3)
```

3.12.5 mindspore.nn.AdaptiveMaxPool2d

class mindspore.nn.AdaptiveMaxPool2d(*output_size, return_indices=False*)

对输入 Tensor，提供二维自适应最大池化操作。对于输入任何格式，指定输出的格式都为 H * W。但是输入和输出特征的数目不会变化。

输入和输出数据格式可以是“NCHW”和“CHW”。N 是批处理大小，C 是通道数，H 是特征高度，W 是特征宽度。运算如下：

$$\begin{aligned} h_{start} &= \text{floor}(i * H_{in}/H_{out}) \\ h_{end} &= \text{ceil}((i + 1) * H_{in}/H_{out}) \\ w_{start} &= \text{floor}(j * W_{in}/W_{out}) \\ w_{end} &= \text{ceil}((j + 1) * W_{in}/W_{out}) \\ \text{Output}(i, j) &= \max \text{Input}[h_{start} : h_{end}, w_{start} : w_{end}] \end{aligned}$$

说明：KBK 模式下，*output_size* 不支持 mutable 类型。

参数：

- **output_size** (Union[int, tuple]) - 输出特征图的 size。*output_size* 可以为二元 tuple 表示 (H, W)。或者是单个 int 表示 (H, H)。H 和 W 可以是 int 或 None，如果是 None，则意味着输出的 size 与输入相同。
- **return_indices** (bool, 可选) - 是否输出最大值的索引。如果为 True，输出最大值的索引。默认值: False。

输入：

- **input** (Tensor) - AdaptiveMaxPool2d 的输入，为三维或四位的 Tensor，数据类型为 float16、float32 或者 float64。

输出：

Tensor，数据类型与 *input* 相同。输出的 shape 为 *input_shape*[: len(*input_shape*) - len(*out_shape*)] + *out_shape*。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 中的数据不是 float16、float32、float64。
- **TypeError** - *output_size* 不是 int 或者 tuple。
- **TypeError** - *return_indices* 不是 bool。
- **ValueError** - *output_size* 是 tuple，但大小不是 2。
- **ValueError** - *input* 的维度不是 CHW 或者 NCHW。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> # case 1: output_size=(None, 2)
>>> input = ms.Tensor(np.array([[[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                               [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                               [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]])), ms.float32)
>>> adaptive_max_pool_2d = ms.nn.AdaptiveMaxPool2d((None, 2))
>>> output = adaptive_max_pool_2d(input)
>>> print(output)
[[[2. 3.]
  [5. 6.]
  [8. 9.]]
 [2. 3.]
  [5. 6.]
  [8. 9.]]
 [2. 3.]
  [5. 6.]
  [8. 9.]]]
>>> # case 2: output_size=2
>>> adaptive_max_pool_2d = ms.nn.AdaptiveMaxPool2d(2)
>>> output = adaptive_max_pool_2d(input)
>>> print(output)
[[[5. 6.]
  [8. 9.]]
 [5. 6.]
  [8. 9.]]
 [5. 6.]
  [8. 9.]]]
>>> # case 3: output_size=(1, 2)
>>> adaptive_max_pool_2d = ms.nn.AdaptiveMaxPool2d((1, 2))
>>> output = adaptive_max_pool_2d(input)
>>> print(output)
[[[8. 9.]]
 [8. 9.]]
 [8. 9.]]]
```

3.12.6 mindspore.nn.AdaptiveMaxPool3d

class mindspore.nn.AdaptiveMaxPool3d(output_size, return_indices=False)

对输入 Tensor 执行三维自适应最大池化操作。对于任何输入尺寸，输出的 size 为 (D, H, W) 。

参数:

- **output_size** (Union[int, tuple]) - 指定输出的 size。可以用一个正整数统一表示输出的深度、高度和宽度，或者用一个正整数三元组来分别表示输出的深度、高度和宽度。如果是 None 则表示对应维度输出和输入相同。
- **return_indices** (bool, 可选) - 如果 return_indices 为 True，将会输出最大值对应的索引，否则不输出索引。默认值: False。

输入:

- **input** (Tensor) - shape 为 (C, D, H, W) 或 (N, C, D, H, W) 的 Tensor。

输出:

- **y** (Tensor) - Tensor，与输入 input 的数据类型和维度相同。

- **argmax** (Tensor) - Tensor，最大值对应的索引，数据类型为 int32，并与 y 的 shape 相同。仅当 *return_indices* 为 True 的时候才返回该值。

异常：

- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *input* 的维度不是 4D 或者 5D。
- **TypeError** - *input* 的数据类型不是 int、uint 或 float。
- **ValueError** - *output_size* 不是一个 int 整数或者 shape 为 (3,) 的 tuple。

支持平台：

GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> input = ms.Tensor(np.arange(0,36).reshape((1, 3, 3, 4)).astype(np.float32))
>>> output_size = (1, 1, 2)
>>> net = ms.nn.AdaptiveMaxPool3d(output_size, True)
>>> output = net(input)
>>> print(output[0].asnumpy())
[[[33. 35.]]]
>>> print(output[1].asnumpy())
[[[33 35]]]
```

3.12.7 mindspore.nn.AvgPool1d

class mindspore.nn.AvgPool1d(*kernel_size=1, stride=1, pad_mode='valid', padding=0, ceil_mode=False, count_include_pad=True*)

在一个输入 Tensor 上应用 1D 平均池化运算，可被视为组成一个 1D 输入平面。

通常，输入的 shape 为 (N_{in}, C_{in}, L_{in}) ，AvgPool1d 在 (L_{in}) 维度上输出区域平均值。给定 *kernel_size* 为 l_{ker} ，*stride* 为 s_0 ，公式定义如下：

$$\text{output}(N_i, C_j, l) = \frac{1}{l_{ker}} \sum_{n=0}^{l_{ker}-1} \text{input}(N_i, C_j, s_0 \times l + n)$$

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **kernel_size** (int, 可选) - 指定池化核尺寸大小，数据类型为整型。默认值：1。
- **stride** (int, 可选) - 池化操作的移动步长，数据类型为整型。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："valid"。
 - "same": 在输入的两端填充，使得当 *stride* 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在右端。如果设置了此模式，*padding* 必须为 0。
 - "valid": 不对输入进行填充，返回输出可能的最大长度，如果不能构成一个完整 *stride*，那么额外的像素将被丢弃。如果设置了此模式，*padding* 必须为 0。

- "pad": 对输入填充指定的量。在这种模式下, 填充的量由 *padding* 参数指定。如果设置此模式, *padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int]), 可选) - 池化填充值, 只有 *pad* 模式才能设置为非 0。默认值: 0。
padding 只能是一个整数或者包含一个整数的 tuple/list, 设定后, 则会在输入的左边和右边填充 *padding* 次或者 *padding[0]* 次。
- **ceil_mode** (bool, 可选) - 若为 True, 使用 ceil 来计算输出 shape。若为 False, 使用 floor 来计算输出 shape。
默认值: False。
- **count_include_pad** (bool, 可选) - 如果为 True, 平均计算将包括零填充。默认值: True。

输入:

- **x** (Tensor) - shape 为 (N, C_{in}, L_{in}) 或 (C_{in}, L_{in}) 的 Tensor。

输出:

shape 为 (N, C_{out}, L_{out}) 或 (C_{out}, L_{out}) 的 Tensor。

其中, 如果 *pad_mode* 为 *pad* 模式时, 输出的 shape 计算公式如下:

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times padding - kernel\_size}{stride} + 1 \right\rfloor$$

异常:

- **TypeError** - *kernel_size* 或 *stride* 不是 int。
- **ValueError** - *pad_mode* 既不是 "valid", 也不是 "same" 或者 "pad", 不区分大小写。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *padding* 为 tuple/list 时长度不为 1。
- **ValueError** - *x* 的 shape 长度不等于 2 或 3。
- **ValueError** - *pad_mode* 不为 "pad" 的时候 *padding* 为非 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> pool = ms.nn.AvgPool1d(kernel_size=6, stride=1)
>>> x = ms.Tensor(np.random.randint(0, 10, [1, 3, 6]), ms.float32)
>>> output = pool(x)
>>> result = output.shape
>>> print(result)
(1, 3, 1)
>>> pool2 = ms.nn.AvgPool1d(4, stride=1, ceil_mode=True, pad_mode="pad", padding=2)
>>> x1 = ms.ops.randn(6, 6, 8)
>>> output = pool2(x1)
>>> print(output.shape)
(6, 6, 9)
```

3.12.8 mindspore.nn.AvgPool2d

class mindspore.nn.AvgPool2d(kernel_size=1, stride=1, pad_mode='valid', padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None, data_format='NCHW')

在输入 Tensor 上应用 2D 平均池化运算，可视为二维输入平面的组合。

通常，输入的 shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$ ，AvgPool2d 的输出为 (H_{in}, W_{in}) 维度的区域平均值。给定 kernel_size 为 $ks = (h_{ker}, w_{ker})$ 和 stride $s = (s_0, s_1)$ ，公式定义如下：

$$\text{output}(N_i, C_j, h, w) = \frac{1}{h_{ker} * w_{ker}} \sum_{m=0}^{h_{ker}-1} \sum_{n=0}^{w_{ker}-1} \text{input}(N_i, C_j, s_0 \times h + m, s_1 \times w + n)$$

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **kernel_size** (Union[int, tuple[int]]) - 指定池化核尺寸大小。如果为整数或单元素 tuple，则代表池化核的高和宽。如果为 tuple 且长度不为 1，其值必须包含两个整数值分别表示池化核的高和宽。默认值：1。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，如果为整数或单元素 tuple，则代表池化核的高和宽方向的移动步长。如果为 tuple 且长度不为 1，其值必须包含两个整数值分别表示池化核的高和宽的移动步长。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："valid"。
 - "same": 在输入的四周填充，使得当 stride 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在底部/右侧。如果设置了此模式，padding 必须为 0。
 - "valid": 不对输入进行填充，返回输出可能的最大高度和宽度，不能构成一个完整 stride 的额外的像素将被丢弃。如果设置了此模式，padding 必须为 0。
 - "pad": 对输入填充指定的量。在这种模式下，在输入的高度和宽度方向上填充的量由 padding 参数指定。如果设置此模式，padding 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int])) - 池化填充值，只有 pad 模式才能设置为非 0。默认值：0。padding 只能是一个整数或者包含一个或两个整数的元组，若 padding 为一个整数或者包含一个整数的 tuple/list，则会分别在输入的上下左右四个方向进行 padding 次的填充，若 padding 为一个包含两个整数的 tuple/list，则会在输入的上下进行 padding[0] 次的填充，在输入的左右进行 padding[1] 次的填充。
- **ceil_mode** (bool) - 若为 True，使用 ceil 来计算输出 shape。若为 False，使用 floor 来计算输出 shape。默认值：False。
- **count_include_pad** (bool) - 平均计算是否包括零填充。默认值：True。
- **divisor_override** (int) - 如果被指定为非 0 参数，该参数将会在平均计算中被用作除数，否则将会使用 kernel_size 作为除数，默认值：None。
- **data_format** (str) - 输入数据格式可为 'NHWC' 或 'NCHW'。默认值：'NCHW'。

输入：

- **x** (Tensor) - 输入数据的 shape 为 $(N, C_{in}, H_{in}, W_{in})$ 或 (C_{in}, H_{in}, W_{in}) 的 Tensor。

输出：

输出数据的 shape 为 $(N, C_{out}, H_{out}, W_{out})$ 或 $(C_{out}, H_{out}, W_{out})$ 的 Tensor。

其中, 如果 *pad_mode* 为 *pad* 模式时, 输出的 *shape* 计算公式如下:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[0] - kernel\_size[0]}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[1] - kernel\_size[1]}{stride[1]} + 1 \right\rfloor$$

异常:

- **TypeError** - *kernel_size* 或 *strides* 既不是整数, 也不是 *tuple*。
- **ValueError** - *pad_mode* 既不是“valid”, 也不是“same”或者“pad”, 不区分大小写。
- **ValueError** - *data_format* 既不是‘NCHW’, 也不是‘NHWC’。
- **ValueError** - *data_format* 为 ‘NHWC’ 时, 使用了 *padding*、*ceil_mode*、*count_include_pad* 或者 *divisor_override* 之一, 或 *pad_mode* 为 *pad*。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *padding* 为 *tuple/list* 时长度不为 1 或 2。
- **ValueError** - *x* 的 *shape* 长度不等于 3 或 4。
- **ValueError** - *divisor_override* 小于等于 0。
- **ValueError** - *pad_mode* 不为“pad”的时候 *padding* 为非 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> pool = ms.nn.AvgPool2d(kernel_size=3, stride=1)
>>> x = ms.Tensor(np.random.randint(0, 10, [1, 2, 4, 4]), ms.float32)
>>> output = pool(x)
>>> print(output.shape)
(1, 2, 2, 2)
>>> x = ms.ops.randn(6, 6, 8, 8)
>>> pool2 = ms.nn.AvgPool2d(4, stride=1, pad_mode="pad", padding=2, divisor_override=5)
>>> output2 = pool2(x)
>>> print(output2.shape)
(6, 6, 9, 9)
```

3.12.9 mindspore.nn.AvgPool3d

class mindspore.nn.AvgPool3d(*kernel_size=1, stride=1, pad_mode='valid', padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None*)

在一个输入 Tensor 上应用 3D 平均池化运算, 输入 Tensor 可看成是由一系列 3D 平面组成的。

通常, 输入的 *shape* 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$, AvgPool3D 输出 (D_{in}, H_{in}, W_{in}) 维度的区域平均值。给定 *kernel_size* 为 $ks = (d_{ker}, h_{ker}, w_{ker})$ 和 *stride* 为 $s = (s_0, s_1, s_2)$, 公式如下。

警告: *kernel_size* 在 [1, 255] 的范围内取值。*stride* 在 [1, 63] 的范围内取值。

$$\text{output}(N_i, C_j, d, h, w) = \frac{1}{d_{ker} * h_{ker} * w_{ker}} \sum_{l=0}^{d_{ker}-1} \sum_{m=0}^{h_{ker}-1} \sum_{n=0}^{w_{ker}-1} \text{input}(N_i, C_j, s_0 \times d + l, s_1 \times h + m, s_2 \times w + n)$$

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **kernel_size** (Union[int, tuple[int]], 可选) - 指定池化核尺寸大小。如果为整数或单元素 **tuple**，则同时代表池化核的深度、高度和宽度。如果为 **tuple** 且长度不为 1，其值必须包含三个正整数，分别表示池化核的深度、高度和宽度。默认值：1。
- **stride** (Union[int, tuple[int]], 可选) - 池化操作的移动步长。如果为整数或单元素 **tuple**，则同时代表池化核的深度、高度和宽度方向上的移动步长。如果为 **tuple** 且长度不为 1，其值必须包含三个整数值，分别表示池化核的深度、高度和宽度方向上的移动步长。取值必须为正整数。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："valid"。
 - "same"：在输入的深度、高度和宽度维度进行填充，使得当 *stride* 为 1 时，输入和输出的 **shape** 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在前方/底部/右侧。如果设置了此模式，*padding* 必须为 0。
 - "valid"：不对输入进行填充，返回输出可能的最大深度、高度和宽度，不能构成一个完整 *stride* 的额外的像素将被丢弃。如果设置了此模式，*padding* 必须为 0。
 - "pad"：对输入填充指定的量。在这种模式下，在输入的深度、高度和宽度方向上填充的量由 *padding* 参数指定。如果设置此模式，*padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int]), 可选) - 池化填充值，只有 *pad_mode* 为 "pad" 时才能设置为非 0。默认值：0。只支持以下情况：
 - *padding* 为一个整数或包含一个整数的 **tuple/list**，此情况下分别在输入的前后上下左右六个方向进行 *padding* 次的填充。
 - *padding* 为一个包含三个 **int** 的 **tuple/list**，此情况下在输入的前后进行 *padding*[0] 次的填充，上下进行 *padding*[1] 次的填充，在输入的左右进行 *padding*[2] 次的填充。
- **ceil_mode** (bool, 可选) - 若为 True，使用 **ceil** 来计算输出 **shape**。若为 False，使用 **floor** 来计算输出 **shape**。默认值：False。
- **count_include_pad** (bool, 可选) - 平均计算是否包括零填充。默认值：True。
- **divisor_override** (int, 可选) - 如果被指定为非 0 参数，该参数将会在平均计算中被用作除数，否则将会使用 *kernel_size* 作为除数，默认值：None。

输入：

- **x** (Tensor) - **shape** 为 $(N, C, D_{in}, H_{in}, W_{in})$ 或者 $(C, D_{in}, H_{in}, W_{in})$ 的 Tensor。数据类型为 float16、float32 和 float64。

输出：

shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 或者 $(C, D_{out}, H_{out}, W_{out})$ 的 Tensor。数据类型与 *x* 一致。

其中，如果 *pad_mode* 为 *pad* 模式时，输出的 **shape** 计算公式如下：

$$D_{out} = \left\lfloor \frac{D_{in} + 2 \times \text{padding}[0] - \text{kernel\_size}[0]}{\text{stride}[0]} + 1 \right\rfloor$$

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[1] - kernel\_size[1]}{stride[1]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[2] - kernel\_size[2]}{stride[2]} + 1 \right\rfloor$$

异常：

- **TypeError** - *kernel_size* 既不是整数也不是元组。
- **TypeError** - *stride* 既不是整数也不是元组。
- **TypeError** - *padding* 既不是整数也不是元组/列表。
- **TypeError** - *ceil_mode* 或 *count_include_pad* 不是 bool。
- **TypeError** - *divisor_override* 不是整数。
- **ValueError** - *kernel_size* 或者 *stride* 中的数字不是正数。
- **ValueError** - *kernel_size* 或 *stride* 是一个长度不为 3 的 tuple。
- **ValueError** - *padding* 为一个 tuple/list 时，长度不为 1 或者 3。
- **ValueError** - *padding* 的元素小于 0。
- **ValueError** - *x* 的 shape 长度不等于 4 或 5。
- **ValueError** - *divisor_override* 小于等于 0。
- **ValueError** - *pad_mode* 不为“pad”的时候 *padding* 为非 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> pool = ms.nn.AvgPool3d(kernel_size=3, stride=1)
>>> x = ms.ops.randn(1, 2, 4, 4, 5).astype(ms.float32)
>>> output = pool(x)
>>> print(output.shape)
(1, 2, 2, 2, 3)
>>> x1 = ms.ops.randn(6, 5, 7, 7, 5).astype(ms.float32)
>>> pool2 = ms.nn.AvgPool3d(4, stride=2, pad_mode="pad", padding=(2, 2, 1), divisor_
↳ override=10)
>>> output2 = pool2(x1)
>>> print(output2.shape)
(6, 5, 4, 4, 2)
```

3.12.10 mindspore.nn.FractionalMaxPool3d

class mindspore.nn.FractionalMaxPool3d(*kernel_size*, *output_size=None*, *output_ratio=None*, *return_indices=False*, *_random_samples=None*)

在输入上应用三维分数最大池化。输出 Tensor 的 shape 可以由 *output_size* 和 *output_ratio* 其中之一确定，步长由 *_random_samples* 随机决定。*output_size* 和 *output_ratio* 同时设置，*output_size* 会生效。*output_size* 和 *output_ratio* 不能同时为 None。

分数最大池化的详细描述在 [Fractional MaxPooling by Ben Graham](#)。

输入输出的数据格式可以是“NCDHW”。其中，N 是批次大小，C 是通道数，D 是特征深度，H 是特征高度，W 是特征宽度。

参数：

- **kernel_size** (Union[int, tuple[int]]) - 指定池化核尺寸大小，如果为正整数，则代表池化核的深度，高和宽。如果为 tuple，其值必须包含三个正整数值分别表示池化核的深度，高和宽。取值必须为正整数。
- **output_size** (Union[int, tuple[int]], 可选) - 目标输出大小。如果是正整数，则表示输出目标的深、高和宽。如果是 tuple，其值必须包含三个正整数值分别表示目标输出的深、高和宽。为 None 时，输出大小由 *output_ratio* 决定。默认值：None。
- **output_ratio** (Union[float, tuple[float]], 可选) - 目标输出 shape 与输入 shape 的比率。通过输入 shape 和 *output_ratio* 确定输出 shape。支持数据类型：float16、float32、float64，数值介于 0 到 1 之间。为 None 时，输出大小由 *output_size* 决定。默认值：None。
- **return_indices** (bool, 可选) - 是否返回最大值的索引值。默认值：False。
- **_random_samples** (Tensor, 可选) - 随机步长。支持的数据类型：float16、float32、double。shape 为 (N, C, 3) 或 (1, C, 3) 的 Tensor。数值范围 [0, 1)。默认值：None，_random_samples 的值由区间 [0, 1) 上的均匀分布随机生成。

输入：

- **input** (Tensor) - 四维或五维的 Tensor，支持的数据类型：float16、float32、float64。支持 shape 为 (N, C, D_{in}, H_{in}, W_{in}) 或 (C, D_{in}, H_{in}, W_{in})。

输出：

- **y** (Tensor) - 3D 分数最大池化的输出，是一个 Tensor。数据类型和 input 相同，shape 是 (N, C, D_{out}, H_{out}, W_{out}) 或 (C, D_{out}, H_{out}, W_{out})。其中，(D_{out}, H_{out}, W_{out}) = *output_size* 或 (D_{out}, H_{out}, W_{out}) = *output_ratio* * (D_{in}, H_{in}, W_{in})。
- **argmax** (Tensor) - 仅当 *return_indices* 为 True 时，输出最大池化的索引值。shape 和输出 y 一致。

异常：

- **TypeError** - *input* 不是四维或五维 Tensor。
- **TypeError** - *_random_samples* 不是三维 Tensor。
- **TypeError** - *input* 数据类型不是 float16、float32、double。
- **TypeError** - *_random_samples* 数据类型不是 float16、float32、double。
- **TypeError** - *argmax* 数据类型不是 int32、int64。
- **TypeError** - *_random_samples* 和 *input* 数据类型不相同。
- **ValueError** - *output_size* 不是长度为 3 的元组。
- **ValueError** - *kernal_size* 不是长度为 3 的元组。
- **ValueError** - *output_size* 和 *kernel_size* 不是正数。
- **ValueError** - *output_size* 和 *output_ratio* 同时为 None。
- **ValueError** - *input* 和 *_random_samples* 的第一维度大小不相等。
- **ValueError** - *input* 和 *_random_samples* 第二维度大小不相等。
- **ValueError** - *_random_samples* 第三维度大小不是 3。

支持平台：

GPU CPU

样例：

```

>>> import numpy as np
>>> import mindspore as ms
>>> x = ms.Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]))
...         .reshape([1, 1, 2, 2, 4]), ms.float32)
>>> _random_samples = ms.Tensor(np.array([0.7, 0.7, 0.7]).reshape([1, 1, 3]), ms.float32)
>>> net = ms.nn.FractionalMaxPool3d(kernel_size=(1, 1, 1), output_size=(1, 1, 3),
...                                 _random_samples=_random_samples, return_indices=True)
>>> output, argmax = net(x)
>>> print(output)
[[[[[13. 14. 16.]]]]]
>>> print(argmax)
[[[[[12 13 15]]]]]
>>> net = ms.nn.FractionalMaxPool3d(kernel_size=(1, 1, 1), output_ratio=(0.5, 0.5, 0.5),
...                                 _random_samples=_random_samples, return_indices=True)
>>> output, argmax = net(x)
>>> print(output)
[[[[[13. 16.]]]]]
>>> print(argmax)
[[[[[12 15]]]]]

```

3.12.11 mindspore.nn.LPPool1d

class mindspore.nn.LPPool1d(*norm_type, kernel_size, stride=None, ceil_mode=False*)

在一个输入 Tensor 上应用 1D LP 池化运算，可被视为组成一个 1D 输入平面。

通常，输入的 shape 为 (N_{in}, C_{in}, L_{in}) 或 (C_{in}, L_{in}) ，输出的 shape 为 $(N_{out}, C_{out}, L_{out})$ 或 (C_{out}, L_{out}) ，输出与输入的 shape 一致，公式如下：

$$f(X) = \sqrt[p]{\sum_{x \in X} x^p}$$

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **norm_type** (Union[int, float]) - 标准化类型，代表公式里的 p ，不能为 0，
 - 如果 $p = 1$ ，得到的结果为池化核内元素之和（与平均池化成比例）；
 - 如果 $p = \infty$ ，得到的结果为最大池化的结果。
- **kernel_size** (int) - 池化核的尺寸大小。
- **stride** (int) - 池化操作的移动步长，数据类型为整型。如果值为 None，则使用默认值 *kernel_size*。默认值：None。
- **ceil_mode** (bool) - 若为 True，使用 ceil 来计算输出 shape。若为 False，使用 floor 来计算输出 shape。默认值：False。

输入：

- **x** (Tensor) - shape 为 (N_{in}, C_{in}, L_{in}) 或 (C_{in}, L_{in}) 的 Tensor。

输出：

- **output** - LPPool1d 的计算结果，shape 为 $(N_{out}, C_{out}, L_{out})$ 或 (C_{out}, L_{out}) 的 Tensor，与输入 *x* 的类型一致，其中：

$$L_{out} = \left\lfloor \frac{L_{in} - \text{kernel\_size}}{\text{stride}} + 1 \right\rfloor$$

异常:

- **TypeError** - x 不是 Tensor。
- **TypeError** - kernel_size 或 stride 不是 int。
- **TypeError** - ceil_mode 不是 bool。
- **TypeError** - norm_type 不是 float 也不是 int。
- **ValueError** - norm_type 等于 0。
- **ValueError** - kernel_size 或 stride 小于 1。
- **ValueError** - x 的 shape 长度不等于 2 或 3。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> a = ms.Tensor(np.arange(2 * 3 * 4).reshape((2, 3, 4)), dtype=ms.float32)
>>> net = ms.nn.LPPool1d(norm_type=1, kernel_size=3, stride=1)
>>> out = net(a)
>>> print(out)
[[[ 3.  6.]
  [15. 18.]
  [27. 30.]]
 [[39. 42.]
  [51. 54.]
  [63. 66.]]]
```

3.12.12 mindspore.nn.LPPool2d

class mindspore.nn.LPPool2d(*norm_type, kernel_size, stride=None, ceil_mode=False*)

在一个输入 Tensor 上应用 2D LP 池化运算, 可被视为组成一个 2D 输入平面。

通常, 输入的 shape 为 (N, C, H_{in}, W_{in}) , 输出的 shape 为 (N, C, H_{in}, W_{in}) , 输出与输入的 shape 一致, 公式如下:

$$f(X) = \sqrt[p]{\sum_{x \in X} x^p}$$

说明: 该接口暂不支持 Atlas A2 训练系列产品。

参数:

- **norm_type** (Union[int, float]) - 标准化类型, 代表公式里的 p , 不能为 0,
 - 如果 $p = 1$, 得到的结果为池化核内元素之和 (与平均池化成比例);
 - 如果 $p = \infty$, 得到的结果为最大池化的结果。

- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。如果为整数，则代表池化核的高和宽。如果为 tuple，其值必须包含两个整数值分别表示池化核的高和宽。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，如果为整数，则代表 stride 的高和宽。如果为 tuple，其值必须包含两个整数值分别表示 stride 的高和宽。如果值为 None，则使用默认值 *kernel_size*。默认值: None。
- **ceil_mode** (bool) - 若为 True，使用 ceil 模式来计算输出 shape。若为 False，使用 floor 模式来计算输出 shape。默认值: False。

输入:

- **x** (Tensor) - shape 为 (N, C, H_{in}, W_{in}) 的 Tensor。

输出:

- **output** - LPPool2d 的计算结果，shape 为 (N, C, H_{in}, W_{in}) 的 Tensor，与输入 *x* 的类型一致，其中

$$H_{out} = \left\lfloor \frac{H_{in} - \text{kernel\_size}[0]}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} - \text{kernel\_size}[1]}{\text{stride}[1]} + 1 \right\rfloor$$

异常:

- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *kernel_size* 或 *stride* 不是 int 也不是 tuple。
- **TypeError** - *ceil_mode* 不是 bool。
- **TypeError** - *norm_type* 不是 float 也不是 int。
- **ValueError** - *norm_type* 等于 0。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *kernel_size* 或 *stride* 是一个长度不为 2 的 tuple。
- **ValueError** - *x* 的 shape 长度不等于 4。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> a = ms.Tensor(np.arange(2 * 3 * 4 * 5).reshape((2, 3, 4, 5)), dtype=ms.float32)
>>> net = ms.nn.LPPool2d(norm_type=1, kernel_size=3, stride=1)
>>> out = net(a)
>>> print(out)
[[[ [ 54.   63.   72.]
      [ 99.  108.  117.]]
  [ [ 234.  243.  252.]
      [ 279.  288.  297.]]
  [ [ 414.  423.  432.]
      [ 459.  468.  477.]]]
 [[ [ 594.  603.  612.]
      [ 639.  648.  657.]]
  [ [ 774.  783.  792.]
      [ 819.  828.  837.]]]]
```

(下页继续)

(续上页)

```
[[ 954.  963.  972.]
 [ 999. 1008. 1017.]]]]
```

3.12.13 mindspore.nn.MaxPool1d

class mindspore.nn.MaxPool1d (*kernel_size=1, stride=1, pad_mode='valid', padding=0, dilation=1, return_indices=False, ceil_mode=False*)

在一个输入 Tensor 上应用 1D 最大池化运算, 该 Tensor 可被视为一维平面的组合。

通常, 输入的 shape 为 (N_{in}, C_{in}, L_{in}) , MaxPool1d 输出 (L_{in}) 维度区域最大值。给定 *kernel_size* 为 $ks = (l_{ker})$ 和 *stride* 为 $s = (s_0)$, 公式如下:

$$\text{output}(N_i, C_j, l) = \max_{n=0, \dots, l_{ker}-1} \text{input}(N_i, C_j, s_0 \times l + n)$$

参数:

- **kernel_size** (int, 可选) - 指定池化核尺寸大小。默认值: 1。
- **stride** (int, 可选) - 池化操作的移动步长, 数据类型为整型。默认值: 1。
- **pad_mode** (str, 可选) - 指定填充模式, 填充值为 0。可选值为 "same"、"valid" 或 "pad"。默认值: "valid"。
 - "same": 在输入的两端填充, 使得当 *stride* 为 1 时, 输入和输出的 shape 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周; 若为奇数, 多余的填充量将补充在右端。如果设置了此模式, *padding* 必须为 0。
 - "valid": 不对输入进行填充, 返回输出可能的最大长度, 不能构成一个完整 *stride* 的额外像素将被丢弃。如果设置了此模式, *padding* 必须为 0。
 - "pad": 对输入填充指定的量。在这种模式下, 填充的量由 *padding* 参数指定。如果设置此模式, *padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int]), 可选) - 池化填充值。默认值: 0。*padding* 只能是一个整数或者包含一个整数的 tuple/list, 设定后, 则会在输入的左边和右边填充 *padding* 次或者 *padding*[0] 次。
- **dilation** (Union(int, tuple[int]), 可选) - 卷积核中各个元素之间的间隔大小, 用于提升池化操作的感受野。如果为 tuple, 其长度只能为 1。默认值: 1。
- **return_indices** (bool, 可选) - 若为 True, 将会同时返回最大池化的结果和索引。默认值: False。
- **ceil_mode** (bool, 可选) - 若为 True, 使用 ceil 来计算输出 shape; 若为 False, 使用 floor 来计算输出 shape。默认值: False。

输入:

- **x** (Tensor) - shape 为 (N, C_{in}, L_{in}) 或 (C_{in}, L_{in}) 的 Tensor。

输出:

如果 *return_indices* 为 False, 则是 shape 为 (N, C_{out}, L_{out}) 或 (C_{out}, L_{out}) 的 Tensor。数据类型与 *x* 一致。如果 *return_indices* 为 True, 则是一个包含了两个 Tensor 的 Tuple, 表示 maxpool 的计算结果以及生成 max 值的位置。

- **output** (Tensor) - 最大池化结果, shape 为 (N, C_{out}, L_{out}) 或 (C_{out}, L_{out}) 的 Tensor。数据类型与 *x* 一致。
- **argmax** (Tensor) - 最大值对应的索引。数据类型为 int64。

其中, 如果 *pad_mode* 为 *pad* 模式时, 输出的 shape 计算公式如下:

$$L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{padding} - \text{dilation} \times (\text{kernel\_size} - 1) - 1}{\text{stride}} + 1 \right\rfloor$$

异常:

- **TypeError** - *kernel_size* 或 *strides* 不是整数。
- **ValueError** - *pad_mode* 既不是 "valid", 也不是 "same" 或者 "pad", 不区分大小写。
- **ValueError** - *data_format* 既不是 'NCHW' 也不是 'NHWC'。
- **ValueError** - *kernel_size* 或 *strides* 小于 1。
- **ValueError** - *x* 的 shape 长度不等于 2 或 3。
- **ValueError** - 当 *pad_mode* 不为 "pad" 时, *padding*、*dilation*、*return_indices*、*ceil_mode* 参数不为默认值。
- **ValueError** - *padding* 参数为 tuple 或 list 时长度不为 1。
- **ValueError** - *dilation* 参数为 tuple 时长度不为 1。
- **ValueError** - *dilation* 参数不为 int 也不为 tuple。
- **ValueError** - *pad_mode* 不为 "pad" 的时候 *padding* 为非 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> mpool1 = nn.MaxPool1d(kernel_size=3, stride=1)
>>> x = ms.Tensor(np.random.randint(0, 10, [1, 2, 4]), ms.float32)
>>> output = mpool1(x)
>>> result = output.shape
>>> print(result)
(1, 2, 2)
>>> np_x = np.random.randint(0, 10, [5, 3, 4])
>>> x = ms.Tensor(np_x, ms.float32)
>>> mpool2 = nn.MaxPool1d(kernel_size=2, stride=1, pad_mode="pad", padding=1, dilation=1,
↪return_indices=True)
>>> output = mpool2(x)
>>> print(output[0].shape)
(5, 3, 5)
>>> print(output[1].shape)
(5, 3, 5)
```

3.12.14 mindspore.nn.MaxPool2d

class mindspore.nn.**MaxPool2d**(*kernel_size=1, stride=1, pad_mode='valid', padding=0, dilation=1, return_indices=False, ceil_mode=False, data_format='NCHW'*)

在一个输入 Tensor 上应用 2D 最大池化运算, 可被视为组成一个 2D 平面。

通常, 输入的 shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$, MaxPool2d 输出 (H_{in}, W_{in}) 维度区域最大值。给定 *kernel_size* 为 (h_{ker}, w_{ker}) , *stride* 为 (s_0, s_1) , 公式如下。

$$\text{output}(N_i, C_j, h, w) = \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} \text{input}(N_i, C_j, s_0 \times h + m, s_1 \times w + n)$$

参数:

- **kernel_size** (Union[int, tuple[int]]) - 指定池化核尺寸大小, 如果为整数或单元素 tuple, 则代表池化核的高和宽。如果为 tuple 且长度不为 1, 其值必须包含两个整数值分别表示池化核的高和宽。默认值: 1。

- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长, 如果为整数或单元素 tuple, 则代表池化核的高和宽方向的移动步长。如果为 tuple 且长度不为 1, 其值必须包含两个整数值分别表示池化核的高和宽的移动步长。默认值: 1。
- **pad_mode** (str, 可选) - 指定填充模式, 填充值为 0。可选值为 "same", "valid" 或 "pad"。默认值: "valid"。
 - "same": 在输入的四周填充, 使得当 *stride* 为 1 时, 输入和输出的 shape 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在底部/右侧。如果设置了此模式, *padding* 必须为 0。
 - "valid": 不对输入进行填充, 返回输出可能的最大高度和宽度, 如果不能构成一个完整 *stride*, 那么额外的像素将被丢弃。如果设置了此模式, *padding* 必须为 0。
 - "pad": 对输入填充指定的量。在这种模式下, 在输入的高度和宽度方向上填充的量由 *padding* 参数指定。如果设置此模式, *padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int])) - 池化填充值。默认值: 0。padding 只能是一个整数或者包含一个或两个整数的元组, 若 *padding* 为一个整数或者包含一个整数的 tuple/list, 则会分别在输入的上下左右四个方向进行 *padding* 次的填充, 若 *padding* 为一个包含两个整数的 tuple/list, 则会在输入的上下进行 *padding[0]* 次的填充, 在输入的左右进行 *padding[1]* 次的填充。
- **dilation** (Union(int, tuple[int])) - 卷积核中各个元素之间的间隔大小, 用于提升池化操作的感受野。如果为 tuple, 其值必须包含一个或两个整数。默认值: 1。
- **return_indices** (bool) - 若为 True, 将会同时返回最大池化的结果和索引。默认值: False。
- **ceil_mode** (bool) - 若为 True, 使用 ceil 来计算输出 shape。若为 False, 使用 floor 来计算输出 shape。默认值: False。
- **data_format** (str) - 输入数据格式可为 'NHWC' 或 'NCHW'。默认值: 'NCHW'。

输入:

- **x** (Tensor) - 输入数据的 shape 为 $(N, C_{in}, H_{in}, W_{in})$ 或 (C_{in}, H_{in}, W_{in}) 的 Tensor。

输出:

如果 *return_indices* 为 False, 则是 shape 为 (N, C, H_{out}, W_{out}) 或者 $(C_{out}, H_{out}, W_{out})$ 的 Tensor。数据类型与 *x* 一致。如果 *return_indices* 为 True, 则是一个包含了两个 Tensor 的 Tuple, 表示 maxpool 的计算结果以及生成 max 值的位置。

- **output** (Tensor) - 最大池化结果, shape 为 $(N_{out}, C_{out}, H_{out}, W_{out})$ 或者 $(C_{out}, H_{out}, W_{out})$ 的 Tensor。数据类型与 *x* 一致。
- **argmax** (Tensor) - 最大值对应的索引。数据类型为 int64。

其中, 如果 *pad_mode* 为 *pad* 模式时, 输出的 shape 计算公式如下:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 * padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

异常:

- **TypeError** - *kernel_size* 或 *strides* 既不是整数也不是元组。
- **ValueError** - *pad_mode* 既不是 "valid", 也不是 "same" 或者 "pad", 不区分大小写。
- **ValueError** - *data_format* 既不是 'NCHW' 也不是 'NHWC'。
- **ValueError** - *kernel_size* 或 *strides* 小于 1。

- **ValueError** - x 的 shape 长度不等于 3 或 4。
- **ValueError** - 当 `pad_mode` 不为 "pad" 时, `padding`、`dilation`、`return_indices`、`ceil_mode` 参数不为默认值。
- **ValueError** - `padding` 参数为 tuple/list 时长度不为 2。
- **ValueError** - `dilation` 参数为 tuple 时长度不为 2。
- **ValueError** - `dilation` 参数不为 int 也不为 tuple。
- **ValueError** - `pad_mode` 为 "pad" 时, `data_format` 为 'NHWC'。
- **ValueError** - `pad_mode` 不为 "pad" 的时候 `padding` 为非 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> pool = ms.nn.MaxPool2d(kernel_size=3, stride=1)
>>> x = ms.Tensor(np.random.randint(0, 10, [1, 2, 4, 4]), ms.float32)
>>> output = pool(x)
>>> print(output.shape)
(1, 2, 2, 2)
>>> np_x = np.random.randint(0, 10, [5, 3, 4, 5])
>>> x = ms.Tensor(np_x, ms.float32)
>>> pool2 = ms.nn.MaxPool2d(kernel_size=2, stride=1, pad_mode="pad", padding=1, dilation=1,
    ↪return_indices=True)
>>> output = pool2(x)
>>> print(output[0].shape)
(5, 3, 5, 6)
>>> print(output[1].shape)
(5, 3, 5, 6)
```

3.12.15 mindspore.nn.MaxPool3d

class mindspore.nn.**MaxPool3d**(*kernel_size=1, stride=1, pad_mode='valid', padding=0, dilation=1, return_indices=False, ceil_mode=False*)

在一个输入 Tensor 上应用 3D 最大池化运算, 输入 Tensor 可看成是由一系列 3D 平面组成的。

通常, 输入的 shape 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$, MaxPool3d 输出的 shape 为 (D_{in}, H_{in}, W_{in}) 维度区域最大值。给定 `kernel_size` 为 $ks = (d_{ker}, h_{ker}, w_{ker})$ 和 `stride` 为 $s = (s_0, s_1, s_2)$, 公式如下。

$$\text{output}(N_i, C_j, d, h, w) = \max_{l=0, \dots, d_{ker}-1} \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} \text{input}(N_i, C_j, s_0 \times d + l, s_1 \times h + m, s_2 \times w + n)$$

说明: Atlas 训练系列产品暂不支持此接口。

参数:

- **kernel_size** (Union[int, tuple[int]], 可选) - 指定池化核尺寸大小。如果为整数或单元素 tuple, 则该值同时代表池化核的深度、高和宽; 如果为 tuple 且长度不为 1, 其值必须包含三个正整数值, 分别表示池化核的深度、高和宽。取值必须为正整数。默认值: 1。

- **stride** (Union[int, tuple[int]], 可选) - 池化操作的移动步长。如果为整数或单元素 tuple, 则该值同时代表池化核的深度、高和宽方向的移动步长; 如果为 tuple 且长度不为 1, 其值必须包含三个正整数值, 分别表示池化核的深度、高和宽的移动步长。取值必须为正整数。如果值为 None, 则使用默认值 *kernel_size*。默认值: 1。
- **pad_mode** (str, 可选) - 指定填充模式。填充值为 0。可选值为 "same", "valid" 或 "pad"。默认值: "valid"。
 - "same": 在输入的深度、高度和宽度维度进行填充, 使得当 *stride* 为 1 时, 输入和输出的 shape 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周; 若为奇数, 多余的填充量将补充在前方、底部、右侧。如果设置了此模式, *padding* 必须为 0。
 - "valid": 不对输入进行填充, 返回输出可能的最大深度、高度和宽度, 不能构成一个完整 *stride* 的额外的像素将被丢弃。如果设置了此模式, *padding* 必须为 0。
 - "pad": 对输入填充指定的量。在这种模式下, 在输入的深度、高度和宽度方向上填充的量由 *padding* 参数指定。如果设置了此模式, *padding* 必须大于或等于 0。
- **padding** (Union(int, tuple[int], list[int]), 可选) - 池化填充值。默认值: 0。*padding* 只能是一个整数, 或者包含一个或三个整数的 tuple/list。若 *padding* 为一个整数或包含一个整数的 tuple/list, 则会分别在输入的前、后、上、下、左、右, 六个方向进行 *padding* 次的填充; 若 *padding* 为一个包含三个整数的 tuple/list, 则会在输入的前、后进行 *padding[0]* 次的填充, 在输入的上、下进行 *padding[1]* 次的填充, 在输入的左、右进行 *padding[2]* 次的填充。
- **dilation** (Union(int, tuple[int]), 可选) - 卷积核中各个元素之间的间隔大小, 用于提升池化操作的感受野。如果为 tuple, 其值必须包含一个或三个整数。默认值: 1。
- **return_indices** (bool, 可选) - 若为 True, 则返回一个包含两个 Tensor 的 Tuple, 表示池化的计算结果以及生成 max 值的位置。否则, 仅返回池化计算结果。默认值: False。
- **ceil_mode** (bool, 可选) - 若为 True, 则使用 ceil 模式来计算输出 shape; 若为 False, 则使用 floor 模式来计算输出 shape。默认值: False。

输入:

- **x** (Tensor) - shape 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$ 或者 $(C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor。

输出:

如果 *return_indices* 为 False, 则是 shape 为 $(N_{out}, C_{out}, D_{out}, H_{out}, W_{out})$ 或者 $(C_{out}, D_{out}, H_{out}, W_{out})$ 的 Tensor。数据类型与 *x* 一致。如果 *return_indices* 为 True, 则是一个包含了两个 Tensor 的 Tuple, 表示 maxpool 的计算结果, 以及生成 max 值的位置。

- **output** (Tensor) - 最大池化结果。shape 为 $(N_{out}, C_{out}, D_{out}, H_{out}, W_{out})$ 或者 $(C_{out}, D_{out}, H_{out}, W_{out})$ 的 Tensor。数据类型与 *x* 一致。
- **argmax** (Tensor) - 最大值对应的索引。数据类型为 int64。

其中, 如果 *pad_mode* 为 "pad" 模式时, 输出的 shape 计算公式如下:

$$D_{out} = \left\lfloor \frac{D_{in} + 2 \times padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[2] - dilation[2] \times (kernel\_size[2] - 1) - 1}{stride[2]} + 1 \right\rfloor$$

异常:

- **ValueError** - *x* 的 shape 长度不等于 4 或 5。

- **TypeError** - *kernel_size*、*stride*、*padding*、*dilation* 既不是 int 也不是 tuple。
- **ValueError** - *kernel_size* 或者 *stride* 小于 1。
- **ValueError** - *padding* 不为 int 也不是长度为 3 的 tuple。
- **ValueError** - *pad_mode* 不为 "pad" 模式时, *return_indices* 设为了 True 或者 *dilation* 不为 1。
- **ValueError** - *pad_mode* 不为 "pad" 模式时 *padding* 为非 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> import numpy as np
>>> np_x = np.random.randint(0, 10, [5, 3, 4, 6, 7])
>>> x = Tensor(np_x, ms.float32)
>>> pool1 = nn.MaxPool3d(kernel_size=2, stride=1, pad_mode="pad", padding=1, dilation=3,
    ↪return_indices=True)
>>> output = pool1(x)
>>> print(output[0].shape)
(5, 3, 3, 5, 6)
>>> print(output[1].shape)
(5, 3, 3, 5, 6)
>>> pool2 = nn.MaxPool3d(kernel_size=2, stride=1, pad_mode="pad", padding=1, dilation=3,
    ↪return_indices=False)
>>> output2 = pool2(x)
>>> print(output2.shape)
(5, 3, 3, 5, 6)
```

3.12.16 mindspore.nn.MaxUnpool1d

class mindspore.nn.MaxUnpool1d(*kernel_size*, *stride*=None, *padding*=0)

计算 *mindspore.nn.MaxPool1d* 的逆过程。*MaxUnPool1d* 保留最大值并将所有非最大值置 0。支持的输入数据格式为 (N, C, H_{in}) 或 (C, H_{in}) ，输出数据的格式为 (N, C, H_{out}) 或 (C, H_{out}) ，计算公式如下：

$$H_{out} = (H_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0]$$

参数:

- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，若取值为 None，*stride* 值与 *kernel_size* 相同。默认值：None。
- **padding** (Union[int, tuple[int]]) - 填充值。默认值：0。

输入:

- **x** (Tensor) - 待求逆的 Tensor。shape 为 (N, C, H_{in}) 或 (C, H_{in}) 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 *x* 相同。取值范围需满足 $[0, H_{in} - 1]$ 。数据类型必须是 int32 或 int64。

- **output_size** (tuple[int], 可选) - 输出 shape。默认值: None。如果 output_size 为 None, 那么输出 shape 根据 kernel_size、stride 和 padding 计算得出。如果 output_size 不为 None, 那么 output_size 必须满足格式 (N, C, H), (C, H) 或 (H), 取值范围需满足: [(N, C, H_{out} - stride[0]), (N, C, H_{out} + stride[0])]

输出:

shape 为 (N, C, H_{out}) 或 (C, H_{out}) 的 Tensor, 数据类型与输入 x 相同。

异常:

- **TypeError** - x 或 indices 的数据类型不支持。
- **TypeError** - kernel_size, stride 或 padding 既不是整数也不是 tuple。
- **ValueError** - stride, padding 或 kernel_size 的值不是非负的。
- **ValueError** - x 和 indices 的 shape 不一致。
- **ValueError** - x 的长度不是 2 或 3。
- **ValueError** - output_size 的类型不是 tuple。
- **ValueError** - output_size 的长度不为 0、2 或 3。
- **ValueError** - output_size 的取值与根据 kernel_size, stride, padding 计算得到的结果差距太大。

支持平台:

GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> x = ms.Tensor(np.array([[2, 4, 6, 8]]).astype(np.float32))
>>> indices = ms.Tensor(np.array([[1, 3, 5, 7]]).astype(np.int64))
>>> maxunpool1d = ms.nn.MaxUnpool1d(kernel_size=2, stride=2, padding=0)
>>> output = maxunpool1d(x, indices)
>>> print(output.asnumpy())
[[0. 2. 0. 4. 0. 6. 0. 8.]]
```

3.12.17 mindspore.nn.MaxUnpool2d

class mindspore.nn.MaxUnpool2d(kernel_size, stride=None, padding=0)

mindspore.nn.MaxPool2d 的逆过程。*MaxUnpool2d* 在计算过程中, 保留最大值位置的元素, 并将非最大值位置元素设置为 0。支持的输入数据格式为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}), 输出数据的格式为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}), 计算公式如下:

$$H_{out} = (H_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0]$$

$$W_{out} = (W_{in} - 1) \times stride[1] - 2 \times padding[1] + kernel\_size[1]$$

参数:

- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。int 类型表示池化核的长宽相同。tuple 类型中的两个值分别代表池化核的长和宽。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长, int 类型表示长宽方向的移动步长相同。tuple 中的两个值分别代表长宽方向移动的步长。若取值为 None, stride 值与 kernel_size 相同。默认值: None。
- **padding** (Union[int, tuple[int]]) - 填充值。默认值: 0。若为 int 类型, 则长宽方向的填充大小相同, 均为 padding。若为 tuple 类型, 则 tuple 中的两个值分别代表长宽方向填充的大小。

输入：

- **x** (Tensor) - 待求逆的 Tensor。shape 为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 x 相同。取值范围需满足 $[0, H_{in} \times W_{in} - 1]$ 。数据类型必须是 int32 或 int64。
- **output_size** (tuple[int], 可选) - 输出 shape。默认值: None。如果 output_size 为 None, 那么输出 shape 根据 kernel_size、stride 和 padding 计算得出。如果 output_size 不为 None, 那么 output_size 必须满足格式 (N, C, H, W) , (C, H, W) 或 (H, W) , 取值范围需满足: $[(N, C, H_{out} - stride[0], W_{out} - stride[1]), (N, C, H_{out} + stride[0], W_{out} + stride[1])]$ 。

输出：

shape 为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) 的 Tensor, 数据类型与输入 x 相同。

异常：

- **TypeError** - x 或 $indices$ 的数据类型不支持。
- **TypeError** - $kernel\_size$, $stride$ 或 $padding$ 既不是整数也不是 tuple。
- **ValueError** - $stride$, $padding$ 或 $kernel\_size$ 的值不是非负的。
- **ValueError** - x 和 $indices$ 的 shape 不一致。
- **ValueError** - $kernel\_size$, $stride$ 或 $padding$ 为 tuple 时长度不等于 2。
- **ValueError** - x 的长度不为 3 或 4。
- **ValueError** - $output\_size$ 的类型不是 tuple。
- **ValueError** - $output\_size$ 的长度不为 0、3 或 4。
- **ValueError** - $output\_size$ 的取值与根据 $kernel\_size$ 、 $stride$ 、 $padding$ 计算得到的结果差距太大。

支持平台：

GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> x = ms.Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> indices = ms.Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> maxunpool2d = ms.nn.MaxUnpool2d(kernel_size=1, stride=1, padding=0)
>>> output = maxunpool2d(x, indices)
>>> print(output.asnumpy())
[[[0. 1.]
  [8. 9.]]]
```

3.12.18 mindspore.nn.MaxUnpool3d

class mindspore.nn.MaxUnpool3d ($kernel\_size, stride=None, padding=0$)

mindspore.nn.MaxPool3d 的逆过程。*MaxUnpool3d* 在计算逆的过程中, 保留最大值位置的元素, 并将非最大值位置元素设置为 0。支持的输入数据格式为 $(N, C, D_{in}, H_{in}, W_{in})$ 或 $(C, D_{in}, H_{in}, W_{in})$, 输出数据的格式为 $(N, C, D_{out}, H_{out}, W_{out})$ 或 $(C, D_{out}, H_{out}, W_{out})$, 计算公式如下:

$$\begin{aligned} D_{out} &= (D_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0] \\ H_{out} &= (H_{in} - 1) \times stride[1] - 2 \times padding[1] + kernel\_size[1] \\ W_{out} &= (W_{in} - 1) \times stride[2] - 2 \times padding[2] + kernel\_size[2] \end{aligned}$$

参数：

- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。int 类型表示池化核的深度、长和宽相同。tuple 类型中的三个值分别代表池化核的深度、长和宽。
- **stride** (Union[int, tuple[int]]) - 池化操作的移动步长，int 类型表示深度、长和宽方向的移动步长相同。tuple 中的三个值分别代表深度、长和宽方向移动的步长。若取值为 None，stride 值与 kernel_size 相同。默认值：None。
- **padding** (Union[int, tuple[int]]) - 填充值。默认值：0。若为 int 类型，则深度、长和宽方向的填充大小相同，均为 padding。若为 tuple 类型，则 tuple 中的三个值分别代表深度、长和宽方向填充的大小。

输入：

- **x** (Tensor) - 待求逆的 Tensor。shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 或 $(C, D_{in}, H_{in}, W_{in})$ 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 x 相同。取值范围需满足 $[0, D_{in} \times H_{in} \times W_{in} - 1]$ 。数据类型必须是 int32 或 int64。
- **output_size** (tuple[int], 可选) - 输出 shape。默认值：None。如果 output_size 为 None，那么输出 shape 根据 kernel_size、stride 和 padding 计算得出。如果 output_size 不为 None，那么 output_size 必须满足格式 (N, C, D, H, W) ， (C, D, H, W) 或 (D, H, W) ，取值范围需满足： $[(N, C, D_{out}-stride[0], H_{out}-stride[1], W_{out}-stride[2]), (N, C, D_{out}+stride[0], H_{out}+stride[1], W_{out}+stride[2])]$ 。

输出：

shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 或 $(C, D_{out}, H_{out}, W_{out})$ 的 Tensor，数据类型与输入 x 相同。

异常：

- **TypeError** - x 或 indices 的数据类型不支持。
- **TypeError** - kernel_size，stride 或 padding 既不是整数也不是 tuple。
- **ValueError** - stride，kernel_size 或 padding 的值不是非负的。
- **ValueError** - x 和 indices 的 shape 不一致。
- **ValueError** - kernel_size，stride 或 padding 为 tuple 时长度不等于 3。
- **ValueError** - x 的长度不为 4 或 5。
- **ValueError** - output_size 的类型不是 tuple。
- **ValueError** - output_size 的长度不为 0、4 或 5。
- **ValueError** - output_size 的取值与根据 kernel_size、stride、padding 计算得到的结果差距太大。

支持平台：

GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> x = ms.Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> indices = ms.Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> maxunpool3d = ms.nn.MaxUnpool3d(kernel_size=1, stride=1, padding=0)
>>> output = maxunpool3d(x, indices)
>>> print(output.asnumpy())
[[[[[0. 1.]
      [8. 9.]]]]]
```

3.13 填充层

接口名	概述	支持平台
<code>mindspore.nn.Pad</code>	根据 <i>paddings</i> 和 <i>mode</i> 对输入进行填充。	Ascend GPU CPU
<code>mindspore.nn.ConstantPad1d</code>	将给定的常量填充到多维输入数据的最后一维。	Ascend GPU CPU
<code>mindspore.nn.ConstantPad2d</code>	将给定的常量填充到多维输入数据的最后两维。	Ascend GPU CPU
<code>mindspore.nn.ConstantPad3d</code>	将给定的常量填充到多维输入数据的最后三维。	Ascend GPU CPU
<code>mindspore.nn.ReflectionPad1d</code>	根据 <i>padding</i> 对输入 <i>x</i> 进行填充。	Ascend GPU CPU
<code>mindspore.nn.ReflectionPad2d</code>	根据 <i>padding</i> 对输入 <i>x</i> 进行填充。	Ascend GPU CPU
<code>mindspore.nn.ReflectionPad3d</code>	使用反射的方式, 以 <i>input</i> 的边界为对称轴, 对 <i>input</i> 进行填充。	Ascend GPU CPU
<code>mindspore.nn.ReplicationPad1d</code>	根据 <i>padding</i> 对输入 <i>x</i> 的 W 维度上进行填充。	GPU
<code>mindspore.nn.ReplicationPad2d</code>	根据 <i>padding</i> 对输入 <i>x</i> 的 HW 维度上进行填充。	GPU
<code>mindspore.nn.ReplicationPad3d</code>	根据 <i>padding</i> 对输入 <i>x</i> 的 DHW 维度进行填充。	GPU
<code>mindspore.nn.ZeroPad2d</code>	将零填充到多维输入数据的最后两维。	Ascend GPU CPU

3.13.1 mindspore.nn.Pad

class mindspore.nn.Pad (*paddings*, *mode*='CONSTANT')

根据 *paddings* 和 *mode* 对输入进行填充。

参数:

- paddings** (tuple) - 填充大小, 其 shape 为 (N,2), N 是输入数据的维度, 填充的元素为 int 类型。对于 *x* 的第 *D* 个维度, *paddings*[D, 0] 表示要在输入 Tensor 的第 *D* 个维度之前扩展的大小, *paddings*[D, 1] 表示在输入 Tensor 的第 *D* 个维度后面要扩展的大小。每个维度填充后的大小为: *paddings*[D,0]+*input_x.dim_size(D)*+*paddings*[D,1]。

```
# 假设参数和输入如下:
mode = "CONSTANT".
paddings = [[1,1], [2,2]].
x = [[1,2,3], [4,5,6], [7,8,9]].
# `x` 的第一个维度为3, `x` 的第二个维度为3。
# 根据以上公式可得:
# 输出的第一个维度是paddings[0][0] + 3 + paddings[0][1] = 1 + 3 + 1 = 5。
# 输出的第二个维度是paddings[1][0] + 3 + paddings[1][1] = 2 + 3 + 2 = 7。
# 所以最终的输出shape为 (5, 7)
```

- mode** (str) - 指定填充模式。取值为 "CONSTANT" (常数填充), "REFLECT" (反射填充), "SYMMETRIC" (对称填充)。默认值: "CONSTANT"。

输入:

- x** (Tensor) - 输入 Tensor。

输出:

Tensor, 填充后的 Tensor。

- 如果 *mode* 为"CONSTANT", *x* 使用 0 进行填充。例如, *x* 为 [[1,2,3], [4,5,6], [7,8,9]], *paddings* 为 [[1,1], [2,2]], 则输出为 [[0,0,0,0,0,0], [0,0,1,2,3,0,0], [0,0,4,5,6,0,0], [0,0,7,8,9,0,0], [0,0,0,0,0,0,0]]。

- 如果 *mode* 为"REFLECT", *x* 使用对称轴进行对称复制的方式进行填充 (复制时不包括对称轴)。例如 *x* 为 `[[1,2,3], [4,5,6], [7,8,9]]`, *paddings* 为 `[[1,1], [2,2]]`, 则输出为 `[[6,5,4,5,6,5,4], [3,2,1,2,3,2,1], [6,5,4,5,6,5,4], [9,8,7,8,9,8,7], [6,5,4,5,6,5,4]]`。
- 如果 *mode* 为"SYMMETRIC", 此填充方法类似于"REFLECT"。也是根据对称轴填充, 包含对称轴。例如 *x* 为 `[[1,2,3], [4,5,6], [7,8,9]]`, *paddings* 为 `[[1,1], [2,2]]`, 则输出为 `[[2,1,1,2,3,3,2], [2,1,1,2,3,3,2], [5,4,4,5,6,6,5], [8,7,7,8,9,9,8], [8,7,7,8,9,9,8]]`。

异常:

- **TypeError** - *paddings* 不是 tuple。
- **ValueError** - *paddings* 的长度超过 4 或其 *shape* 不是 $(N, 2)$ 。
- **ValueError** - *mode* 不是 'CONSTANT', 'REFLECT' 或 'SYMMETRIC'。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn, ops
>>> import numpy as np
>>> # If `mode` is "CONSTANT"
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.pad = nn.Pad(paddings=((1, 1), (2, 2)), mode="CONSTANT")
...     def construct(self, x):
...         return self.pad(x)
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6]]), mindspore.float32)
>>> pad = Net()
>>> output = pad(x)
>>> print(output)
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 2. 3. 0. 0.]
 [0. 0. 4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
>>> # Another way to call
>>> pad = ops.Pad(paddings=((1, 1), (2, 2)))
>>> # From the above code, we can see following:
>>> # "paddings=((1, 1), (2, 2))",
>>> # paddings[0][0] = 1, indicates a row of values is filled top of the input data in the
↳1st dimension.
>>> # Shown as follows:
>>> # [[0. 0. 0.]
>>> #  [1. 2. 3.]
>>> #  [4. 5. 6.]]
>>> # paddings[0][1] = 1 indicates a row of values is filled below input data in the 1st
↳dimension.
>>> # Shown as follows:
>>> # [[0. 0. 0.]
>>> #  [1. 2. 3.]
>>> #  [4. 5. 6.]
>>> #  [0. 0. 0.]]
>>> # paddings[1][0] = 2, indicates 2 rows of values is filled in front of input data in the
↳2nd dimension.
>>> # Shown as follows:
```

(下页继续)

```

>>> # [[0. 0. 0. 0. 0.]
>>> #  [0. 0. 1. 2. 3.]
>>> #  [0. 0. 4. 5. 6.]
>>> #  [0. 0. 0. 0. 0.]]
>>> # paddings[1][1] = 2, indicates 2 rows of values is filled in front of input data in the
    ↳ 2nd dimension.
>>> # Shown as follows:
>>> # [[0. 0. 0. 0. 0. 0. 0.]
>>> #  [0. 0. 1. 2. 3. 0. 0.]
>>> #  [0. 0. 4. 5. 6. 0. 0.]
>>> #  [0. 0. 0. 0. 0. 0. 0.]]
>>> output = pad(x)
>>> print(output)
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 2. 3. 0. 0.]
 [0. 0. 4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]]
>>> # if mode is "REFLECT"
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.pad = nn.Pad(paddings=((1, 1), (2, 2)), mode="REFLECT")
...     def construct(self, x):
...         return self.pad(x)
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6]]), mindspore.float32)
>>> pad = Net()
>>> output = pad(x)
>>> print(output)
[[6. 5. 4. 5. 6. 5. 4.]
 [3. 2. 1. 2. 3. 2. 1.]
 [6. 5. 4. 5. 6. 5. 4.]
 [3. 2. 1. 2. 3. 2. 1.]]
>>> # if mode is "SYMMETRIC"
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.pad = nn.Pad(paddings=((1, 1), (2, 2)), mode="SYMMETRIC")
...     def construct(self, x):
...         return self.pad(x)
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6]]), mindspore.float32)
>>> pad = Net()
>>> output = pad(x)
>>> print(output)
[[2. 1. 1. 2. 3. 3. 2.]
 [2. 1. 1. 2. 3. 3. 2.]
 [5. 4. 4. 5. 6. 6. 5.]
 [5. 4. 4. 5. 6. 6. 5.]]

```

3.13.2 mindspore.nn.ConstantPad1d

class mindspore.nn.ConstantPad1d(padding, value)

将给定的常量填充到多维输入数据的最后一维。

参数：

- **padding** (Union[int, tuple]) - 指定填充的大小。如果 *padding* 的类型为 *int*，则在输入最后一维的前后均填充 *padding* 大小，如果 *padding* 的类型为 *tuple*，形如 (*padding_0*, *padding_1*)，那么输入 *x* 对应输出的最后一维的 *shape* 为 *padding_0* + *x.shape*[-1] + *padding_1*，输出的其余维度与输入保持一致。在 Ascend 后端运行时，不支持 *padding* 包含负值情况。
- **value** (Union[int, float]) - 填充值。

输入：

- **x** (Tensor) - 输入 Tensor，*shape* 为 (*N*, *)，其中 * 表示任意维度。在 Ascend 后端运行时，不支持维度大于 5。

返回：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 既不是 *tuple* 或者 *int*。
- **TypeError** - *value* 既不是 *int*，也不是 *float*。
- **ValueError** - *tuple* 类型的 *padding* 长度不等于 2。
- **ValueError** - 填充后输出的维度不是正数。
- **ValueError** - 在 Ascend 后端运行时，*x* 的维度大于 5。
- **ValueError** - 在 Ascend 后端运行时，*padding* 中包含负值。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> # padding is tuple
>>> padding = (0, 1)
>>> value = 0.5
>>> pad1d = ms.nn.ConstantPad1d(padding, value)
>>> out = pad1d(x)
>>> print(out)
[[[[[1.  1.  1.  1.  0.5]
      [1.  1.  1.  1.  0.5]
      [1.  1.  1.  1.  0.5]]
     [[1.  1.  1.  1.  0.5]
      [1.  1.  1.  1.  0.5]
      [1.  1.  1.  1.  0.5]]]]]]
>>> print(out.shape)
(1, 2, 3, 5)
>>> # padding is int
>>> padding = 1
>>> value = 0.5
```

(下页继续)

(续上页)

```
>>> pad1d = ms.nn.ConstantPad1d(padding, value)
>>> out = pad1d(x)
>>> print(out)
[[[0.5 1. 1. 1. 1. 0.5]
  [0.5 1. 1. 1. 1. 0.5]
  [0.5 1. 1. 1. 1. 0.5]]
 [[0.5 1. 1. 1. 1. 0.5]
  [0.5 1. 1. 1. 1. 0.5]
  [0.5 1. 1. 1. 1. 0.5]]]]
>>> print(out.shape)
(1, 2, 3, 6)
```

3.13.3 mindspore.nn.ConstantPad2d

class mindspore.nn.ConstantPad2d(*padding, value*)

将给定的常量填充到多维输入数据的最后两维。

参数：

- **padding** (Union[int, tuple]) - 对输入的最后两维进行填充的大小。如果 **padding** 的类型为 int，则在输入最后两维的前后均填充 *padding* 大小，如果 **padding** 为长度为 4 的 tuple，形如 (*padding_0*, *padding_1*, *padding_2*, *padding_3*)，那么输入 *x* 对应输出的最后一维的 shape 为 *padding_0* + *x.shape*[-1] + *padding_1*，输入 *x* 对应输出的倒数第二维的 shape 为 *padding_2* + *x.shape*[-2] + *padding_3*，输出的其余维度与输入保持一致。在 Ascend 后端运行时，不支持 *padding* 包含负值情况。
- **value** (Union[int, float]) - 填充值。

输入：

- **x** (Tensor) - 输入 Tensor，shape 为 (*N*, *)，其中 * 表示任意维度。在 Ascend 后端运行时，不支持维度大于 5。

返回：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 既不是 tuple 或者 int。
- **TypeError** - *value* 既不是 int，也不是 float。
- **ValueError** - tuple 类型的 *padding* 长度大于 4 或者长度不是 2 的倍数。
- **ValueError** - 填充后输出的维度不是正数。
- **ValueError** - 在 Ascend 后端运行时，*x* 的维度大于 5。
- **ValueError** - 在 Ascend 后端运行时，*padding* 中包含负值。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> padding = (1, 1, 0, 1)
```

(下页继续)

(续上页)

```

>>> value = 0.5
>>> pad2d = ms.nn.ConstantPad2d(padding, value)
>>> out = pad2d(x)
>>> print(out)
[[[0.5  1.  1.  1.  1.  0.5]
  [0.5  1.  1.  1.  1.  0.5]
  [0.5  1.  1.  1.  1.  0.5]
  [0.5  0.5 0.5 0.5 0.5 0.5]]
 [0.5  1.  1.  1.  1.  0.5]
  [0.5  1.  1.  1.  1.  0.5]
  [0.5  1.  1.  1.  1.  0.5]
  [0.5  0.5 0.5 0.5 0.5 0.5]]]
>>> print(out.shape)
(1, 2, 4, 6)

```

3.13.4 mindspore.nn.ConstantPad3d

class mindspore.nn.ConstantPad3d(*padding, value*)

将给定的常量填充到多维输入数据的最后三维。

参数：

- **padding** (Union[int, tuple]) - 对输入的最后三维进行填充的大小。如果 **padding** 的类型为 **int**，则在输入最后三维的前后均填充 **padding** 大小；如果 **padding** 是长度为 6 的 **tuple**，形如 (**padding_0**, **padding_1**, **padding_2**, **padding_3**, **padding_4**, **padding_5**)，那么输入 *x* 对应输出的最后一维的 **shape** 为 **padding_0** + *x.shape*[-1] + **padding_1**，输入 *x* 对应输出的倒数第二维的 **shape** 为 **padding_2** + *x.shape*[-2] + **padding_3**，输入 *x* 对应输出的倒数第三维的 **shape** 为 **padding_4** + *x.shape*[-3] + **padding_5**，输出的其余维度与输入保持一致。在 Ascend 后端运行时，不支持 **padding** 包含负值。
- **value** (Union[int, float]) - 填充值。

输入：

- **x** (Tensor) - 输入 Tensor，**shape** 为 (*N*, *)，其中 * 表示任意维度。在 Ascend 后端运行时，不支持维度大于 5。

返回：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - **padding** 既不是 **tuple**，也不是 **int**。
- **TypeError** - **value** 既不是 **int**，也不是 **float**。
- **ValueError** - **tuple** 类型的 **padding** 长度大于 6 或者长度不是 2 的倍数。
- **ValueError** - 填充后输出的维度不是正数。
- **ValueError** - 在 Ascend 后端运行时，*x* 的维度大于 5。
- **ValueError** - 在 Ascend 后端运行时，**padding** 中包含负值。

支持平台：

Ascend GPU CPU

样例：

```

>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> padding = (1, 1, 0, 1, 1, 0)
>>> value = 0.5
>>> pad3d = ms.nn.ConstantPad3d(padding, value)
>>> out = pad3d(x)
>>> print(out)
[[[0.5 0.5 0.5 0.5 0.5 0.5]
  [0.5 0.5 0.5 0.5 0.5 0.5]
  [0.5 0.5 0.5 0.5 0.5 0.5]
  [0.5 0.5 0.5 0.5 0.5 0.5]]
 [0.5 1.  1.  1.  1.  0.5]
 [0.5 1.  1.  1.  1.  0.5]
 [0.5 1.  1.  1.  1.  0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5]]
 [0.5 1.  1.  1.  1.  0.5]
 [0.5 1.  1.  1.  1.  0.5]
 [0.5 1.  1.  1.  1.  0.5]
 [0.5 0.5 0.5 0.5 0.5 0.5]]]]
>>> print(out.shape)
(1, 3, 4, 6)

```

3.13.5 mindspore.nn.ReflectionPad1d

class mindspore.nn.ReflectionPad1d(*padding*)

根据 *padding* 对输入 *x* 进行填充。1d 代表填充的维度为一维。

参数：

- **padding** (union[int, tuple]) - 填充大小，如果输入为 int，则对所有边界进行相同大小的填充；如果是 tuple，则为 (*pad_left*, *pad_right*)。

输入：

- **x** (Tensor) - 输入 Tensor，2D 或 3D。shape 为 (*C*, *W_{in}*) 或 (*N*, *C*, *W_{in}*)。

输出：

Tensor，填充后的 Tensor，shape 为 (*C*, *W_{out}*) 或 (*N*, *C*, *W_{out}*)。其中 $W_{out} = W_{in} + pad\_left + pad\_right$ 。

异常：

- **TypeError** - *padding* 不是 tuple 或 int。
- **TypeError** - *padding* 中存在不是 int 的元素。
- **ValueError** - *padding* 是 tuple，且长度不能被 2 整除。
- **ValueError** - *padding* 是 tuple，且存在负数。
- **ValueError** - *padding* 是 tuple，且长度和 tensor 的维度不匹配。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = ms.Tensor(np.array([[0, 1, 2, 3], [4, 5, 6, 7]]).astype(np.float32))
>>> # x has shape (1, 2, 4)
>>> padding = (3, 1)
>>> # The first and the second dimension of x remain the same.
>>> # The third dimension of x: W_out = W_in + pad_left + pad_right = 4 + 3 + 1 = 8
>>> pad1d = ms.nn.ReflectionPad1d(padding)
>>> out = pad1d(x)
>>> # The shape of out is (1, 2, 8)
>>> print(out)
[[[3. 2. 1. 0. 1. 2. 3. 2.]
  [7. 6. 5. 4. 5. 6. 7. 6.]]]
```

3.13.6 mindspore.nn.ReflectionPad2d

class mindspore.nn.ReflectionPad2d(padding)

根据 padding 对输入 x 进行填充。2d 代表填充的维度为二维。

参数：

- **padding** (union[int, tuple]) - 填充大小，如果输入为 int，则对所有边界进行相同大小的填充；如果是 tuple，则顺序为 (pad_left, pad_right, pad_up, pad_down)。

输入：

- **x** (Tensor) - 输入 Tensor，3D 或 4D。shape 为 (C, H_{in}, W_{in}) 或 (N, C, H_{in}, W_{in}) 。

输出：

Tensor，填充后的 Tensor，shape 为 (C, H_{out}, W_{out}) 或 (N, C, H_{out}, W_{out}) 。其中 $H_{out} = H_{in} + pad_{up} + pad_{down}$ ， $W_{out} = W_{in} + pad_{left} + pad_{right}$ 。

异常：

- **TypeError** - padding 不是 tuple 或 int。
- **TypeError** - padding 中存在不是 int 的元素。
- **ValueError** - padding 是 tuple，且长度不能被 2 整除。
- **ValueError** - padding 是 tuple，且存在负数。
- **ValueError** - padding 是 tuple，且长度和 tensor 的维度不匹配。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = ms.Tensor(np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]]).astype(np.float32))
>>> # x has shape (1, 3, 3)
>>> padding = (1, 1, 2, 0)
>>> pad2d = ms.nn.ReflectionPad2d(padding)
>>> # The first dimension of x remains the same.
>>> # The second dimension of x: H_out = H_in + pad_up + pad_down = 3 + 1 + 1 = 5
>>> # The third dimension of x: W_out = W_in + pad_left + pad_right = 3 + 2 + 0 = 5
```

(下页继续)

(续上页)

```
>>> out = pad2d(x)
>>> # The shape of out is (1, 5, 5)
>>> print(out)
[[[7. 6. 7. 8. 7.]
  [4. 3. 4. 5. 4.]
  [1. 0. 1. 2. 1.]
  [4. 3. 4. 5. 4.]
  [7. 6. 7. 8. 7.]]]]
```

3.13.7 mindspore.nn.ReflectionPad3d

class mindspore.nn.ReflectionPad3d(padding)

使用反射的方式，以 *input* 的边界为对称轴，对 *input* 进行填充。3d 代表填充的维度为三维。

说明：ReflectionPad3d 尚不支持 5D Tensor 输入。

参数：

- **padding** (union[int, tuple]) - 填充大小，如果输入为 int，则对所有边界进行相同大小的填充；如果是 tuple，则顺序为 (*pad_left*, *pad_right*, *pad_up*, *pad_down*, *pad_front*, *pad_back*)。

输入：

- **x** (Tensor) - 4D Tensor，shape 为 ($N, D_{in}, H_{in}, W_{in}$)。

输出：

Tensor，填充后的 Tensor，shape 为 ($N, D_{out}, H_{out}, W_{out}$)。其中 $H_{out} = H_{in} + pad_{up} + pad_{down}$ 、 $W_{out} = W_{in} + pad_{left} + pad_{right}$ 、 $D_{out} = D_{in} + pad_{front} + pad_{back}$ 。

异常：

- **TypeError** - *padding* 不是 tuple 或 int。
- **TypeError** - *padding* 中存在不是 int 的元素。
- **ValueError** - *padding* 是 tuple，且长度不能被 2 整除。
- **ValueError** - *padding* 是 tuple，且存在负数。
- **ValueError** - *padding* 是 tuple，且长度和 Tensor 的维度不匹配。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> arr = np.arange(8).astype(np.float32).reshape((1, 2, 2, 2))
>>> x = ms.Tensor(arr)
>>> # x has shape (1, 2, 2, 2)
>>> padding = (1, 1, 1, 0, 0, 1)
>>> pad3d = ms.nn.ReflectionPad3d(padding)
>>> out = pad3d(x)
>>> # The first dimension of x remains the same.
```

(下页继续)

(续上页)

```
>>> # The second dimension of x: D_out = D_in + pad_front + pad_back = 2 + 0 + 1 = 3
>>> # The third dimension of x: H_out = H_in + pad_up + pad_down = 2 + 1 + 0 = 3
>>> # The last dimension of x: W_out = W_in + pad_left + pad_right = 2 + 1 + 1 = 4
>>> # The shape of out is (1, 3, 3, 4)
>>> print(out)
[[[ [3. 2. 3. 2.]
      [1. 0. 1. 0.]
      [3. 2. 3. 2.]]
  [[7. 6. 7. 6.]
    [5. 4. 5. 4.]
    [7. 6. 7. 6.]]
  [[3. 2. 3. 2.]
    [1. 0. 1. 0.]
    [3. 2. 3. 2.]]]]
```

3.13.8 mindspore.nn.ReplicationPad1d

class mindspore.nn.ReplicationPad1d(padding)

根据 padding 对输入 x 的 W 维度上进行填充。

参数：

- **padding** (union[int, tuple]) - 填充 x 最后一个维度的大小。
 - 如果输入为 int，则对所有边界进行相同大小的填充。
 - 如果是 tuple，则为 (pad_{left}, pad_{right})。

输入：

- **x** (Tensor) - 维度为 2D 或者 3D 的 Tensor。shape 为 (C, W_{in}) 或 (N, C, W_{in}) 。

输出：

Tensor，填充后的 Tensor，shape 为 (C, W_{out}) 或 (N, C, W_{out}) 。其中 $W_{out} = W_{in} + pad_{left} + pad_{right}$ 。

异常：

- **TypeError** - padding 不是 tuple 或 int。
- **TypeError** - padding 中存在不是 int 的元素。
- **ValueError** - padding 是 tuple，且长度不能被 2 整除。
- **ValueError** - padding 是 tuple，且长度和 Tensor 的维度不匹配。

支持平台：

GPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> pad1d = ms.nn.ReplicationPad1d(2)
>>> input = ms.Tensor(np.arange(0, 8).reshape(1, 2, 4), ms.float32)
>>> print(input)
[[[0. 1. 2. 3.]
  [4. 5. 6. 7.]]]
>>> out = pad1d(input)
```

(下页继续)

(续上页)

```
>>> print(out)
[[[0. 0. 0. 1. 2. 3. 3. 3.]
  [4. 4. 4. 5. 6. 7. 7. 7.]]]
>>> pad1d = ms.nn.ReplicationPad1d((3, 1))
>>> out = pad1d(input)
>>> print(out)
[[[0. 0. 0. 0. 0. 1. 2. 3. 3.]
  [4. 4. 4. 4. 5. 6. 7. 7.]]]
```

3.13.9 mindspore.nn.ReplicationPad2d

```
class mindspore.nn.ReplicationPad2d(padding)
```

根据 *padding* 对输入 x 的 HW 维度上进行填充。

参数:

- **padding** (union[int, tuple]) - 填充 x 最后两个维度的大小。
 - 如果输入为 int, 则对所有边界进行相同大小的填充。
 - 如果是 tuple, 则顺序为 (pad_{left} , pad_{right} , pad_{up} , pad_{down})。

输入:

- $\mathbf{x}$ (Tensor) - 维度为 3D 或 4D 的 Tensor, shape 为 (C, H_{in}, W_{in}) 或 (N, C, H_{in}, W_{in}) 。

输出:

Tensor, 填充后的 Tensor, shape 为 (C, H_{out}, W_{out}) 或 (N, C, H_{out}, W_{out}) 。其中 $H_{out} = H_{in} + pad_{up} + pad_{down}$, $W_{out} = W_{in} + pad_{left} + pad_{right}$ 。

異常:

- **TypeError** - *padding* 不是 tuple 或 int。
- **TypeError** - *padding* 中存在不是 int 的元素。
- **ValueError** - *padding* 是 tuple，且长度不能被 2 整除。
- **ValueError** - *padding* 是 tuple，且长度和 Tensor 的维度不匹配。

支持平台:

GPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> pad2d = ms.nn.ReplicationPad2d(2)
>>> input = ms.Tensor(np.arange(0, 9).reshape(1, 1, 3, 3), ms.float32)
>>> print(input)
[[[ [0. 1. 2.]
      [3. 4. 5.]
      [6. 7. 8.]]]]
>>> out = pad2d(input)
>>> print(out)
[[[ [0. 0. 0. 1. 2. 2. 2.]
      [0. 0. 0. 1. 2. 2. 2.]
      [0. 0. 0. 1. 2. 2. 2.] ]]]
```

(下页继续)

(续上页)

```

[3. 3. 3. 4. 5. 5. 5.]
[6. 6. 6. 7. 8. 8. 8.]
[6. 6. 6. 7. 8. 8. 8.]
[6. 6. 6. 7. 8. 8. 8.]]]]
>>> pad2d = ms.nn.ReplicationPad2d((1, 1, 2, 0))
>>> out = pad2d(input)
>>> print(out)
[[[0. 0. 1. 2. 2.]
  [0. 0. 1. 2. 2.]
  [0. 0. 1. 2. 2.]
  [3. 3. 4. 5. 5.]
  [6. 6. 7. 8. 8.]]]]

```

3.13.10 mindspore.nn.ReplicationPad3d

class mindspore.nn.ReplicationPad3d(padding)

根据 *padding* 对输入 *x* 的 DHW 维度进行填充。

参数：

- **padding** (union[int, tuple]) - 填充 *x* 最后三个维度的大小。
 - 如果输入为 int，则对所有边界进行相同大小的填充。
 - 如果是 tuple，则顺序为 (*pad_{left}*, *pad_{right}*, *pad_{up}*, *pad_{down}*, *pad_{front}*, *pad_{back}*)。

输入：

- **x** (Tensor) - 维度为 4D 或 5D 的 Tensor，shape 为 (*C*, *D_{in}*, *H_{in}*, *W_{in}*) 或 (*N*, *C*, *D_{in}*, *H_{in}*, *W_{in}*)。

输出：

Tensor，填充后的 Tensor，shape 为 (*C*, *D_{out}*, *H_{out}*, *W_{out}*) 或 (*N*, *C*, *D_{out}*, *H_{out}*, *W_{out}*)。其中 $D_{out} = D_{in} + pad_{front} + pad_{back}$, $H_{out} = H_{in} + pad_{up} + pad_{down}$, $W_{out} = W_{in} + pad_{left} + pad_{right}$ 。

异常：

- **TypeError** - *padding* 不是 tuple 或 int。
- **TypeError** - *padding* 中存在不是 int 的元素。
- **ValueError** - *padding* 是 tuple，且长度不能被 2 整除。
- **ValueError** - *padding* 是 tuple，且长度和 Tensor 的维度不匹配。

支持平台：

GPU

样例：

```

>>> import numpy as np
>>> import mindspore as ms
>>> pad3d = ms.nn.ReplicationPad3d(1)
>>> input = ms.Tensor(np.arange(0, 9).reshape(1, 1, 1, 3, 3), ms.float32)
>>> out = pad3d(input)
>>> print(out)
[[[[[0. 0. 1. 2. 2.]
     [0. 0. 1. 2. 2.]
     [3. 3. 4. 5. 5.]

```

(下页继续)

(续上页)

```
[6. 6. 7. 8. 8.]
[6. 6. 7. 8. 8.]]
[[0. 0. 1. 2. 2.]
 [0. 0. 1. 2. 2.]
 [3. 3. 4. 5. 5.]
 [6. 6. 7. 8. 8.]
 [6. 6. 7. 8. 8.]]
[[0. 0. 1. 2. 2.]
 [0. 0. 1. 2. 2.]
 [3. 3. 4. 5. 5.]
 [6. 6. 7. 8. 8.]
 [6. 6. 7. 8. 8.]]]]]
```

3.13.11 mindspore.nn.ZeroPad2d

class mindspore.nn.ZeroPad2d(*padding*)

将零填充到多维输入数据的最后两维。

参数：

- **padding** (Union[int, tuple]) - 对输入的最后两维进行填充的大小。如果 **padding** 的类型为 **int**，则在输入最后两维的前后均填充 *padding* 大小，如果 **padding** 为长度为 4 的 **tuple**，形如 (**padding_0**, **padding_1**, **padding_2**, **padding_3**)，那么输入 *x* 对应输出的最后一维的 **shape** 为 *padding_0* + *x.shape*[-1] + *padding_1*，输入 *x* 对应输出的倒数第二维的 **shape** 为 *padding_2* + *x.shape*[-2] + *padding_3*，输出的其余维度与输入保持一致。在 Ascend 后端运行时，不支持 *padding* 包含负值情况。

输入：

- **x** (Tensor) - 输入 Tensor，**shape** 为 (*N*, *)，其中 * 表示任意维度。在 Ascend 后端运行时，不支持维度大于 5。

返回：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 既不是 **tuple** 或者 **int**。
- **ValueError** - **tuple** 类型的 *padding* 长度大于 4 或者长度不是 2 的倍数。
- **ValueError** - 填充后输出的维度不是正数。
- **ValueError** - 在 Ascend 后端运行时，*x* 的维度大于 5。
- **ValueError** - 在 Ascend 后端运行时，*padding* 中包含负值。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> padding = (1, 1, 0, 1)
>>> pad = ms.nn.ZeroPad2d(padding)
>>> out = pad(x)
```

(下页继续)

(续上页)

```
>>> print(out)
[[[0. 1. 1. 1. 1. 0.]
  [0. 1. 1. 1. 1. 0.]
  [0. 1. 1. 1. 1. 0.]
  [0. 0. 0. 0. 0. 0.]]
 [[0. 1. 1. 1. 1. 0.]
  [0. 1. 1. 1. 1. 0.]
  [0. 1. 1. 1. 1. 0.]
  [0. 0. 0. 0. 0. 0.]]]]
>>> print(out.shape)
(1, 2, 4, 6)
```

3.14 损失函数

接口名	概述	支持平台
<code>mindspore.nn.BCELoss</code>	计算目标值和预测值之间的二值交叉熵损失值。	Ascend GPU CPU
<code>mindspore.nn.BCEWithLogitsLoss</code>	输入经过 <code>sigmoid</code> 激活函数后作为预测值, <code>BCEWithLogitsLoss</code> 计算预测值和目标值之间的二值交叉熵损失。	Ascend GPU CPU
<code>mindspore.nn.CosineEmbeddingLoss</code>	余弦相似度损失函数, 用于测量两个 <code>Tensor</code> 之间的相似性。	Ascend GPU CPU
<code>mindspore.nn.CrossEntropyLoss</code>	计算预测值和目标值之间的交叉熵损失。	Ascend GPU CPU
<code>mindspore.nn.CTCLoss</code>	<code>CTCLoss</code> 损失函数。	Ascend GPU CPU
<code>mindspore.nn.DiceLoss</code>	<code>Dice</code> 系数是一个集合相似性 <code>loss</code> , 用于计算两个样本之间的相似性。	Ascend GPU CPU
<code>mindspore.nn.FocalLoss</code>	<code>FocalLoss</code> 函数解决了类别不平衡的问题。	Ascend
<code>mindspore.nn.GaussianNLLLoss</code>	服从高斯分布的负对数似然损失。	Ascend GPU CPU
<code>mindspore.nn.HingeEmbeddingLoss</code>	<code>Hinge Embedding</code> 损失函数。	Ascend GPU CPU
<code>mindspore.nn.HuberLoss</code>	<code>HuberLoss</code> 计算预测值和目标值之间的误差。	Ascend GPU CPU
<code>mindspore.nn.KLDivLoss</code>	计算输入 <code>logits</code> 和 <code>labels</code> 的 <code>KL</code> 散度。	Ascend GPU CPU
<code>mindspore.nn.L1Loss</code>	<code>L1Loss</code> 用于计算预测值和目标值之间的平均绝对误差。	Ascend GPU CPU
<code>mindspore.nn.MarginRankingLoss</code>	排序损失函数, 用于创建一个衡量给定损失的标准。	Ascend GPU CPU
<code>mindspore.nn.MAELoss</code>	衡量 <code>x</code> 和 <code>y</code> 之间的平均绝对误差。	Ascend GPU CPU
<code>mindspore.nn.MSELoss</code>	用于计算预测值与标签值之间的均方误差。	Ascend GPU CPU
<code>mindspore.nn.MultiClassDiceLoss</code>	对于多标签问题, 可以将标签通过 <code>one-hot</code> 编码转换为多个二分类标签。	Ascend GPU CPU
<code>mindspore.nn.MultilabelMarginLoss</code>	创建一个损失函数, 用于最小化多分类任务的合页损失。	Ascend GPU
<code>mindspore.nn.MultiLabelSoftMarginLoss</code>	基于最大熵计算用于多标签优化的损失。	Ascend GPU CPU
<code>mindspore.nn.MultiMarginLoss</code>	多分类场景下用于计算 <code>x</code> 和 <code>y</code> 之间的合页损失 (<code>Hinge Loss</code>), 其中 <code>x</code> 为一个 2-D <code>Tensor</code> , <code>y</code> 为一个表示类别索引的 1-D <code>Tensor</code> , $0 \leq y \leq x.size(1) - 1$ 。	Ascend GPU CPU
<code>mindspore.nn.NLLLoss</code>	计算预测值和目标值之间的负对数似然损失。	Ascend GPU CPU
<code>mindspore.nn.PoissonNLLLoss</code>	计算泊松负对数似然损失。	Ascend GPU CPU

下页继续

表 14 – 续上页

<code>mindspore.nn.RMSELoss</code>	RMSELoss 用来测量 x 和 y 元素之间的均方根误差，其中 x 是输入 Tensor， y 是目标值。	Ascend GPU CPU
<code>mindspore.nn.SampledSoftmaxLoss</code>	抽样交叉熵损失函数。	GPU
<code>mindspore.nn.SmoothL1Loss</code>	SmoothL1 损失函数。	Ascend GPU CPU
<code>mindspore.nn.SoftMarginLoss</code>	针对二分类问题的损失函数。	Ascend GPU
<code>mindspore.nn.SoftmaxCrossEntropyWithLogits</code>	计算预测值与真实值之间的交叉熵。	Ascend GPU CPU
<code>mindspore.nn.TripletMarginLoss</code>	执行三元组损失函数的操作。	GPU

3.14.1 mindspore.nn.BCELoss

`class mindspore.nn.BCELoss (weight=None, reduction='mean')`

计算目标值和预测值之间的二值交叉熵损失值。
将预测值设置为 x ，目标值设置为 y ，输出损失设置为 $\ell(x, y)$ 。
则公式如下：

$$L = \{l_1, \dots, l_n, \dots, l_N\}^T, \quad l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

其中 N 是批次大小。公式如下：

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

说明：预测值一般是 sigmoid 函数的输出。因为是二分类，所以目标值应是 0 或者 1。如果输入是 0 或 1，则上述损失函数是无意义的。

参数：

- weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。与输入数据的 shape 和数据类型相同。默认值：None。
- reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none"：不应用规约方法。
 - "mean"：计算输出元素的加权平均值。
 - "sum"：计算输出元素的总和。

输入：

- logits** (Tensor) - 输入预测值 Tensor，shape $(N, *)$ ，其中 $*$ 代表任意数量的附加维度。数据类型必须为 float16 或 float32。
- labels** (Tensor) - 输入目标值 Tensor，shape $(N, *)$ ，其中 $*$ 代表任意数量的附加维度。与 logits 的 shape 和数据类型相同。

输出：

Tensor，数据类型与 logits 相同。如果 reduction 为 'none'，则 shape 与 logits 相同。否则，输出为 Scalar 的 Tensor。
异常：

- **TypeError** - *logits* 的数据类型, *labels* 或 *weight* (如果给定) 既不是 float16, 也不是 float32。
- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。
- **ValueError** - *logits* 的 shape 与 *labels* 或 *weight* (如果给定) 不同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> weight = ms.Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 3.3, 2.2]]), ms.float32)
>>> loss = nn.BCELoss(weight=weight, reduction='mean')
>>> logits = ms.Tensor(np.array([[0.1, 0.2, 0.3], [0.5, 0.7, 0.9]]), ms.float32)
>>> labels = ms.Tensor(np.array([[0, 1, 0], [0, 0, 1]]), ms.float32)
>>> output = loss(logits, labels)
>>> print(output)
1.8952923
```

3.14.2 mindspore.nn.BCEWithLogitsLoss

class mindspore.nn.BCEWithLogitsLoss (*reduction='mean', weight=None, pos_weight=None*)

输入经过 sigmoid 激活函数后作为预测值, *BCEWithLogitsLoss* 计算预测值和目标值之间的二值交叉熵损失。

将输入 *input* 设置为 *X*, 输入 *target* 为 *Y*, 输出为 *L*。则公式如下:

$$p_{ij} = \text{sigmoid}(X_{ij}) = \frac{1}{1 + e^{-X_{ij}}}$$

$$L_{ij} = -[Y_{ij} \cdot \log(p_{ij}) + (1 - Y_{ij}) \cdot \log(1 - p_{ij})]$$

然后,

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数:

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。
- **weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。如果不是 None, 将进行广播, 其 shape 与 *input* 的 shape 保持一致, 数据类型为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。默认值: None。
- **pos_weight** (Tensor, 可选) - 指定正样本的权重。是一个长度等于分类数的向量。如果不是 None, 将进行广播, 其 shape 与 *input* 的 shape 保持一致, 数据类型必须为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。默认值: None。

输入:

- **input** (Tensor) - 输入预测值 Tensor, shape $(N, *)$, 其中 $*$ 代表任意数量的附加维度。数据类型必须为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 输入目标值 Tensor, shape $(N, *)$, 其中 $*$ 代表任意数量的附加维度。与 **input** 的 shape 和数据类型相同。

输出:

Tensor 或 Scalar, 如果 *reduction* 为 'none', 其 shape 需和 *input* 相同。否则, 将返回 Scalar。

异常:

- **TypeError** - *input* 或 *target* 的不为 Tensor。
- **TypeError** - *input* 或 *target* 的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - *weight* 或 *pos_weight* 是 Parameter。
- **TypeError** - *weight* 或 *pos_weight* 的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - *reduction* 的数据类型不是 string。
- **ValueError** - *weight* 或 *pos_weight* 不能广播到 shape 为 *input* 的 Tensor。
- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> input = ms.Tensor(np.array([[ -0.8, 1.2, 0.7], [ -0.1, -0.4, 0.7]]).astype(np.float32))
>>> target = ms.Tensor(np.array([[ 0.3, 0.8, 1.2], [ -0.6, 0.1, 2.2]]).astype(np.float32))
>>> loss = nn.BCEWithLogitsLoss()
>>> output = loss(input, target)
>>> print(output)
0.3463612
```

3.14.3 mindspore.nn.CosineEmbeddingLoss

class mindspore.nn.CosineEmbeddingLoss (*margin=0.0, reduction='mean'*)

余弦相似度损失函数, 用于测量两个 Tensor 之间的相似性。

给定两个 Tensor x_1 和 x_2 , 以及一个 Tensor 标签 y , 值为 1 或 -1, 公式如下:

$$loss(x_1, x_2, y) = \begin{cases} 1 - \cos(x_1, x_2), & \text{if } y = 1 \\ \max(0, \cos(x_1, x_2) - \text{margin}), & \text{if } y = -1 \end{cases}$$

参数:

- **margin** (float) - 指定运算的调节因子, 取值范围 [-1.0, 1.0]。默认值: 0.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 "none"、"mean"、"sum", 默认值: "mean"。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。

- "sum": 计算输出元素的总和。

输入:

- **logits_x1** (Tensor) - 输入 Tensor, shape $(N, *)$, 其中 $*$ 代表任意数量的附加维度。
- **logits_x2** (Tensor) - 输入 Tensor, shape $(N, *)$ 。shape 和数据类型与 *logits_x1* 相同。
- **labels** (Tensor) - 输入值为 1 或 -1。假设 *logits_x1* 的 shape 是 $(x_1, x_2, x_3, \dots, x_R)$, 那么 *labels* 的 shape 必须是 $(x_1, x_3, x_4, \dots, x_R)$ 。

输出:

Tensor 或 Scalar, 如果 *reduction* 为 "none", 其 shape 与 *labels* 相同。否则, 将返回为 Scalar。

异常:

- **TypeError** - *margin* 不是 float。
- **ValueError** - *reduction* 不为 "none"、"mean" 或 "sum"。
- **ValueError** - *margin* 的值不在范围 $[-1.0, 1.0]$ 内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> logits_x1 = ms.Tensor(np.array([[0.3, 0.8], [0.4, 0.3]]), ms.float32)
>>> logits_x2 = ms.Tensor(np.array([[0.4, 1.2], [-0.4, -0.9]]), ms.float32)
>>> labels = ms.Tensor(np.array([1, -1]), ms.int32)
>>> cosine_embedding_loss = nn.CosineEmbeddingLoss()
>>> output = cosine_embedding_loss(logits_x1, logits_x2, labels)
>>> print(output)
0.0003425479
```

3.14.4 mindspore.nn.CrossEntropyLoss

class mindspore.nn.CrossEntropyLoss (*weight=None, ignore_index=-100, reduction='mean', label_smoothing=0.0*)
计算预测值和目标值之间的交叉熵损失。

CrossEntropyLoss 支持两种不同的目标值 (target):

- 类别索引 (int), 取值范围为 $[0, C)$ 其中 C 为类别数, 当 *reduction* 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot \mathbb{1}_{\{y_n \neq \text{ignore\_index}\}}$$

其中, x 表示预测值, t 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 *reduction* 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{y_n} \cdot \mathbb{1}_{\{y_n \neq \text{ignore\_index}\}}} l_n}{\sum_{n=1}^N l_n}, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

- 类别概率 (float), 用于目标值为多个类别标签的情况。当 **reduction** 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = - \sum_{c=1}^C w_c \log \frac{\exp(x_{n,c})}{\sum_{i=1}^C \exp(x_{n,i})} y_{n,c}$$

其中, x 表示预测值, t 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 **reduction** 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N l_n}{N}, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

参数:

- **weight** (Tensor) - 指定各类别的权重。若值不为 None, 则 shape 为 $(C,)$ 。数据类型仅支持 float32 或 float16。默认值: None。
- **ignore_index** (int) - 指定 target 中需要忽略的值 (一般为填充值), 使其不对梯度产生影响。默认值: -100。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。
- **label_smoothing** (float) - 标签平滑值, 用于计算 Loss 时防止模型过拟合的正则化手段。取值范围为 $[0.0, 1.0]$ 。默认值: 0.0。

输入:

- **logits** (Tensor) - 输入预测值, shape 为 $(C,)$ 、 (N, C) 或 $(N, C, d_1, d_2, \dots, d_K)$ (针对高维数据)。输入值需为对数概率。数据类型仅支持 float32 或 float16。
- **labels** (Tensor) - 输入目标值。若目标值为类别索引, 则 shape 为 $()$ 、 (N) 或 $(N, d_1, d_2, \dots, d_K)$, 数据类型仅支持 int32。若目标值为类别概率, 则 shape 为 $(C,)$ 、 (N, C) 或 $(N, C, d_1, d_2, \dots, d_K)$, 数据类型仅支持 float32 或 float16。

返回:

Tensor, 一个数据类型与 logits 相同的 Tensor。

异常:

- **TypeError** - weight 不是 Tensor。
- **TypeError** - weight 的 dtype 既不是 float16, 也不是 float32。
- **TypeError** - ignore_index 不是 int。
- **ValueError** - reduction 不为 'mean'、'sum' 或 'none'。
- **TypeError** - label_smoothing 不是 float。
- **TypeError** - logits 不是 Tensor。
- **TypeError** - labels 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> # Case 1: Indices labels
>>> inputs = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> target = ms.Tensor(np.array([1, 0, 4]), ms.int32)
>>> loss = nn.CrossEntropyLoss()
>>> output = loss(inputs, target)
>>> # Case 2: Probability labels
>>> inputs = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> target = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> loss = nn.CrossEntropyLoss()
>>> output = loss(inputs, target)
```

3.14.5 mindspore.nn.CTCLoss

class mindspore.nn.CTCLoss (blank=0, reduction='mean', zero_infinity=False)

CTCLoss 损失函数。主要用来计算连续未分段的时间序列与目标序列之间的损失。

关于 CTCLoss 算法详细介绍, 请参考 [Connectionist Temporal Classification: Labeling Unsegmented Sequence Data with Recurrent Neural Networks](#)。

参数:

- **blank** (int, 可选) - 空白标签。默认值: 0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 "none"、"mean"、"sum", 默认值: "mean"。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。
- **zero_infinity** (bool, 可选) - 在损失无限大的时候, 是否将无限损失和相关梯度置为零。默认值: False。

输入:

- **log_probs** (Tensor) - 预测值, shape 为 (T, N, C) 或 (T, C) 。其中 T 表示输入长度, N 表示批次大小, C 是分类数。 T , N , C 均为正整数。
- **targets** (Tensor) - 目标值, shape 为 (N, S) 或 $(\text{sum}(\text{target\_lengths}))$ 。其中 S 表示最大目标长度。
- **input_lengths** (Union[tuple, Tensor]) - shape 为 (N) 的 Tensor 或 tuple。表示输入长度。
- **target_lengths** (Union[tuple, Tensor]) - shape 为 (N) 的 Tensor 或 tuple。表示目标长度。

输出:

- **neg_log_likelihood** (Tensor) - 对每一个输入节点可微调的损失值。

异常:

- **TypeError** - `log_probs` 或 `targets` 不是 Tensor。
- **TypeError** - `zero_infinity` 不是布尔值, `reduction` 不是字符串。
- **TypeError** - `log_probs` 的数据类型不是 float 或 double。
- **TypeError** - `targets`, `input_lengths` 或 `target_lengths` 数据类型不是 int32 或 int64。

- **ValueError** - *reduction* 不为 "none" , "mean" 或 "sum" 。
- **ValueError** - *blank* 值不介于 0 到 C 之间。C 是 *log_probs* 的分类数。
- **ValueError** - 当 *log_prob* 的 shape 是 (T, C) 时, *target* 的维度不是 1 或 2。
- **ValueError** - 当 *log_prob* 的 shape 是 (T, C) 时, *target* 的首个维度的长度不是 1。
- **RuntimeError** - *input_lengths* 的值大于 T。T 是 *log_probs* 的长度。
- **RuntimeError** - *target_lengths[i]* 的值不介于 0 到 *input_length[i]* 之间。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> T = 5          # Input sequence length
>>> C = 2          # Number of classes
>>> N = 2          # Batch size
>>> S = 3          # Target sequence length of longest target in batch (padding length)
>>> S_min = 2      # Minimum target length, for demonstration purposes
>>> arr = np.arange(T*N*C).reshape((T, N, C))
>>> ms_input = ms.Tensor(arr, dtype=ms.float32)
>>> input_lengths = np.full(shape=(N), fill_value=T)
>>> input_lengths = ms.Tensor(input_lengths, dtype=ms.int32)
>>> target_lengths = np.full(shape=(N), fill_value=S_min)
>>> target_lengths = ms.Tensor(target_lengths, dtype=ms.int32)
>>> target = np.random.randint(1, C, size=(N, S))
>>> target = ms.Tensor(target, dtype=ms.int32)
>>> ctc_loss = nn.CTCLoss(blank=0, reduction='none', zero_infinity=False)
>>> loss = ctc_loss(ms_input, target, input_lengths, target_lengths)
>>> print(loss)
[-45.79497 -55.794968]
>>> arr = np.arange(T*C).reshape((T, C))
>>> ms_input = ms.Tensor(arr, dtype=ms.float32)
>>> input_lengths = ms.Tensor([T], dtype=ms.int32)
>>> target_lengths = ms.Tensor([S_min], dtype=ms.int32)
>>> target = np.random.randint(1, C, size=(S_min,))
>>> target = ms.Tensor(target, dtype=ms.int32)
>>> ctc_loss = nn.CTCLoss(blank=0, reduction='none', zero_infinity=False)
>>> loss = ctc_loss(ms_input, target, input_lengths, target_lengths)
>>> print(loss)
-25.794968
```

3.14.6 mindspore.nn.DiceLoss

class mindspore.nn.DiceLoss (*smooth=1e-5*)

Dice 系数是一个集合相似性 loss, 用于计算两个样本之间的相似性。当分割结果最好时, Dice 系数的值为 1; 当分割结果最差时, Dice 系数的值为 0。

Dice 系数表示两个对象之间的面积与总面积的比率。函数如下:

$$dice = 1 - \frac{2 * |pred \cap true|}{|pred| + |true| + smooth}$$

pred 表示 logits , *true* 表示 labels 。

参数:

- **smooth** (float, 可选) - 将添加到分母中, 以提高数值稳定性的参数。取值大于 0。默认值: $1e-5$ 。

输入:

- **logits** (Tensor) - 输入预测值。数据类型必须为 float16 或 float32。
- **labels** (Tensor) - 输入目标值, 一般与 *logits* 的 shape 相同。数据类型必须为 float16 或 float32。

输出:

Tensor, shape 为每样本采样的 Dice 系数的 Tensor。

异常:

- **ValueError** - *logits* 的维度与 *labels* 不同。
- **TypeError** - *logits* 或 *labels* 的类型不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> loss = nn.DiceLoss(smooth=1e-5)
>>> logits = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]), mindspore.float32)
>>> labels = Tensor(np.array([[0, 1], [1, 0], [0, 1]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.38596618
```

3.14.7 mindspore.nn.FocalLoss

class mindspore.nn.FocalLoss (*weight=None, gamma=2.0, reduction='mean'*)

FocalLoss 函数解决了类别不平衡的问题。

FocalLoss 函数由 Kaiming 团队在论文 [Focal Loss for Dense Object Detection](#) 中提出, 提高了图像目标检测的效果。

函数如下:

$$FL(p_t) = -(1 - p_t)^\gamma \log(p_t)$$

参数:

- **gamma** (float) - gamma 用于调整 Focal Loss 的权重曲线的陡峭程度。默认值: 2.0。
- **weight** (Union[Tensor, None]) - Focal Loss 的权重, 维度为 1。如果为 None, 则不使用权重。默认值: None。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。

输入:

- **logits** (Tensor) - shape 为 (N, C) 、 (N, C, H) 、或 (N, C, H, W) 的 Tensor, 其中 C 是分类的数量, 值大于 1。如果 shape 为 (N, C, H, W) 或 (N, C, H) , 则 H 或 H 和 W 的乘积应与 $labels$ 的相同。
- **labels** (Tensor) - shape 为 (N, C) 、 (N, C, H) 、或 (N, C, H, W) 的 Tensor, C 的值为 1, 或者与 $logits$ 的 C 相同。如果 C 不为 1, 则 shape 应与 $logits$ 的 shape 相同, 其中 C 是分类的数量。如果 shape 为 (N, C, H, W) 或 (N, C, H) , 则 H 或 H 和 W 的乘积应与 $logits$ 相同。 $labels$ 的值应该在 $[-C, C)$ 范围内, 其中 C 是 $logits$ 中类的数量。

输出:

Tensor 或 Scalar, 如果 *reduction* 为 "none", 其 shape 与 *logits* 相同。否则, 将返回 Scalar。

异常:

- **TypeError** - *gamma* 的数据类型不是 float。
- **TypeError** - *weight* 不是 Tensor。
- **ValueError** - *labels* 维度与 *logits* 不同。
- **ValueError** - *labels* 通道不为 1, 且 *labels* 的 shape 与 *logits* 不同。
- **ValueError** - *reduction* 不为 'mean', 'sum', 或 'none'。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> logits = ms.Tensor([[0.8, 1.4], [0.5, 0.9], [1.2, 0.9]], ms.float32)
>>> labels = ms.Tensor([[1], [1], [0]], ms.int32)
>>> focalloss = nn.FocalLoss(weight=ms.Tensor([1, 2]), gamma=2.0, reduction='mean')
>>> output = focalloss(logits, labels)
>>> print(output)
0.12516622
```

3.14.8 mindspore.nn.GaussianNLLLoss

class mindspore.nn.GaussianNLLLoss (*, full=False, eps=1e-6, reduction='mean')

服从高斯分布的负对数似然损失。

目标值被认为是高斯分布的采样, 其中期望和方差通过神经网络来预测。对于以高斯分布为模型的 Tensor *labels* 和记录期望的 Tensor *logits*, 以及均为正数的方差 Tensor *var* 来说, 计算的 loss 为:

$$\text{loss} = \frac{1}{2} \left(\log(\max(\text{var}, \text{eps})) + \frac{(\text{logits} - \text{labels})^2}{\max(\text{var}, \text{eps})} \right) + \text{const.}$$

其中, *eps* 用于 *log* 的稳定性。当 *full* = True 时, 一个常数 *const* 会被添加到 loss 中。如果 *var* 和 *logits* 的 shape 不一致 (出于同方差性的假设), 那么它们必须能够正确地广播。

关键字参数:

- **full** (bool, 可选) - 指定损失函数中的常数部分。如果 *full* = True, 则常数 *const*. 为 $0.5 * \log(2\pi)$ 。默认值: False。
- **eps** (float, 可选) - 用于提高 *log* 的稳定性。默认值: $1e-6$ 。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。

- "none": 不应用规约方法。
- "mean": 计算输出元素的平均值。
- "sum": 计算输出元素的总和。

输入:

- **logits** (Tensor) - shape 为 $(N, *)$ 或 $(*)$ 。* 代表着任意数量的额外维度。
- **labels** (Tensor) - shape 为 $(N, *)$ 或 $(*)$ 。和 *logits* 具有相同 shape, 或者相同 shape 但有一个维度为 1 (以允许广播)。
- **var** (Tensor) - shape 为 $(N, *)$ 或 $(*)$ 。和 *logits* 具有相同 shape, 或者相同 shape 但有一个维度为 1, 或者少一个维度 (以允许广播)。

返回:

Tensor 或 Tensor scalar, 根据 *reduction* 计算的 loss。

异常:

- **TypeError** - *logits* 不是 Tensor。
- **TypeError** - *labels* 不是 Tensor。
- **TypeError** - *full* 不是 bool。
- **TypeError** - *eps* 不是 float。
- **ValueError** - *eps* 不是在 $(0, \text{inf})$ 区间的 float。
- **ValueError** - *reduction* 不是 "none"、"mean" 或者 "sum"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> arr1 = np.arange(8).reshape((4, 2))
>>> arr2 = np.array([2, 3, 1, 4, 6, 4, 4, 9]).reshape((4, 2))
>>> logits = ms.Tensor(arr1, ms.float32)
>>> labels = ms.Tensor(arr2, ms.float32)
>>> loss = nn.GaussianNLLLoss(reduction='mean')
>>> var = ms.Tensor(np.ones((4, 1)), ms.float32)
>>> output = loss(logits, labels, var)
>>> print(output)
1.4374993
```

3.14.9 mindspore.nn.HingeEmbeddingLoss

class mindspore.nn.HingeEmbeddingLoss (*margin=1.0, reduction='mean'*)

Hinge Embedding 损失函数。根据输入 *logits* 和 *labels* (只包含 1 或 -1) 计算 Hinge Embedding 损失值。通常用来衡量两个输入之间的相似度。

mini-batch 中的第 n 个样例的损失函数为:

$$l_n = \begin{cases} x_n, & \text{if } y_n = 1, \\ \max\{0, \Delta - x_n\}, & \text{if } y_n = -1, \end{cases}$$

总损失值为：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction} = \text{'mean'}; \\ \text{sum}(L), & \text{if reduction} = \text{'sum'}. \end{cases}$$

其中 $L = \{l_1, \dots, l_N\}^\top$ 。

参数：

- **margin** (float, int, 可选) - Hinge Embedding Loss 公式定义的阈值 *margin*。公式中表示为 Δ 。默认值：1.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none"：不应用规约方法。
 - "mean"：计算输出元素的平均值。
 - "sum"：计算输出元素的总和。

输入：

- **logits** (Tensor) - 预测值，公式中表示为 x 。其 shape 为 $(*)$ 。* 代表着任意数量的维度。
- **labels** (Tensor) - 标签值，公式中表示为 y 。其与 *logits* 具有相同 shape，包含 1 或 -1。

返回：

Tensor 或 Tensor scalar，根据 *reduction* 计算的 loss。

异常：

- **TypeError** - *logits* 不是 Tensor。
- **TypeError** - *labels* 不是 Tensor。
- **TypeError** - *margin* 不是 float 或 int。
- **ValueError** - *labels* 和 *logits* 的 shape 不一致且不能广播。
- **ValueError** - *reduction* 不是 "none"、"mean" 或者 "sum"。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> arr1 = np.array([0.9, -1.2, 2, 0.8, 3.9, 2, 1, 0, -1]).reshape((3, 3))
>>> arr2 = np.array([1, 1, -1, 1, -1, 1, -1, 1, 1]).reshape((3, 3))
>>> logits = ms.Tensor(arr1, ms.float32)
>>> labels = ms.Tensor(arr2, ms.float32)
>>> loss = nn.HingeEmbeddingLoss(reduction='mean')
>>> output = loss(logits, labels)
>>> print(output)
0.16666667
```


3.14.10 mindspore.nn.HuberLoss

class mindspore.nn.HuberLoss (reduction='mean', delta=1.0)

HuberLoss 计算预测值和目标值之间的误差。它兼有 L1Loss 和 MSELoss 的优点。

假设 x 和 y 为一维 Tensor，长度 N ，则计算 x 和 y 的 loss 而不进行降维操作（即 reduction 参数设置为“none”）。公式如下：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T$$

以及

$$l_n = \begin{cases} 0.5 * (x_n - y_n)^2, & \text{if } |x_n - y_n| < \text{delta}; \\ \text{delta} * (|x_n - y_n| - 0.5 * \text{delta}), & \text{otherwise.} \end{cases}$$

其中， N 为 batch size。如果 reduction 不是“none”，则：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = "mean"}; \\ \text{sum}(L), & \text{if reduction = "sum"} \end{cases}$$

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none"：不应用规约方法。
 - "mean"：计算输出元素的平均值。
 - "sum"：计算输出元素的总和。
- **delta** (Union[int, float]) - 两种损失之间变化的阈值。该值必须为正。默认值：1.0。

输入：

- **logits** (Tensor) - 输入预测值，任意维度的 Tensor。其数据类型为 float16 或 float32。
- **labels** (Tensor) - 目标值，通常情况下与 logits 的 shape 和 dtype 相同。但是如果 logits 和 labels 的 shape 不同，需要保证他们之间可以互相广播。

输出：

Tensor 或 Scalar，如果 reduction 为“none”，返回与 logits 具有相同 shape 和 dtype 的 Tensor。否则，将返回一个 Scalar。

异常：

- **TypeError** - logits 或 labels 的数据类型既不是 float16 也不是 float32。
- **TypeError** - logits 和 labels 的数据类型不同。
- **TypeError** - delta 不是 float 或 int。
- **ValueError** - delta 的值小于或等于 0。
- **ValueError** - reduction 不为 "mean"、"sum" 或 "none"。
- **ValueError** - logits 和 labels 有不同的 shape，且不能互相广播。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> loss = nn.HuberLoss()
>>> logits = ms.Tensor(np.array([1, 2, 3]), ms.float32)
>>> labels = ms.Tensor(np.array([1, 2, 2]), ms.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.16666667
>>> # Case 2: logits.shape = (3,), labels.shape = (2, 3)
>>> loss = nn.HuberLoss(reduction="none")
>>> logits = ms.Tensor(np.array([1, 2, 3]), ms.float32)
>>> labels = ms.Tensor(np.array([[1, 1, 1], [1, 2, 2]]), ms.float32)
>>> output = loss(logits, labels)
>>> print(output)
[[0.  0.5 1.5]
 [0.  0.  0.5]]

```

3.14.11 mindspore.nn.KLDivLoss

class mindspore.nn.KLDivLoss (reduction='mean')

计算输入 *logits* 和 *labels* 的 KL 散度。

对于相同 shape 的 Tensor *x* 和 *target* , KLDivLoss 的计算公式如下:

$$L(x, target) = target \cdot (\log target - \log x)$$

输出

$$\ell(x, target) = \begin{cases} L(x, target), & \text{if reduction = 'none';} \\ \text{mean}(L(x, target)), & \text{if reduction = 'mean';} \\ \text{sum}(L(x, target))/x.\text{shape}[0], & \text{if reduction = 'batchmean';} \\ \text{sum}(L(x, target)), & \text{if reduction = 'sum'}. \end{cases}$$

其中 *x* 代表 *logits* ; *target* 代表 *labels* ; $\ell(x, target)$ 为 *output* 。

说明:

- 目前 Ascend 平台不支持数据类型 float64。
- 仅当 *reduction* 设置为“batchmean” 时输出才与 Kullback-Leibler 散度的数学定义一致。

参数:

- **reduction** (str) - 指定输出结果的计算方式。默认值: 'mean' 。
 - 在 Ascend 平台上, *reduction* 的可选值为 'batchmean' 、 'none' 或 'sum' 。
 - 在 GPU 平台上, *reduction* 的可选值为 'mean' 、 'none' 或 'sum' 。
 - 在 CPU 平台上, *reduction* 的可选值为 'mean' 、 'batchmean' 、 'none' 或 'sum' 。

输入:

- **logits** (Tensor) - 数据类型支持 float16、float32 或 float64。

- **labels** (Tensor) - 标签 Tensor, 与 *logits* 的 shape 和数据类型相同。

输出:

Tensor 或标量。如果 *reduction* 为 'none', 则输出为 Tensor 且与 *logits* 的 shape 相同。否则为标量。

异常:

- **TypeError** - *reduction* 不是 str。
- **TypeError** - *logits* 或 *labels* 不是 Tensor。
- **TypeError** - *logits* 或 *labels* 的数据类型不是支持的类型。
- **ValueError** - *logits* 和 *labels* 的 shape 不一致。
- **RuntimeError** - *logits* 或 *labels* 是标量并且 *reduction* 是 "batchmean"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> logits = ms.Tensor(np.array([0.2, 0.7, 0.1]), ms.float32)
>>> labels = ms.Tensor(np.array([0., 1., 0.]), ms.float32)
>>> loss = nn.KLDivLoss(reduction='mean')
>>> output = loss(logits, labels)
>>> print(output)
-0.23333333
```

3.14.12 mindspore.nn.L1Loss

class mindspore.nn.L1Loss (*reduction='mean'*)

L1Loss 用于计算预测值和目标值之间的平均绝对误差。

假设 x 和 y 为一维 Tensor, 长度 N , 则计算 x 和 y 的 loss, 而不进行降维操作 (即 *reduction* 参数设置为 "none")。

公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad \text{with } l_n = |x_n - y_n|,$$

其中, N 为 batch size。如果 *reduction* 不是 "none", 则:

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = 'mean'}; \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数:

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

输入:

- **logits** (Tensor) - 预测值。可以是任意维度的 Tensor。
- **labels** (Tensor) - 目标值。通常情况下与 *logits* 的 shape 相同。但是如果 *logits* 和 *labels* 的 shape 不同，需要保证它们之间可以互相广播。

输出：

Tensor, 类型为 float。

异常：

- **ValueError** - *reduction* 不为 "mean"、"sum" 或 "none"。
- **ValueError** - *logits* 和 *labels* 有不同的 shape，且不能互相广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> loss = nn.L1Loss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.33333334
>>> # Case 2: logits.shape = (3,), labels.shape = (2, 3)
>>> loss = nn.L1Loss(reduction='none')
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
[[0. 1. 2.]
 [0. 0. 1.]]
```

3.14.13 mindspore.nn.MarginRankingLoss

class mindspore.nn.**MarginRankingLoss** (*margin=0.0, reduction='mean'*)

排序损失函数，用于创建一个衡量给定损失的标准。

给定两个 Tensor *input1* 和 *input2*，以及一个 Tensor 标签 *target*，值为 1 或 -1，公式如下：

$$\text{loss}(\text{input1}, \text{input2}, \text{target}) = \max(0, -\text{target} * (\text{input1} - \text{input2}) + \text{margin})$$

参数：

- **margin** (float, 可选) - 指定运算的调节因子。默认值：0.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - 'none'：不应用规约方法。
 - 'mean'：计算输出元素的平均值。
 - 'sum'：计算输出元素的总和。

输入：

- **input1** (Tensor) - 输入 Tensor, shape $(N,*)$, 其中 $*$ 代表任意数量的附加维度。
- **input2** (Tensor) - 输入 Tensor, shape $(N,*)$ 。shape 和数据类型与 *input1* 相同。
- **target** (Tensor) - 输入值为 1 或-1。假设 *input1* 的 shape 是 $(x_1, x_2, x_3, \dots, x_R)$, 那么 *target* 的 shape 必须是 $(x_1, x_2, x_3, \dots, x_R)$ 。

输出：

Tensor 或 Scalar, 如果 *reduction* 为 'none', 其 shape 与 *input1* 相同。否则, 将返回为 Scalar。

异常：

- **TypeError** - *margin* 不是 float。
- **TypeError** - *input1*, *input2* 和 *target* 不是 Tensor。
- **TypeError** - *input1* 和 *input2* 类型不一致。
- **TypeError** - *input1* 和 *target* 类型不一致。
- **ValueError** - *input1* 和 *input2* shape 不一致。
- **ValueError** - *input1* 和 *target* shape 不一致。
- **ValueError** - *reduction* 不为 'none', 'mean' 或 'sum'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, ops
>>> import numpy as np
>>> loss1 = nn.MarginRankingLoss(reduction='none')
>>> loss2 = nn.MarginRankingLoss(reduction='mean')
>>> loss3 = nn.MarginRankingLoss(reduction='sum')
>>> sign = ops.Sign()
>>> input1 = Tensor(np.array([0.3864, -2.4093, -1.4076]), ms.float32)
>>> input2 = Tensor(np.array([-0.6012, -1.6681, 1.2928]), ms.float32)
>>> target = sign(Tensor(np.array([-2, -2, 3]), ms.float32))
>>> output1 = loss1(input1, input2, target)
>>> print(output1)
[0.98759997 0.          2.7003999 ]
>>> output2 = loss2(input1, input2, target)
>>> print(output2)
1.2293333
>>> output3 = loss3(input1, input2, target)
>>> print(output3)
3.6879997
```

3.14.14 mindspore.nn.MAELoss

class mindspore.nn.**MAELoss** (*reduction='mean'*)

衡量 x 和 y 之间的平均绝对误差。其中 x 是输入 *logits* , y 是标签 *labels* 。

简单来说, 假设 x 和 y 是两个长度为 N 的 1D Tensor。未归约前的 (参数 *reduction* 是 'none') 损失为:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad \text{with } l_n = |x_n - y_n|$$

N 是批次 (batch) 个数。

如果 *reduction* 不是 'none', 则:

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = 'mean'}; \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数:

- **reduction** (str, 可选) - 对输出使用归约算法: 'none'、'mean'、'sum'。默认值: 'mean'。
 - 'none': 不使用规约算法。
 - 'mean': 计算输出的平均值。
 - 'sum': 计算输出中所有元素的和。

输入:

- **logits** (Tensor) - Tensor 的 shape 是 $(M, *)$, 其中, $*$ 的含义是任意额外的维度。
- **labels** (Tensor) - Tensor 的 shape 是 $(N, *)$, 通常和 *logits* 的 shape 相同。然而, 当 *logits* 和 *labels* 的 shape 不同时, 它们需要支持广播。

输出:

Tensor, 加权损失, dtype 是 float, 如果 *reduction* 是 'mean' 或 'sum', shape 则为 0, 否则当 *reduction* 是 'none' 时, shape 是广播之后的 shape。

异常:

- **ValueError** - 如果 *reduction* 不是 'none', 'mean', 'sum' 中的一个。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> loss = nn.MAELoss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.33333334
>>> # Case 2: logits.shape = (3,), labels.shape = (2, 3)
>>> loss = nn.MAELoss(reduction='none')
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> output = loss(logits, labels)
>>> print(output)
[[0. 1. 2.]
 [0. 0. 1.]]
```

3.14.15 mindspore.nn.MSELoss

class mindspore.nn.MSELoss (*reduction='mean'*)

用于计算预测值与标签值之间的均方误差。

假设 x 和 y 为一维 Tensor，长度 N ，则计算 x 和 y 的 unreduced loss（即 *reduction* 参数设置为 'none'）的公式如下：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad \text{with} \quad l_n = (x_n - y_n)^2.$$

其中， N 为 batch size。如果 *reduction* 不是 'none'，则：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = 'mean'}; \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'mean'、'sum'、'none'，默认值：'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。
 - 'sum': 计算输出元素的总和。

输入：

- **logits** (Tensor) - 输入预测值，任意维度的 Tensor。
- **labels** (Tensor) - 输入标签，任意维度的 Tensor。支持在 *logits* 和 *labels* shape 不相同的情况下，通过广播保持一致。

输出：

- Tensor，如果 *reduction* 为 'mean' 或 'sum' 时，输出的 shape 为 *Tensor Scalar*。
- 如果 *reduction* 为 'none'，输出的 shape 则是 *logits* 和 *labels* 广播之后的 shape。

异常：

- **ValueError** - 如果 *reduction* 不为 'mean'，'sum' 或 'none' 中的一个。
- **ValueError** - 如果 *logits* 和 *labels* 的 shape 不同，且不能广播。
- **TypeError** - 如果 *logits* 和 *labels* 数据类型不一致。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> loss = nn.MSELoss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 1, 1]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
1.6666667
>>> # Case 2: logits.shape = (3,), labels.shape = (2, 3)
>>> loss = nn.MSELoss(reduction='none')
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
[[0. 1. 4.]
 [0. 0. 1.]]

```

3.14.16 mindspore.nn.MultiClassDiceLoss

class mindspore.nn.MultiClassDiceLoss (weights=None, ignore_index=None, activation='softmax')

对于多标签问题，可以将标签通过 one-hot 编码转换为多个二分类标签。每个通道可以看做是一个二分类问题，所以损失可以通过先计算每个类别的二分类的 `mindspore.nn.DiceLoss` 损失，再计算各二分类损失的平均值得到。

参数：

- **weights** (Union[Tensor, None]) - Shape 为 (num_classes, dim) 的 Tensor。权重 shape[0] 应等于标签 shape[1]。默认值：None。
- **ignore_index** (Union[int, None]) - 指定需要忽略的类别序号，如果为 None，计算所有类别的 Dice Loss 值。默认值：None。
- **activation** (Union[str, Cell]) - 应用于全连接层输出的激活函数，如'ReLU'。取值范围：['softmax', 'logsoftmax', 'relu', 'relu6', 'tanh', 'Sigmoid']。默认值：'softmax'。

输入：

- **logits** (Tensor) - shape 为 (N, C, *) 的 Tensor，其中 * 表示任意数量的附加维度。logits 维度应大于 1。数据类型必须为 float16 或 float32。
- **labels** (Tensor) - shape 为 (N, C, *) 的 Tensor，与 logits 的 shape 相同。标签维度应大于 1。数据类型必须为 float16 或 float32。

输出：

Tensor，输出为每个样本采样通过 MultiClassDiceLoss 函数计算所得。

异常：

- **ValueError** - logits 与 labels 的 shape 不同。
- **TypeError** - logits 或 labels 的类型不是 Tensor。
- **ValueError** - logits 或 labels 的维度小于 2。
- **ValueError** - weights 的 shape[0] 和 labels 的 shape[1] 不相等。
- **ValueError** - weights 是 Tensor，但其维度不是 2。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> loss = nn.MultiClassDiceLoss(weights=None, ignore_index=None, activation="softmax")
>>> logits = Tensor(np.array([[0.2, 0.5, 0.7], [0.3, 0.1, 0.5], [0.9, 0.6, 0.3]]), mindspore.
    ↪float32)
>>> labels = Tensor(np.array([[0, 1, 0], [1, 0, 0], [0, 0, 1]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.54958105
```

3.14.17 mindspore.nn.MultilabelMarginLoss

class mindspore.nn.MultilabelMarginLoss (*reduction='mean'*)

创建一个损失函数，用于最小化多分类任务的合页损失。它以一个 2D mini-batch Tensor x 作为输入，以包含目标类索引的 2D Tensor y 作为输出。

对于每个小批量样本，loss 值根据如下公式计算：

$$\text{loss}(x, y) = \sum_{ij} \frac{\max(0, 1 - (x[y[j]] - x[i]))}{x.size(0)}$$

其中 $x \in \{0, \dots, x.size(0) - 1\}$, $y \in \{0, \dots, y.size(0) - 1\}$, $0 \leq y[j] \leq x.size(0) - 1$, 并且对于所有 i 和 j , i 不等于 $y[j]$ 。

y 和 x shape 必须相同。

说明：该算子仅考虑从前方开始的连续非负目标块。这允许不同的样本具有不同数量的目标类别。

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 "none"、"mean"、"sum"，默认值："mean"。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

输入：

- **x** (Tensor) - 预测值，shape 为 (C) 或 (N, C)，其中 N 为批量大小， C 为类别数。数据类型必须为：float16 或 float32。
- **target** (Tensor) - 目标值，shape 与 x 相同，数据类型必须为 int32，标签目标值由-1 填充。

输出：

- **y** (Union[Tensor, Scalar]) - MultilabelMarginLoss 损失。如果 *reduction* 的值为 "none"，那么返回 shape 为 (N) 的 Tensor 类型数据。否则返回一个标量。

异常：

- **TypeError** - 当 x 或者 *target* 数据不是 Tensor 时。

- **TypeError** - 当 x 数据类型不是以下其中之一时: float16、float32。
- **TypeError** - 当 $target$ 数据类型不是 int32 时。
- **ValueError** - 当 x 的数据维度不是以下其中之一时: 1、2。
- **ValueError** - 当 x 和 $target$ 的 shape 不相同。
- **ValueError** - 当 $reduction$ 的值不是以下其中之一时: "none"、"mean"、"sum"。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> loss = nn.MultilabelMarginLoss()
>>> x = ms.Tensor(np.array([[0.1, 0.2, 0.4, 0.8], [0.2, 0.3, 0.5, 0.7]]), ms.float32)
>>> target = ms.Tensor(np.array([[1, 2, 0, 3], [2, 3, -1, 1]]), ms.int32)
>>> output = loss(x, target)
>>> print(output)
0.325
```

3.14.18 mindspore.nn.MultiLabelSoftMarginLoss

class mindspore.nn.MultiLabelSoftMarginLoss (*weight=None, reduction='mean'*)

基于最大熵计算用于多标签优化的损失。

多标签软间隔损失通常用于多标签分类任务中, 输入样本可以属于多个目标类别。给定输入 x 和二元标签 y , 其 shape 为 (N, C) , N 表示样本数量, C 为样本类别数, 损失计算公式如下:

$$loss(x, y) = -\frac{1}{N} \frac{1}{C} \sum_{i=1}^N \sum_{j=1}^C \left(y_{ij} \log \frac{1}{1 + e^{-x_{ij}}} + (1 - y_{ij}) \log \frac{e^{-x_{ij}}}{1 + e^{-x_{ij}}} \right)$$

其中 x_{ij} 表示样本 i 在 j 类别的概率得分。 y_{ij} 表示样本 i 是否属于类别 j , $y_{ij} = 1$ 时属于, 为 0 时不属于。对于多标签分类任务, 每个样本可以属于多个类别, 即标签中含有多个 1。如果 *weight* 不为 None, 将会和每个分类的 loss 相乘。

参数:

- **weight** (Union[Tensor, int, float]) - 每个类别的缩放权重。默认值: None。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的加权平均值。
 - "sum": 计算输出元素的总和。

输入:

- **x** (Tensor) - shape 为 (N, C) 的 Tensor, N 为 batch size, C 为类别个数。
- **target** (Tensor) - 目标值, 数据类型和 shape 与 x 的相同。

输出:

Tensor, 数据类型和 x 相同。如果 $reduction$ 为 "none", 其 shape 为 (N)。否则, 其 shape 为 0。

异常:

- **ValueError** - x 或 $target$ 的维度不等于 2。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> x = ms.Tensor([[0.3, 0.6, 0.6], [0.9, 0.4, 0.2]])
>>> target = ms.Tensor([[0.0, 0.0, 1.0], [0.0, 0.0, 1.0]])
>>> loss = nn.MultiLabelSoftMarginLoss(reduction='mean')
>>> out = loss(x, target)
>>> print(out.asnumpy())
0.84693956
```

3.14.19 mindspore.nn.MultiMarginLoss

class mindspore.nn.MultiMarginLoss ($p=1$, $margin=1.0$, $reduction='mean'$, $weight=None$)

多分类场景下用于计算 x 和 y 之间的合页损失 (Hinge Loss), 其中 x 为一个 2-D Tensor, y 为一个表示类别索引的 1-D Tensor, $0 \leq y \leq x.size(1) - 1$ 。

对于每个小批量样本, 1D 输入 x 和标量 y 的损失为:

$$\text{loss}(x, y) = \frac{\sum_i \max(0, w[y] * (\text{margin} - x[y] + x[i]))^p}{x.size(0)}$$

其中 $x \in \{0, \dots, x.size(0) - 1\}$ 并且 $i \neq y$ 。

参数:

- **p** (int, 可选) - 对偶距离的范数度。必须为 1 或 2。默认值: 1。
- **margin** (float, 可选) - 改变对偶距离的参数。默认值: 1.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的加权平均值。
 - "sum": 计算输出元素的总和。
- **weight** (Tensor, 可选) - 每个类别的缩放权重, shape 为 (C,)。数据类型只支持 float32、float16 或 float64。默认值: None, 表示各个类别权重相同。

输入:

- **x** (Tensor) - shape 为 (N, C)。数据类型只支持 float32、float16 或 float64。即上述公式中的 x 。
- **target** (Tensor) - 真实标签, shape 为 (N,)。数据类型只支持 int64。值应为非负值, 且小于 C。 $target$ 即上述公式中的 y 。

输出:

Tensor, 当 $reduction$ 为 'none' 时, 类型为 Tensor, shape 为 (N,) , 和 $target$ 相同。否则为标量 Tensor。

异常:

- **TypeError** - p 或者 $target$ 数据类型不是 `int`。
- **TypeError** - $margin$ 数据类型不是 `int`。
- **TypeError** - $reduction$ 数据类型不是 `str`。
- **TypeError** - x 数据类型不是以下之一: `float16`、`float`、`float64`。
- **TypeError** - $weight$ 和 x 的数据类型不相同。
- **ValueError** - p 的值不是以下之一: 1、2。
- **ValueError** - $reduction$ 的值不是以下之一: { 'none', 'sum', 'mean' }。
- **ValueError** - x 的 `shape[0]` 和 $target$ 的 `shape[0]` 不相等。
- **ValueError** - x 的 `shape[1]` 和 $weight$ 的 `shape[0]` 不相等。
- **ValueError** - 如果 $weight$ 的维度不是 1。
- **ValueError** - 如果 x 的维度不是 2 或 $target$ 的维度不是 1。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> x = ms.Tensor(np.ones(shape=[3, 3]), ms.float32)
>>> target = ms.Tensor(np.array([1, 2, 1]), ms.int64)
>>> loss = nn.MultiMarginLoss()
>>> output = loss(x, target)
>>> print(output)
0.6666667
```

3.14.20 mindspore.nn.NLLLoss

class mindspore.nn.NLLLoss ($weight=None$, $ignore\_index=-100$, $reduction='mean'$)

计算预测值和目标值之间的负对数似然损失。

$reduction = none$ 时, 负对数似然损失公式如下:

$$\ell(x, t) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{t_n} x_{n, t_n}, \quad w_c = \text{weight}[c] \cdot \mathbb{1}\{c \neq \text{ignore\_index}\}$$

其中, x 表示预测值, t 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 $reduction \neq none$ 时 (默认为 'mean'), 则

$$\ell(x, t) = \begin{cases} \sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{t_n}} l_n, & \text{if } reduction = 'mean', \\ \sum_{n=1}^N l_n, & \text{if } reduction = 'sum' \end{cases}$$

参数:

- **weight** (Tensor) - 指定各类别的权重。若值不为 `None`, 则 `shape` 为 $(C,)$ 。数据类型仅支持 `float32` 或 `float16`。默认值: `None`。
- **ignore_index** (int) - 指定 `target` 中需要忽略的值 (一般为填充值), 使其不对梯度产生影响。默认值: `-100`。

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的加权平均值。
 - "sum": 计算输出元素的总和。

输入:

- **logits** (Tensor) - 输入预测值, shape 为 (N, C) 或 $(N, C, d_1, d_2, \dots, d_K)$ (针对 K 维数据)。inputs 需为对数概率。数据类型仅支持 float32 或 float16。
- **labels** (Tensor) - 输入目标值, shape 为 (N) 或 $(N, d_1, d_2, \dots, d_K)$ (针对 K 维数据)。数据类型仅支持 int32。

返回:

Tensor, 一个数据类型与 logits 相同的 Tensor。

异常:

- **TypeError** - weight 不是 Tensor。
- **TypeError** - weight 的 dtype 既不是 float16, 也不是 float32。
- **TypeError** - ignore_index 不是 int。
- **ValueError** - reduction 不为 "mean"、"sum" 或 "none"。
- **TypeError** - logits 不是 Tensor。
- **TypeError** - labels 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> logits = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> labels = ms.Tensor(np.array([1, 0, 4]), ms.int32)
>>> loss = nn.NLLLoss()
>>> output = loss(logits, labels)
```

3.14.21 mindspore.nn.PoissonNLLLoss

class mindspore.nn.PoissonNLLLoss (log_input=True, full=False, eps=1e-08, reduction='mean')

计算泊松负对数似然损失。

损失为:

$$\mathcal{L}_D = \sum_{i=0}^{|D|} (x_i - y_i \ln x_i + \ln y_i!)$$

其中 $\mathcal{L}_D$ 为损失值, y_i 为 target, x_i 为 input。

如果 log_input 为 True, 使用 $e^{x_i} - y_i x_i$ 而不是 $x_i - y_i \ln x_i$ 进行计算。计算对数时, input 的下界设置为 eps, 以避免数值误差。如果 full 为 False, 则最后一项: $\ln y_i!$ 将被省略。否则, 最后一项将使用斯特林公式近似:

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

说明：在 Ascend 下计算负数的对数或大正数的指数将具有与 GPU 和 CPU 下不同的返回值和结果范围。

参数：

- **log_input** (bool, 可选) - 是否使用对数输入。默认值：True。
- **full** (bool, 可选) - 是否在损失计算中包括斯特林近似项。默认值：False。
- **eps** (float, 可选) - 算对数时 *input* 的下界。默认值：1e-08。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none"：不应用规约方法。
 - "mean"：计算输出元素的平均值。
 - "sum"：计算输出元素的总和。

输入：

- **input** (Tensor) - 输入 Tensor。shape 可以是任意维。
- **target** (Tensor) - 标签 Tensor，其 shape 与 *input* 相同。

输出：

Tensor 或 Scalar，如果 *reduction* 为 none，则输出是 Tensor，其形状与 *input* 相同。否则，它是 Scalar。

异常：

- **TypeError** - *reduction* 不是 str 类型。
- **TypeError** - *input* 或 *target* 都不是 Tensor。
- **TypeError** - *input* 或 *target* 的数据类型不支持。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> x = ms.Tensor([[0.3, 0.7], [0.5, 0.5]])
>>> target = ms.Tensor([[1.0, 2.0], [3.0, 4.0]])
>>> loss = nn.PoissonNLLLoss()
>>> output = loss(x, target)
>>> print(output.asnumpy())
0.3652635
```

3.14.22 mindspore.nn.RMSELoss

class mindspore.nn.RMSELoss

RMSELoss 用来测量 x 和 y 元素之间的均方根误差, 其中 x 是输入 Tensor, y 是目标值。

假设 x 和 y 为一维 Tensor, 长度为 N , x 和 y 的 loss 为:

$$loss = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - y_i)^2}$$

输入:

- **logits** (Tensor) - 输入的预测值 Tensor, shape ($N, *$), 其中 $*$ 代表任意数量的附加维度。
- **labels** (Tensor) - 输入的目标值 Tensor, shape ($N, *$)。一般与 *logits* 的 shape 相同。如果 *logits* 和 *labels* 的 shape 不同, 需支持广播。

输出:

Tensor, 输出值为加权损失值, 其数据类型为 float, 其 shape 为 ()。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> loss = nn.RMSELoss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.57735026
>>> # Case 2: logits.shape = (3,), labels.shape = (2, 3)
>>> loss = nn.RMSELoss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
1.0
```

3.14.23 mindspore.nn.SampledSoftmaxLoss

class mindspore.nn.SampledSoftmaxLoss (*num_sampled, num_classes, num_true=1, sampled_values=None, remove_accidental_hits=True, seed=0, reduction='none'*)

抽样交叉熵损失函数。

一般在类别数很大时使用。可加速训练以交叉熵为损失函数的分类器。

参数:

- **num_sampled** (int) - 抽样的类别数。
- **num_classes** (int) - 类别总数。
- **num_true** (int) - 每个训练样本的类别数。默认值: 1。

- **sampled_values** (Union[list, tuple]) - 抽样候选值。由 **CandidateSampler* 函数返回的 (*sampled_candidates*, *true_expected_count*, *sampled_expected_count*) 的 list 或 tuple。如果默认值为 None，则应用 *UniformCandidateSampler*。默认值: None。
- **remove_accidental_hits** (bool) - 是否移除抽样中的目标类等于标签的情况。默认值: True。
- **seed** (int) - 抽样的随机种子。默认值: 0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'none'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

输入:

- **weights** (Tensor) - 输入的权重, shape 为 (*C*, *dim*) 的 Tensor。
- **bias** (Tensor) - 分类的偏置。shape 为 (*C*,) 的 Tensor。
- **labels** (Tensor) - 输入目标值 Tensor, 其 shape 为 (*N*, *num_true*), 其数据类型为 *int64*, *int32*。
- **logits** (Tensor) - 输入预测值 Tensor, 其 shape 为 (*N*, *dim*)。

输出:

Tensor 或 Scalar, 如果 *reduction* 为 'none', 则输出是 shape 为 (*N*,) 的 Tensor。否则, 输出为 Scalar。

异常:

- **TypeError** - *sampled_values* 不是 list 或 tuple。
- **TypeError** - *labels* 的数据类型既不是 *int32*, 也不是 *int64*。
- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。
- **ValueError** - *num_sampled* 或 *num_true* 大于 *num_classes*。
- **ValueError** - *sampled_values* 的长度不等于 3。

支持平台:

GPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> mindspore.set_seed(1)
>>> loss = nn.SampledSoftmaxLoss(num_sampled=4, num_classes=7, num_true=1)
>>> weights = Tensor(np.random.randint(0, 9, [7, 10]), mindspore.float32)
>>> biases = Tensor(np.random.randint(0, 9, [7]), mindspore.float32)
>>> labels = Tensor([0, 1, 2])
>>> logits = Tensor(np.random.randint(0, 9, [3, 10]), mindspore.float32)
>>> output = loss(weights, biases, labels, logits)
>>> print(output)
[4.6051701e+01 1.4000047e+01 6.1989022e-06]
```


3.14.24 mindspore.nn.SmoothL1Loss

class mindspore.nn.SmoothL1Loss (beta=1.0, reduction='none')

SmoothL1 损失函数。逐元素进行对比，如果预测值和目标值的绝对误差小于设定阈值 *beta*，则用平方项，否则用绝对误差项。

给定两个输入 *x*, *y*, SmoothL1Loss 定义如下：

$$L_i = \begin{cases} \frac{0.5(x_i - y_i)^2}{\text{beta}}, & \text{if } |x_i - y_i| < \text{beta} \\ |x_i - y_i| - 0.5 * \text{beta}, & \text{otherwise.} \end{cases}$$

其中，*beta* 代表阈值 *beta*。

当 *reduction* 不是设定为 *none* 时，计算如下：

$$L = \begin{cases} \text{mean}(L_i), & \text{if reduction = 'mean'}; \\ \text{sum}(L_i), & \text{if reduction = 'sum'}. \end{cases}$$

说明：

- SmoothL1Loss 可以看成 *mindspore.nn.L1Loss* 的修改版本，也可以看成 *mindspore.nn.L1Loss* 和 *mindspore.ops.L2Loss* 的组合。
- *mindspore.nn.L1Loss* 计算两个输入 Tensor 之间的绝对误差，而 *mindspore.ops.L2Loss* 计算两个输入 Tensor 之间的平方误差。
- *mindspore.ops.L2Loss* 通常更快收敛，但对离群值的鲁棒性较差。该损失函数具有较好的鲁棒性。

参数：

- **beta** (number, 可选) - 损失函数计算在 L1Loss 和 L2Loss 间变换的阈值。默认值：1.0。
 - Ascend: 该值必须大于等于 0。
 - CPU/GPU: 该值必须大于 0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'none'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

输入：

- **logits** (Tensor) - 预测值，任意维度 Tensor。支持数据类型：
 - Ascend: float16、float32、bfloat16。
 - CPU/GPU: float16、float32、float64。
- **labels** (Tensor) - 目标值，数据类型和 shape 与 *logits* 相同的 Tensor。
 - CPU/Ascend: 与 *logits* 的 shape 相同，*logits* 和 *labels* 遵循隐式类型转换规则，使数据类型一致。
 - GPU: 与 *logits* 的 shape 和数据类型相同。

输出：

Tensor。如果 *reduction* 为 'none'，则输出为 Tensor，且 shape 与 *logits* 的 shape 相同。否则 shape 为 ()。

异常：

- **TypeError** - *logits* 或 *labels* 不是 Tensor。
- **RuntimeError** - *logits* 或 *labels* 的数据类型不是 float16、float32、float64 或 bfloat16。
- **ValueError** - *logits* 与 *labels* 的 shape 不同。
- **ValueError** - *reduction* 不是 'none'、'mean' 或 'sum'。
- **TypeError** - *beta* 不是 float、bool 或 int。
- **RuntimeError** - *beta* 小于等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> loss = nn.SmoothL1Loss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
[0.  0.  0.5]
```

3.14.25 mindspore.nn.SoftMarginLoss

class mindspore.nn.SoftMarginLoss (*reduction='mean'*)

针对二分类问题的损失函数。

SoftMarginLoss 用于计算输入 Tensor *x* 和目标值 Tensor *y*（包含 1 或 -1）的二分类损失值。

$$\text{loss}(x, y) = \sum_i \frac{\log(1 + \exp(-y[i] * x[i]))}{x.\text{nelement}()}$$

x.nelement() 代表 *x* 中元素的个数。

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

输入：

- **logits** (Tensor) - 预测值，数据类型为 float16、float32、bfloat16（Atlas 训练系列产品不支持）。
- **labels** (Tensor) - 目标值，shape 与 *logits* 的相同。GE 模式下类型需要与 *logits* 相同。

输出：

Tensor 或 Scalar，如果 *reduction* 为 "none"，其 shape 与 *logits* 相同。否则，将返回 Scalar。

异常：

- **TypeError** - *logits* 或 *labels* 不是 Tensor。
- **TypeError** - *logits* 或 *labels* 的数据类型不是 float16、float32、bfloat16（Atlas 训练系列产品不支持）。
- **ValueError** - *logits* 的 shape 与 *labels* 不同。
- **ValueError** - *reduction* 不为 "mean" , "sum" , 或 "none" 。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> loss = nn.SoftMarginLoss()
>>> logits = Tensor(np.array([[0.3, 0.7], [0.5, 0.5]]), mindspore.float32)
>>> labels = Tensor(np.array([[-1, 1], [1, -1]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.6764238
```

3.14.26 mindspore.nn.SoftmaxCrossEntropyWithLogits

class mindspore.nn.SoftmaxCrossEntropyWithLogits (*sparse=False, reduction='none'*)

计算预测值与真实值之间的交叉熵。

使用交叉熵损失函数计算出输入概率（使用 softmax 函数计算）和真实值之间的误差。

函数的输入是未标准化的值，表示为 x ，格式为 (N, C) ，以及相应的目标。

通常情况下，该函数的输入为各类别的分数值以及对应的目标值，输入格式是 (N, C) 。

对于每个实例 x_i ， i 的范围为 0 到 $N-1$ ，则可得损失为：

$$\ell(x_i, c) = -\log \left(\frac{\exp(x_i[c])}{\sum_j \exp(x_i[j])} \right) = -x_i[c] + \log \left(\sum_j \exp(x_i[j]) \right)$$

其中 x_i 是一维的 Tensor， c 为 one-hot 中等于 1 的位置。

说明：虽然目标值是互斥的，即目标值中只有一个为正，但预测的概率不为互斥。只要求输入的预测概率分布有效。

参数：

- **sparse** (bool, 可选) - 指定目标值是否使用稀疏格式。默认值：False。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'none'。
 - "none"：不应用规约方法。
 - "mean"：计算输出元素的平均值。
 - "sum"：计算输出元素的总和。

输入：

- **logits** (Tensor) - shape (N, C) 的 Tensor。数据类型为 float16 或 float32。

- **labels** (Tensor) - shape (N ,) 的 Tensor。如果 *sparse* 为 True , 则 *labels* 的类型为 int32 或 int64。否则, *labels* 的类型与 *logits* 的类型相同。

输出:

Tensor, 一个 shape 和数据类型与 logits 相同的 Tensor。

异常:

- **TypeError** - *sparse* 不是 bool。
- **TypeError** - *sparse* 为 True , 并且 *labels* 的 dtype 既不是 int32, 也不是 int64。
- **TypeError** - *sparse* 为 False, 并且 *labels* 的 dtype 既不是 float16, 也不是 float32。
- **ValueError** - *reduction* 不为 "mean"、"sum" 或 "none" 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> # case 1: sparse=True
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> logits = Tensor(np.array([[3, 5, 6, 9, 12, 33, 42, 12, 32, 72]]), mindspore.float32)
>>> labels_np = np.array([1]).astype(np.int32)
>>> labels = Tensor(labels_np)
>>> output = loss(logits, labels)
>>> print(output)
[67.]
>>> # case 2: sparse=False
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> logits = Tensor(np.array([[3, 5, 6, 9, 12, 33, 42, 12, 32, 72]]), mindspore.float32)
>>> labels_np = np.array([[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]]).astype(np.float32)
>>> labels = Tensor(labels_np)
>>> output = loss(logits, labels)
>>> print(output)
[30.]
```

3.14.27 mindspore.nn.TripletMarginLoss

class mindspore.nn.TripletMarginLoss ($p=2$, $swap=False$, $eps=1e-06$, $reduction='mean'$, $margin=1.$)

执行三元组损失函数的操作。

三元组损失值通常用来测量样本之间的相似度, 由一个三元组和一个大于 0 的 *margin* 计算得到。其中, 三元组由下面公式中的 a 、 p 和 n 组成。

所有输入 Tensor 的 shape 都应该为 (N ,*), 其中 N 代表批处理的数量, * 代表任意数量的附加维度。距离交换在 V. Balntas、E. Riba 等人在论文 [Learning local feature descriptors with triplets and shallow convolutional neural networks](#) 中有详细的阐述。

对于每个小批量样本, 损失值为:

$$L(a, p, n) = \max\{d(a_i, p_i) - d(a_i, n_i) + \text{margin}, 0\}$$

其中

$$d(x_i, y_i) = \|x_i - y_i\|_p$$

参数:

- **p** (int, 可选) - 成对距离的范数。默认值: 2。
- **swap** (bool, 可选) - 距离交换。默认值: False。
- **eps** (float, 可选) - 防止除数为 0。默认值: 1e-06。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。
- **margin** (Union[Tensor, float]) - 用于拉近 x 和 $positive$ 之间的距离, 拉远 x 和 $negative$ 之间的距离, $margin$ 的形状长度必须为 0, 默认值: 1.0。

输入:

- **x** (Tensor) - 从训练集随机选取的样本。数据类型为 BasicType。即上述公式中的 a 。
- **positive** (Tensor) - 与 x 为同一类的样本, 数据类型与 $shape$ 与 x 一致。即上述公式中的 p 。
- **negative** (Tensor) - 与 x 为异类的样本, 数据类型与 $shape$ 与 x 一致。即上述公式中的 n 。
- **margin** (Union[Tensor, float]) - 用于拉近 x 和 $positive$ 之间的距离, 拉远 x 和 $negative$ 之间的距离, $margin$ 的 $shape$ 长度必须为 0, 默认值: 1.0。

输出:

Tensor。如果 $reduction$ 为 "none", 其 $shape$ 为 (N) 。否则, 将返回 Scalar。

异常:

- **TypeError** - x 、 $positive$ 、 $negative$ 不是 Tensor。
- **TypeError** - x 、 $positive$ 或者 $negative$ 的数据类型不一致。
- **TypeError** - p 的数据类型不是 int。
- **TypeError** - eps 的数据类型不是 float。
- **TypeError** - $swap$ 的数据类型不是 bool。
- **ValueError** - x 、 $positive$ 和 $negative$ 的维度同时小于等于 1。
- **ValueError** - x 、 $positive$ 或 $negative$ 的维度大于等于 8。
- **ValueError** - $margin$ 的 $shape$ 长度不为 0。
- **ValueError** - x 、 $positive$ 和 $negative$ 三者之间的 $shape$ 无法广播。
- **ValueError** - $reduction$ 不为 "mean"、"sum" 或 "none"。

支持平台:

GPU

样例:

```

>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> import numpy as np
>>> loss = nn.TripletMarginLoss()
>>> x = ms.Tensor(np.array([[0.3, 0.7], [0.5, 0.5]]), ms.float32)
>>> positive = ms.Tensor(np.array([[0.4, 0.6], [0.4, 0.6]]), ms.float32)
>>> negative = ms.Tensor(np.array([[0.2, 0.9], [0.3, 0.7]]), ms.float32)
>>> output = loss(x, positive, negative)
>>> print(output)
0.8881968

```

3.15 优化器

接口名	概述	支持平台
<code>mindspore.nn.Adadelta</code>	Adadelta 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.Adagrad</code>	Adagrad 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.Adam</code>	Adaptive Moment Estimation (Adam) 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.AdaMax</code>	AdaMax 算法是基于无穷范数的 Adam 的一种变体。	Ascend GPU CPU
<code>mindspore.nn.AdamWeightDecay</code>	权重衰减 Adam 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.AdaSumByDeltaWeightWrapCell</code>	Adaptive Summation (AdaSum) 算法的实现, 根据更新前后的参数差计算。	Ascend GPU
<code>mindspore.nn.AdaSumByGradWrapCell</code>	Adaptive Summation (AdaSum) 算法的实现, 根据梯度计算。	Ascend GPU
<code>mindspore.nn.ASGD</code>	随机平均梯度下降 (ASGD) 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.FTRL</code>	FTRL 算法实现。	Ascend GPU
<code>mindspore.nn.Lamb</code>	LAMB (Layer-wise Adaptive Moments optimizer for Batch training, 用于批训练的分层自适应矩优化器) 算法的实现。	Ascend GPU
<code>mindspore.nn.LARS</code>	LARS 算法的实现。	Ascend
<code>mindspore.nn.LazyAdam</code>	Adaptive Moment Estimation (Adam) 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.Momentum</code>	Momentum 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.OptTFTWrapper</code>	实现 TFT 优化器封装器。	Ascend
<code>mindspore.nn.ProximalAdagrad</code>	ProximalAdagrad 算法的实现, 用于在线学习和随机优化。	Ascend GPU
<code>mindspore.nn.RMSProp</code>	均方根传播 (RMSProp) 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.Rprop</code>	弹性反向传播 (Rprop) 算法的实现。	Ascend GPU CPU
<code>mindspore.nn.SGD</code>	随机梯度下降的实现。	Ascend GPU CPU

3.15.1 mindspore.nn.Adadelta

class mindspore.nn.Adadelta (*params*, *learning_rate*=1.0, *rho*=0.9, *epsilon*=1e-6, *loss_scale*=1.0, *weight_decay*=0.0)
Adadelta 算法的实现。

Adadelta 用于在线学习和随机优化。请参阅论文 [ADADELTA: AN ADAPTIVE LEARNING RATE METHOD](#).

$$\begin{aligned} accum_t &= \rho * accum_{t-1} + (1 - \rho) * g_t^2 \\ update_t &= \sqrt{accum\_update_{t-1} + \epsilon} * \frac{g_t}{\sqrt{accum_t + \epsilon}} \\ accum\_update_t &= \rho * accum\_update_{t-1} + (1 - \rho) * update_t^2 \\ w_t &= w_{t-1} - \gamma * update_t \end{aligned}$$

其中 g 代表 *grads*, γ 代表 *learning_rate*, ρ 代表 *rho*, ϵ 代表 *epsilon*, w 代表 *params*, *accum* 表示累加器, *accum_update* 表示累加器更新, t 代表当前 *step*。

说明:

- 在参数未分组时, 优化器配置的 *weight_decay* 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 *weight_decay*。若未配置, 则该组网络参数使用优化器中配置的 *weight_decay*。

参数:

- params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是 “*params*”、“*lr*”、“*weight_decay*”、“*grad_centralization*”和 “*order_params*”:
 - params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - lr** - 可选。如果键中存在“lr”, 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
 - weight_decay** - 可选。如果键中存在“weight_decay”, 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 *weight_decay* 作为权重衰减值。
 - grad_centralization** - 可选。如果键中存在“grad_centralization”, 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 *grad_centralization* 为 False。该参数仅适用于卷积层。
 - order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时, 通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”, 则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 默认值: 1.0。
 - float** - 固定的学习率。必须大于等于零。
 - int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 i 步将取向量中第 i 个值作为学习率。
 - Iterable** - 动态的学习率。第 i 步将取迭代器第 i 个值作为学习率。
 - LearningRateSchedule** - 动态的学习率。在训练过程中, 优化器将使用步数 (*step*) 作为输入, 调用 [LearningRateSchedule](#) 实例来计算当前学习率。
- rho** (float, 可选) - 衰减率, 应在 [0.0, 1.0] 范围内。默认值: 0.9。

- **epsilon** (float, 可选) - 分母添加项, 非负数。默认值: $1e-6$ 。
- **loss_scale** (float, 可选) - 梯度缩放系数, 必须大于 0.0。如果 *loss_scale* 是整数, 它将被转换为浮点数。通常使用默认值, 仅当训练时使用了 *FixedLossScaleManager*, 且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 *False* 时, 此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息, 请参阅 *mindspore.amp.FixedLossScaleManager*。默认值: 1.0。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减值, 必须大于等于 0.0。默认值: 0.0。
 - float: 固定的权重衰减值。必须等于或大于 0。
 - int: 固定的权重衰减值。必须等于或大于 0。它将会被转换为 float 类型。
 - Cell: 权重衰减此时是动态的。在训练期间, 优化器调用该 Cell 的实例, 以获取当前阶段所要使用的权重衰减值。

输入:

- **grads** (tuple[Tensor]) - 优化器中 *params* 的梯度, *shape* 和数据类型与 *params* 相同。数据类型为 float16 或 float32。

输出:

Tensor[bool], 值为 True。

异常:

- **TypeError** - *learning_rate* 不是 int、float、Tensor、Iterable 或 *LearningRateSchedule*。
- **TypeError** - *params* 的元素既不是 *Parameter* 也不是字典。
- **TypeError** - *rho*、*epsilon* 或 *loss_scale* 不是 float。
- **TypeError** - *weight_decay* 不是 float、int 或 cell。
- **ValueError** - *rho* 不在范围 [0.0, 1.0] 内。
- **ValueError** - *loss_scale* 小于或等于 0。
- **ValueError** - *learning_rate*、*epsilon* 或 *weight_decay* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv = nn.Conv1d(120, 240, 4)
...         self.fc = nn.Dense(4, 1)
...     def construct(self, x):
...         x = self.conv(x)
...         x = self.fc(x)
...         return x
>>> net = Net()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.Adadelta(params=net.trainable_params())
>>>
```

(下页继续)

(续上页)

```

>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
    ↪,
...                 {'params': no_conv_params, 'lr': 0.01},
...                 {'order_params': net.trainable_params()}]
>>> optim = nn.Adadelta(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of_
    ↪0.01 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight_
    ↪decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
    ↪'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)

```

3.15.2 mindspore.nn.Adagrad

class mindspore.nn.**Adagrad** (params, accum=0.1, learning_rate=0.001, update_slots=True, loss_scale=1.0, weight_decay=0.0)

Adagrad 算法的实现。

Adagrad 用于在线学习和随机优化。请参阅论文 [Efficient Learning using Forward-Backward Splitting](#)。

Adagrad 可以根据不同参数的样本数量不均，自适应地给每个参数分配不同的学习率。

公式如下：

Parameters : learning rate γ , params w_0 , weight decay λ ,
initial accumulator value $state\_sum$

Init : $state\_sum_0 \leftarrow 0$

for $t = 1$ **to** ... **do**

$g_t \leftarrow \nabla_w f_t(w_{t-1})$

if $\lambda \neq 0$

$g_t \leftarrow g_t + \lambda w_{t-1}$

$state\_sum_t \leftarrow state\_sum_{t-1} + g_t^2$

$w_t \leftarrow w_{t-1} - \gamma * \frac{g_t}{\sqrt{state\_sum_t + \epsilon}}$

return w_t

$state\_sum$ 表示梯度平方的累加和 $accum$ 。 g 表示 $grads$ ， λ 代表 $weight\_decay$ 。 γ 代表 $learning\_rate$ ， w 代表 $params$ 。 t 代表当前的 $step$ 。

说明:

- 在参数未分组时, 优化器配置的 *weight_decay* 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 *weight_decay*。若未配置, 则该组网络参数使用优化器中配置的 *weight_decay*。

参数:

- **params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”:
 - **params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - **lr** - 可选。如果键中存在“lr”, 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在“weight_decay”, 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 *weight_decay* 作为权重衰减值。
 - **grad_centralization** - 可选。如果键中存在“grad_centralization”, 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 *grad_centralization* 为 False。该参数仅适用于卷积层。
 - **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时, 通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”, 则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **accum** (float, 可选) - 累加器 *h* 的初始值, 必须大于等于零。默认值: 0.1。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 默认值: 0.001。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中, 优化器将使用步数 (step) 作为输入, 调用 *LearningRateSchedule* 实例来计算当前学习率。
- **update_slots** (bool, 可选) - 如果为 True, 则更新累加器 *h*。默认值: True。
- **loss_scale** (float, 可选) - 梯度缩放系数, 必须大于 0.0。如果 *loss_scale* 是整数, 它将被转换为浮点数。通常使用默认值, 仅当训练时使用了 *FixedLossScaleManager*, 且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 False 时, 此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息, 请参阅 *mindspore.amp.FixedLossScaleManager*。默认值: 1.0。
- **weight_decay** (Union[float, int, Cell], 可选) - 要乘以权重的权重衰减值, 必须大于等于 0.0。默认值: 0.0。
 - **float**: 固定的权重衰减值。必须等于或大于 0。
 - **int**: 固定的权重衰减值。必须等于或大于 0。它将会被转换为 float 类型。
 - **Cell**: 权重衰减此时是动态的。在训练期间, 优化器调用该 Cell 的实例, 以获取当前阶段所要使用的权重衰减值。

输入:

- **grads** (tuple[*Tensor*]) - 优化器中 *params* 的梯度, 形状 (shape) 与 *params* 相同。

输出:

Tensor[bool], 值为 `True`。

异常:

- **TypeError** - *learning_rate* 不是 `int`、`float`、*Tensor*、*Iterable* 或 *LearningRateSchedule*。
- **TypeError** - *parameters* 的元素是 *Parameter* 或字典。
- **TypeError** - *accum* 或 *loss_scale* 不是 `float`。
- **TypeError** - *update_slots* 不是 `bool`。
- **TypeError** - *weight_decay* 不是 `float` 或 `int`。
- **ValueError** - *loss_scale* 小于或等于 0。
- **ValueError** - *accum* 或 *weight_decay* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import train
>>> import mindspore.nn as nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.Adagrad(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
>>>                  {'params': no_conv_params, 'lr': 0.01},
>>>                  {'order_params': net.trainable_params()}]
>>> optim = nn.Adagrad(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of
>>> ↪ 0.01 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight
>>> ↪ decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
>>> ↪ 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = train.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.3 mindspore.nn.Adam

class mindspore.nn.Adam(*params*, *learning_rate*=1e-3, *beta1*=0.9, *beta2*=0.999, *eps*=1e-8, *use_locking*=False, *use_nesterov*=False, *weight_decay*=0.0, *loss_scale*=1.0, *use_amsgrad*=False, ***kwargs*)

Adaptive Moment Estimation (Adam) 算法的实现。

Adam optimizer 可以使用梯度的 first-order moment estimation 和 second-order moment estimation，动态地调整每一个参数的学习率。

请参阅论文 [Adam: A Method for Stochastic Optimization](#)。

公式如下：

Parameters : 1st moment vector m , 2nd moment vector v ,

gradients g , learning rate γ , exponential decay rates for the moment estimates $\beta_1 \beta_2$,

parameter vector w_0 , timestep t , weight decay λ

Init : $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$, init parameter vector w_0

while w_t not converged **do**

$g_t \leftarrow \nabla_w f_t(w_{t-1})$

if $\lambda \neq 0$

$g_t \leftarrow g_t + \lambda w_{t-1}$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

$w_t \leftarrow w_{t-1} - \gamma \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

end while

return w_t

m 代表第一个动量矩阵, v 代表第二个动量矩阵, g 代表梯度 *gradients*, γ 代表学习率 *learning_rate*, β_1, β_2 代表衰减速率 *beta1* 和 *beta2*, t 代表当前 step, β_1^t 和 β_2^t 代表 *beta1* 和 *beta2* 的 t 次方, w 代表 *params*, ϵ 代表 *eps*。

说明:

- 如果前向网络使用了 `SparseGatherV2` 等算子, 优化器会执行稀疏运算。通过设置 `target` 为 `CPU`, 可在主机 (host) 上进行稀疏运算。稀疏特性在持续开发中。
- 在参数未分组时, 优化器配置的 `weight_decay` 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 `weight_decay`。若未配置, 则该组网络参数使用优化器中配置的 `weight_decay`。

当使用 Adam 时, 如果设置 `use_lazy=True`, 需要注意的是, 梯度稀疏时该优化器只更新网络参数的当前的索引位置, 稀疏行为不等同于 Adam 算法。如果想执行稀疏策略, 那么需要把 `target` 设置为 `CPU`。

当使用 Adam 时, 如果设置 `use_offload=True`, 需要注意的是, 此优化器目前仅支持图模式。

参数:

- **params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”:
 - **params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - **lr** - 可选。如果键中存在“lr”, 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 `learning_rate` 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在“weight_decay”, 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 `weight_decay` 作为权重衰减值。值得注意的是, `weight_decay` 可以是常量, 也可以是 `Cell` 类型。`Cell` 类型的 `weight decay` 用于实现动态 `weight decay` 算法。动态权重衰减和动态学习率相似, 用户需要自定义一个输入为 `global step` 的 `weight_decay_schedule`。在训练的过程中, 优化器会调用 `WeightDecaySchedule` 的实例来获取当前 `step` 的 `weight decay` 值。
 - **grad_centralization** - 可选。如果键中存在“grad_centralization”, 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 `grad_centralization` 为 `False`。该参数仅适用于卷积层。
 - **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时, 通常使用该配置项保持 `parameters` 的顺序以提升性能。如果键中存在“order_params”, 则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 `params` 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 默认值: `1e-3`。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 `i` 步将取向量中第 `i` 个值作为学习率。
 - **Iterable** - 动态的学习率。第 `i` 步将取迭代器第 `i` 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中, 优化器将使用步数 (`step`) 作为输入, 调用 `LearningRateSchedule` 实例来计算当前学习率。
- **beta1** (float, 可选) - 第一个动量矩阵的指数衰减率。参数范围 (0.0,1.0)。默认值: `0.9`。
- **beta2** (float, 可选) - 第二个动量矩阵的指数衰减率。参数范围 (0.0,1.0)。默认值: `0.999`。
- **eps** (float, 可选) - 加在分母上的值, 以确保数值稳定。必须大于 `0`。默认值: `1e-8`。

- **use_locking** (bool, 可选) - 是否对参数更新加锁保护。如果为 True，则 w 、 m 和 v 的 Tensor 更新将受到锁的保护。如果为 False，则结果不可预测。默认值: False。
- **use_nesterov** (bool, 可选) - 是否使用 Nesterov Accelerated Gradient (NAG) 算法更新梯度。如果为 True，使用 NAG 更新梯度。如果为 False，则在不使用 NAG 的情况下更新梯度。默认值: False。
- **use_amsgrad** (bool, 可选) - 是否使用 Amsgrad 算法更新梯度。如果为 True，使用 Amsgrad 更新梯度。如果为 False，则在不使用 Amsgrad 的情况下更新梯度。默认值: False。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减 (L2 penalty)。默认值: 0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 float。
 - **Cell**: 动态 weight decay。在训练过程中，优化器会使用步数 (step) 作为输入，调用该 Cell 实例来计算当前 weight decay 值。
- **loss_scale** (float, 可选) - 梯度缩放系数，必须大于 0.0。如果 *loss_scale* 是整数，它将被转换为浮点数。通常使用默认值，仅当训练时使用了 *FixedLossScaleManager*，且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 False 时，此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息，请参阅 [mindspore.amp.FixedLossScaleManager](#)。默认值: 1.0。
- **kwargs** -
 - **use_lazy** (bool): 是否使用 Lazy Adam 算法。默认值: False。如果为 True，使用 lazy Adam，反之使用普通 Adam 算法。
 - **use_offload** (bool): 是否在主机 CPU 上运行 Adam 优化算法。默认值: False。如果为 True，使用 offload 方法，反之使用普通 Adam 算法。

输入:

- **gradients** (tuple[Tensor]) - *params* 的梯度，形状 (shape) 与 *params* 相同。

输出:

Tensor[bool]，值为 True。

异常:

- **KeyError** - *kwargs* 接收到非 'use_lazy' 或 'use_offload' 的关键字。
- **TypeError** - *learning_rate* 不是 int、float、Tensor、iterable 或 LearningRateSchedule。
- **TypeError** - *params* 的元素不是 Parameter 或字典。
- **TypeError** - *beta1*、*beta2*、*eps* 或 *loss_scale* 不是 float。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **TypeError** - *use_locking*、*use_nesterov*、*use_amsgrad*、*use_lazy* 或 *use_offload* 不是 bool。
- **ValueError** - *loss_scale* 或 *eps* 小于或等于 0。
- **ValueError** - *beta1*、*beta2* 不在 (0.0, 1.0) 范围内。
- **ValueError** - *weight_decay* 小于 0。
- **ValueError** - *use_lazy* 和 *use_offload* 均为 True。
- **ValueError** - *use_amsgrad* 为 True 时，*use_lazy* 或 *use_offload* 亦为 True。
- **ValueError** - 在分布式训练时，*use_amsgrad* 为 True。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.Adam(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
>>>                  {'params': no_conv_params, 'lr': 0.01},
>>>                  {'order_params': net.trainable_params()}]
>>> optim = nn.Adam(group_params, learning_rate=0.1, weight_decay=0.0, use_lazy=False, use_
>>>                  offload=False)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of
>>> # 0.01 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight
>>> # decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
>>> # 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.4 mindspore.nn.AdaMax

class mindspore.nn.AdaMax (params, learning_rate=0.001, beta1=0.9, beta2=0.999, eps=1e-08, weight_decay=0.0, loss_scale=1.0)

AdaMax 算法是基于无穷范数的 Adam 的一种变体。

AdaMax 算法详情请参阅论文 [Adam: A Method for Stochastic Optimization](#)。

公式如下:

$$\begin{aligned} m_{t+1} &= \beta_1 * m_t + (1 - \beta_1) * g \\ v_{t+1} &= \max(\beta_2 * v_t, |g|) \\ w &= w - \frac{l}{1 - \beta_1^{t+1}} * \frac{m_{t+1}}{v_{t+1} + \epsilon} \end{aligned}$$

m 代表第一个动量矩阵, v 代表第二个动量矩阵, g 代表梯度 *gradients*, β_1, β_2 代表衰减速率 *beta1* 和 *beta2*, t 代表当前 step, β_1^t 代表 *beta1* 的 t 次方, l 代表学习率 *learning_rate*, w 代表 *params*, ϵ 代表 *eps*。

说明:

- 在参数未分组时, 优化器配置的 *weight_decay* 应用于名称不含“beta”或“gamma”的网络参数。

- 用户可以分组调整权重衰减策略。分组时，每组网络参数均可配置 `weight_decay`。若未配置，则该组网络参数使用优化器中配置的 `weight_decay`。

参数：

- **params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时，字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”：
 - **params** - 必填。当前组别的权重，该值必须是 *Parameter* 列表。
 - **lr** - 可选。如果键中存在“lr”，则使用对应的值作为学习率。如果没有，则使用优化器中的参数 `learning_rate` 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在“weight_decay”，则使用对应的值作为权重衰减值。如果没有，则使用优化器中配置的 `weight_decay` 作为权重衰减值。值得注意的是，`weight_decay` 可以是常量，也可以是 *Cell* 类型。*Cell* 类型的 `weight decay` 用于实现动态 `weight decay` 算法。动态权重衰减和动态学习率相似，用户需要自定义一个输入为 `global step` 的 `weight_decay_schedule`。在训练的过程中，优化器会调用 `WeightDecaySchedule` 的实例来获取当前 `step` 的 `weight decay` 值。
 - **grad_centralization** - 可选。如果键中存在“grad_centralization”，则使用对应的值，该值必须为布尔类型。如果没有，则认为 `grad_centralization` 为 `False`。该参数仅适用于卷积层。
 - **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 `parameters` 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 `params` 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule]) - 默认值：0.001。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率，第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中，优化器将使用步数 (`step`) 作为输入，调用 `LearningRateSchedule` 实例来计算当前学习率。
- **beta1** (float, 可选) - 第一个动量矩阵的指数衰减率。参数范围 (0.0,1.0)。默认值：0.9。
- **beta2** (float, 可选) - 第二个动量矩阵的指数衰减率。参数范围 (0.0,1.0)。默认值：0.999。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值：1e-08。
- **weight_decay** (Union[float, int, Cell]) - 权重衰减 (L2 penalty)。默认值：0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 `float`。
 - **Cell**: 动态 `weight decay`。在训练过程中，优化器会使用步数 (`step`) 作为输入，调用该 `Cell` 实例来计算当前 `weight decay` 值。
- **loss_scale** (float, 可选) - 梯度缩放系数，必须大于 0.0。如果 `loss_scale` 是整数，它将被转换为浮点数。通常使用默认值，仅当训练时使用了 `FixedLossScaleManager`，且 `FixedLossScaleManager` 的 `drop_overflow_update` 属性配置为 `False` 时，此值需要与 `FixedLossScaleManager` 中的 `loss_scale` 相同。有关更多详细信息，请参阅 `mindspore.amp.FixedLossScaleManager`。默认值：1.0。

输入:

- **gradients** (tuple[*Tensor*]) - *params* 的梯度, 形状 (shape) 与 *params* 相同。

输出:

Tensor[bool], 值为 True。

异常:

- **TypeError** - *learning_rate* 不是 int、float、*Tensor*、iterable 或 *LearningRateSchedule*。
- **TypeError** - *parameters* 的元素不是 *Parameter* 或字典。
- **TypeError** - *beta1*、*beta2*、*eps* 或 *loss_scale* 不是 float。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **ValueError** - *loss_scale* 或 *eps* 小于或等于 0。
- **ValueError** - *beta1*、*beta2* 不在 (0.0,1.0) 范围内。
- **ValueError** - *weight_decay* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.AdaMax(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
>>>                  {'params': no_conv_params, 'lr': 0.01},
>>>                  {'order_params': net.trainable_params()}]
>>> optim = nn.AdaMax(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of 0.01 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.5 mindspore.nn.AdamWeightDecay

class mindspore.nn.AdamWeightDecay (*params, learning_rate=1e-3, beta1=0.9, beta2=0.999, eps=1e-6, weight_decay=0.0*)

权重衰减 Adam 算法的实现。

Parameters : 1st moment vector m , 2nd moment vector v ,

gradients g , learning rate γ , exponential decay rates for the moment estimates $\beta_1 \beta_2$,

parameter vector w_0 , timestep t , weight decay λ

Init : $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $t \leftarrow 0$, init parameter vector w_0

repeat

$t \leftarrow t + 1$

$g_t \leftarrow \nabla f_t(w_{t-1})$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$w_t \leftarrow w_{t-1} - \gamma (m_t / (\sqrt{v_t} + \epsilon) + \lambda w_{t-1})$

until stopping criterion is met

return w_t

m 代表第一个动量矩阵 *moment1* , v 代表第二个动量矩阵 *moment2* , g 代表 *gradients* , γ 代表 *learning_rate* , β_1, β_2 代表 *beta1* 和 *beta2* , t 代表当前 step , w 代表 *params* , λ 代表 *weight_decay* 。

说明: 优化器和混合精度之间通常没有联系。但是, 当使用 *FixedLossScaleManager* 且 *FixedLossScaleManager* 中的 *drop_overflow_update* 设置为 *False* 时, 优化器需要设置 'loss_scale'。

由于此优化器没有 *loss_scale* 的参数, 因此需要通过其他方式处理 *loss_scale* 。

如何正确处理 *loss_scale* 详见 [LossScale](#)。

- 在参数未分组时, 优化器配置的 *weight_decay* 应用于名称不含 "beta" 或 "gamma" 的网络参数。
 - 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 *weight_decay* 。若未配置, 则该组网络参数使用优化器中配置的 *weight_decay* 。
-

参数：

- **params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时，字典的键可以是“params”、“lr”、“weight_decay”和“order_params”：
 - **params** - 必填。当前组别的权重，该值必须是 *Parameter* 列表。
 - **lr** - 可选。如果键中存在“lr”，则使用对应的值作为学习率。如果没有，则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在“weight_decay”，则使用对应的值作为权重衰减值。如果没有，则使用优化器中配置的 *weight_decay* 作为权重衰减值。值得注意的是，*weight_decay* 可以是常量，也可以是 *Cell* 类型。*Cell* 类型的 *weight decay* 用于实现动态 *weight decay* 算法。动态权重衰减和动态学习率相似，用户需要自定义一个输入为 *global step* 的 *weight_decay_schedule*。在训练的过程中，优化器会调用 *WeightDecaySchedule* 的实例来获取当前 *step* 的 *weight decay* 值。
 - **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule]) - 默认值：1e-3。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率，第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中，优化器将使用步数 (*step*) 作为输入，调用 *LearningRateSchedule* 实例来计算当前学习率。
- **beta1** (float) - *moment1* 的指数衰减率。参数范围 (0.0,1.0)。默认值：0.9。
- **beta2** (float) - *moment2* 的指数衰减率。参数范围 (0.0,1.0)。默认值：0.999。
- **eps** (float) - 将添加到分母中，以提高数值稳定性。必须大于 0。默认值：1e-6。
- **weight_decay** (Union[float, int, Cell]) - 权重衰减 (L2 penalty)。默认值：0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 *float*。
 - **Cell**: 动态 *weight decay*。在训练过程中，优化器会使用步数 (*step*) 作为输入，调用该 *Cell* 实例来计算当前 *weight decay* 值。

输入：

- **gradients** (tuple[Tensor]) - *params* 的梯度，*shape* 与 *params* 相同。

输出：

tuple[bool]，所有元素都为 True。

异常：

- **TypeError** - *learning_rate* 不是 int、float、Tensor、Iterable 或 *LearningRateSchedule*。
- **TypeError** - *parameters* 的元素不是 *Parameter* 或字典。
- **TypeError** - *beta1*、*beta2* 或 *eps* 不是 float。

- **TypeError** - *weight_decay* 不是 float 或 int。
- **ValueError** - *eps* 小于等于 0。
- **ValueError** - *beta1*、*beta2* 不在 (0.0,1.0) 范围内。
- **ValueError** - *weight_decay* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.AdamWeightDecay(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01},
...                 {'params': no_conv_params, 'lr': 0.01},
...                 {'order_params': net.trainable_params()}]
>>> optim = nn.AdamWeightDecay(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of 0.01.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight decay of 0.0.
>>> # The final parameters order in which the optimizer will be followed is the value of 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.6 mindspore.nn.AdaSumByDeltaWeightWrapCell

class mindspore.nn.AdaSumByDeltaWeightWrapCell(*optimizer*)

Adaptive Summation (AdaSum) 算法的实现，根据更新前后的参数差计算。应用于 semi_auto_parallel/auto_parallel 模式。

请参阅论文 [AdaSum: Scaling Distributed Training with Adaptive Summation](#)。

公式如下：

$$w_{t+1} = w_t - \alpha \cdot \text{Adasum}(g_1, g_2)$$

$$w_{t+1} = w_t - \alpha \cdot \left[\left(1 - \frac{g_2^T \cdot g_1}{2 \cdot \|g_1\|^2}\right) \cdot g_1 + \left(1 - \frac{g_1^T \cdot g_2}{2 \cdot \|g_2\|^2}\right) \cdot g_2 \right]$$

在本实现中，*g* 代表优化器更新前后的权重的变化量，下标代表数据并行维度下不同的设备。

说明：本接口推荐应用于半自动并行或者全自动并行模式。针对数据并行模式，推荐使用 mindspore.boost 功能以使用 AdaSum。使用本接口时，训练的卡的数量必须是 2 的幂，并且至少需要 16 张卡。目前，使用本接口时不支持优化器并行和流水线并行。

参数:

- **optimizer** (Union[Cell]) - 必须是单输入的优化器。

输入:

- **grads** (tuple[Tensor]) - *params* 的梯度, 形状 (shape) 与 *params* 相同, 与所传优化器的输入一致。

异常:

- **RuntimeError** - *parallel_mode* 使用了 *stand_alone* 模式, AdaSum 仅支持在分布式场景下使用。
- **RuntimeError** - 同时使用了优化器并行, 暂时不支持在优化器并行场景下使用 AdaSum。
- **RuntimeError** - 同时使用了流水线并行, 暂时不支持在流水线并行场景下使用 AdaSum。
- **RuntimeError** - *device_num* 不是 2 的幂, 或者小于 16。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> optim = nn.AdaSumByDeltaWeightWrapCell(nn.Momentum(params=net.trainable_params(),
...                                                    learning_rate=0.1, momentum=0.9))
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim, metrics=None)
```

3.15.7 mindspore.nn.AdaSumByGradWrapCell

class mindspore.nn.AdaSumByGradWrapCell (optimizer)

Adaptive Summation (AdaSum) 算法的实现, 根据梯度计算。应用于 semi_auto_parallel/auto_parallel 模式。

请参阅论文 [AdaSum: Scaling Distributed Training with Adaptive Summation](#)。

公式如下:

$$w_{t+1} = w_t - \alpha \cdot \text{Adasum}(g_1, g_2)$$

$$w_{t+1} = w_t - \alpha \cdot \left[\left(1 - \frac{g_2^T \cdot g_1}{2 \cdot \|g_1\|^2}\right) \cdot g_1 + \left(1 - \frac{g_1^T \cdot g_2}{2 \cdot \|g_2\|^2}\right) \cdot g_2 \right]$$

在本实现中, *g* 代表权重的梯度, 下标代表数据并行维度下不同的设备。

说明:

- 本接口推荐应用于半自动并行或者全自动并行模式。针对数据并行模式, 推荐使用 mindspore.boost 功能以使用 AdaSum。
- 使用本接口时, 训练的卡的数量必须是 2 的幂, 并且至少需要 16 张卡。目前, 使用本接口时不支持优化器并行和流水线并行。

参数:

- **optimizer** (Union[Cell]) - 用于更新权重的优化器。优化器的构造函数只允许一个输入。

输入：

- **grads** (tuple[Tensor]) - *params* 的梯度，*shape* 与 *params* 相同，与所传优化器的输入一致。

异常：

- **RuntimeError** - *parallel_mode* 使用了 *stand_alone* 模式，AdaSum 仅支持在分布式场景下使用。
- **RuntimeError** - 同时使用了优化器并行，暂时不支持在优化器并行场景下使用 AdaSum。
- **RuntimeError** - 同时使用了流水线并行，暂时不支持在流水线并行场景下使用 AdaSum。
- **RuntimeError** - *device_num* 不是 2 的幂，或者小于 16。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> optim = nn.AdaSumByGradWrapCell(nn.Momentum(params=net.trainable_params(), learning_
↪rate=0.1, momentum=0.9))
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim, metrics=None)
```

3.15.8 mindspore.nn.ASGD

class mindspore.nn.ASGD (*params*, *learning_rate*=0.1, *lambd*=1e-4, *alpha*=0.75, *t0*=1e6, *weight_decay*=0.0)

随机平均梯度下降 (ASGD) 算法的实现。

请参阅论文 [Acceleration of stochastic approximation by average](#) 。

更新公式如下：

$$\begin{aligned}w_t &= w_{t-1} * (1 - \lambda * \eta_{t-1}) - \eta_{t-1} * g_t \\ax_t &= (w_t - ax_{t-1}) * \mu_{t-1} \\ \eta_t &= \frac{1}{(1+\lambda*lr*t)^\alpha} \\ \mu_t &= \frac{1}{\max(1, t-t_0)}\end{aligned}$$

λ 代表衰减项， μ 和 η 被跟踪以更新 ax 和 w ， t_0 代表开始平均的点， α 代表 η 更新的系数， ax 表示平均参数值， t 表示当前步数 (step)， g 表示 *gradients*， w 表示 *params*。

说明：如果参数未分组，则优化器中的 *weight_decay* 将应用于名称中没有“beta”或“gamma”的参数。用户可以对参数进行分组，以更改权重衰减策略。当参数分组时，每个组都可以设置 *weight_decay*，如果没有，将应用优化器中的 *weight_decay*。

参数：

- **params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时，字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”：
 - **params** - 必填。当前组别的权重，该值必须是 *Parameter* 列表。

- **lr** - 可选。如果键中存在“lr”，则使用对应的值作为学习率。如果没有，则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
- **weight_decay** - 可选。如果键中存在“weight_decay”，则使用对应的值作为权重衰减值。如果没有，则使用优化器中配置的 *weight_decay* 作为权重衰减值。值得注意的是，*weight_decay* 可以是常量，也可以是 Cell 类型。Cell 类型的 *weight decay* 用于实现动态 *weight decay* 算法。动态权重衰减和动态学习率相似，用户需要自定义一个输入为 *global step* 的 *weight_decay_schedule*。在训练的过程中，优化器会调用 *WeightDecaySchedule* 的实例来获取当前 *step* 的 *weight decay* 值。
- **grad_centralization** - 可选。如果键中存在“grad_centralization”，则使用对应的值，该值必须为布尔类型。如果没有，则认为 *grad_centralization* 为 *False*。该参数仅适用于卷积层。
- **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule]) - 学习率。默认值：0.1。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率，第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中，优化器将使用步数 (*step*) 作为输入，调用 *LearningRateSchedule* 实例来计算当前学习率。
- **lambd** (float) - 衰减项。默认值：1e-4。
- **alpha** (float) - η 更新的系数。默认值：0.75。
- **t0** (float) - 开始平均的点。默认值：1e6。
- **weight_decay** (Union[float, int, Cell]) - 权重衰减 (L2 penalty)。默认值：0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 float。
 - **Cell**: 动态 *weight decay*。在训练过程中，优化器会使用步数 (*step*) 作为输入，调用该 Cell 实例来计算当前 *weight decay* 值。

输入：

- **gradients** (tuple[Tensor]) - *params* 的梯度，*shape* 与 *params* 相同。

输出：

Tensor[bool]，值为 *True*。

异常：

- **TypeError** - 如果 *learning_rate* 不是 int、float、Tensor、Iterable 或 LearningRateSchedule。
- **TypeError** - 如果 *parameters* 的元素不是 Parameter 或字典。
- **TypeError** - 如果 *lambd*、*alpha* 或 *t0* 不是 float。
- **TypeError** - 如果 *weight_decay* 既不是 float 也不是 int。
- **ValueError** - 如果 *weight_decay* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.ASGD(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'grad_centralization': True},
...                  {'params': no_conv_params, 'lr': 0.01},
...                  {'order_params': net.trainable_params()}]
>>> optim = nn.ASGD(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 default weight_
↳decay of 0.0 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight_
↳decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
↳'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.9 mindspore.nn.FTRL

class mindspore.nn.FTRL (params, initial_accum=0.1, learning_rate=0.001, lr_power=-0.5, l1=0.0, l2=0.0, use_locking=False, loss_scale=1.0, weight_decay=0.0)

FTRL 算法实现。

FTRL 是一种在线凸优化算法，根据损失函数自适应地选择正则化函数。详见论文 [Adaptive Bound Optimization for Online Convex Optimization](#)。

更新公式如下：

$$m_{t+1} = m_t + g^2$$

$$u_{t+1} = u_t + g - \frac{m_{t+1}^p - m_t^p}{\alpha} * \omega_t$$

$$\omega_{t+1} = \begin{cases} \frac{(\text{sign}(u_{t+1}) * l1 - u_{t+1})}{\frac{m_{t+1}^p}{\alpha} + 2 * l2} & \text{if } |u_{t+1}| > l1 \\ 0.0 & \text{otherwise} \end{cases}$$

m 表示累加器， g 表示 *grads*， t 表示当前 step， u 表示需要更新的线性系数， p 表示 *lr_power*， α 表示 *learning_rate*， ω 表示 *params*。

说明:

- 如果前向网络使用了 `SparseGatherV2` 等算子，优化器会执行稀疏运算。通过设置 `target` 为 `CPU`，可在主机（host）上进行稀疏运算。稀疏特性在持续开发中。
- 在参数未分组时，优化器配置的 `weight_decay` 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时，每组网络参数均可配置 `weight_decay`。若未配置，则该组网络参数使用优化器中配置的 `weight_decay`。

参数：

- **params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时，字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”：
 - **params** - 必填。当前组别的权重，该值必须是 *Parameter* 列表。
 - **lr** - 学习率当前不支持参数分组。
 - **weight_decay** - 可选。如果键中存在“weight_decay”，则使用对应的值作为权重衰减值。如果没有，则使用优化器中配置的 `weight_decay` 作为权重衰减值。值得注意的是，`weight_decay` 可以是常量，也可以是 *Cell* 类型。*Cell* 类型的 `weight_decay` 用于实现动态 `weight_decay` 算法。动态权重衰减和动态学习率相似，用户需要自定义一个输入为 `global step` 的 `weight_decay_schedule`。在训练的过程中，优化器会调用 `WeightDecaySchedule` 的实例来获取当前 `step` 的 `weight_decay` 值。
 - **grad_centralization** - 可选。如果键中存在“grad_centralization”，则使用对应的值，该值必须为布尔类型。如果没有，则认为 `grad_centralization` 为 `False`。该参数仅适用于卷积层。
 - **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **initial_accum** (float, 可选) - 累加器 *m* 的初始值，必须大于等于零。默认值：0.1。
- **learning_rate** (float, 可选) - 学习率值必须为零或正数，当前不支持动态学习率。默认值：0.001。
- **lr_power** (float, 可选) - 学习率的幂值，控制训练期间学习率的下降方式，必须小于或等于零。如果 `lr_power` 为零，则使用固定的学习率。默认值：-0.5。
- **l1** (float, 可选) - l1 正则化强度，必须大于等于零。默认值：0.0。
- **l2** (float, 可选) - l2 正则化强度，必须大于等于零。默认值：0.0。
- **use_locking** (bool, 可选) - 如果为 `True`，则更新操作使用锁保护。默认值：`False`。
- **loss_scale** (float, 可选) - 梯度缩放系数，必须大于 0.0。如果 `loss_scale` 是整数，它将被转换为浮点数。通常使用默认值，仅当训练时使用了 *FixedLossScaleManager*，且 *FixedLossScaleManager* 的 `drop_overflow_update` 属性配置为 `False` 时，此值需要与 *FixedLossScaleManager* 中的 `loss_scale` 相同。有关更多详细信息，请参阅 [mindspore.amp.FixedLossScaleManager](#)。默认值：1.0。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减（L2 penalty）。默认值：0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 `float`。
 - **Cell**: 动态 `weight_decay`。在训练过程中，优化器会使用步数（step）作为输入，调用该 *Cell* 实例来计算当前 `weight_decay` 值。

输入：

- **grads** (tuple[Tensor]) - 优化器中 *params* 的梯度, *shape* 与优化器中的 *params* 相同。

输出:

tuple[Parameter], 更新的参数, *shape* 与 *params* 相同。

异常:

- **TypeError** - *initial_accum*、*learning_rate*、*lr_power*、*l1*、*l2* 或 *loss_scale* 不是 float。
- **TypeError** - *parameters* 的元素不是 Parameter 或 dict。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **TypeError** - *use_nesterov* 不是 bool。
- **ValueError** - *lr_power* 大于 0。
- **ValueError** - *loss_scale* 小于等于 0。
- **ValueError** - *initial_accum*、*l1* 或 *l2* 小于 0。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.FTRL(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
>>>                  {'params': no_conv_params,
>>>                  {'order_params': net.trainable_params()}}]
>>> optim = nn.FTRL(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of
>>> 0.01 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use default learning rate of 0.1 will use default
>>> weight decay
>>> # of 0.0 and grad centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
>>> 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.10 mindspore.nn.Lamb

class mindspore.nn.Lamb (*params, learning_rate, beta1=0.9, beta2=0.999, eps=1e-6, weight_decay=0.0*)

LAMB (Layer-wise Adaptive Moments optimizer for Batch training, 用于批训练的分层自适应矩优化器) 算法的实现。

LAMB 是一种采用分层自适应批优化技术的优化算法。详见论文 [LARGE BATCH OPTIMIZATION FOR DEEP LEARNING: TRAINING BERT IN 76 MINUTES](#)。

LAMB 优化器旨在不降低精度的情况下增加训练 batch size，支持自适应逐元素更新和精确的分层校正。

参数更新如下：

Parameters : 1st moment vector m , 2nd moment vector v ,

learning rate $\{\gamma_t\}_{t=1}^T$, exponential decay rates for the moment estimates $\beta_1 \beta_2$,

scaling function ϕ

Init : $m_0 \leftarrow 0, v_0 \leftarrow 0$

for $t=1$ to T **do**

Draw b samples S_t from $\mathbb{P}$.

Compute $g_t = \frac{1}{|S_t|} \sum_{s_t \in S_t} \nabla \ell(x_t, s_t)$.

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

Compute ratio $r_t = \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

$w_{t+1}^{(i)} = w_t^{(i)} - \gamma_t \frac{\phi(\|w_t^{(i)}\|)}{\|r_t^{(i)} + \lambda w_t^{(i)}\|} (r_t^{(i)} + \lambda w_t^{(i)})$

end for

return w_{t+1}

其中, m 代表第一个动量矩阵 *moment1*, v 代表第二个动量矩阵 *moment2*, g 代表梯度 *gradients*, γ 代表学习率

$learning\_rate$, β_1, β_2 代表衰减速率 β_1 和 β_2 , t 代表当前 step, β_1^t 和 β_2^t 代表 β_1 和 β_2 的 t 次方, w 代表 $params$, ϵ 代表 eps , λ 表示 LAMB 权重衰减率, 对应 $weight\_decay$ 。

说明:

- 在参数未分组时, 优化器配置的 $weight\_decay$ 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 $weight\_decay$ 。若未配置, 则该组网络参数使用优化器中配置的 $weight\_decay$ 。

优化器和混合精度之间通常没有联系。但是, 当使用 `FixedLossScaleManager` 且 `FixedLossScaleManager` 中的 `drop_overflow_update` 设置为 `False` 时, 优化器需要设置 `loss_scale`。

由于此优化器没有 `loss_scale` 的参数, 因此需要通过其他方式处理 `loss_scale`。

如何正确处理 `loss_scale` 详见 [LossScale](#)。

警告: Lamb 优化器的更新过程不是完全 element-wise 的, 分布式并行下对权重进行切分可能会影响更新结果。

参数:

- **params** (Union[list[Parameter], list[dict]]) - 必须是 `Parameter` 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”:
 - **params** - 必填。当前组别的权重, 该值必须是 `Parameter` 列表。
 - **lr** - 可选。如果键中存在“lr”, 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 `learning_rate` 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在“weight_decay”, 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 `weight_decay` 作为权重衰减值。值得注意的是, `weight_decay` 可以是常量, 也可以是 `Cell` 类型。Cell 类型的 `weight decay` 用于实现动态 `weight decay` 算法。动态权重衰减和动态学习率相似, 用户需要自定义一个输入为 `global step` 的 `weight_decay_schedule`。在训练的过程中, 优化器会调用 `WeightDecaySchedule` 的实例来获取当前 step 的 `weight decay` 值。
 - **grad_centralization** - 可选。如果键中存在“grad_centralization”, 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 `grad_centralization` 为 `False`。该参数仅适用于卷积层。
 - **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时, 通常使用该配置项保持 `parameters` 的顺序以提升性能。如果键中存在“order_params”, 则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 `params` 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule]) -
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 i 步将取向量中第 i 个值作为学习率。
 - **Iterable** - 动态的学习率。第 i 步将取迭代器第 i 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中, 优化器将使用步数 (step) 作为输入, 调用 `LearningRateSchedule` 实例来计算当前学习率。
- **beta1** (float) - 第一矩的指数衰减率。参数范围 (0.0,1.0)。默认值: 0.9。

- **beta2** (float) - 第二矩的指数衰减率。参数范围 (0.0,1.0)。默认值: 0.999。
- **eps** (float) - 将添加到分母中, 以提高数值稳定性。必须大于 0。默认值: 1e-6。
- **weight_decay** (Union[float, int, Cell]) - 权重衰减 (L2 penalty)。默认值: 0.0。
 - **float**: 固定值, 必须大于或者等于 0。
 - **int**: 固定值, 必须大于或者等于 0, 会被转换成 float。
 - **Cell**: 动态 weight decay。在训练过程中, 优化器会使用步数 (step) 作为输入, 调用该 Cell 实例来计算当前 weight decay 值。

输入:

- **gradients** (tuple[Tensor]) - *params* 的梯度, shape 与 *params* 相同。

输出:

tuple[bool], 所有元素都为 True。

异常:

- **TypeError** - *learning_rate* 不是 int、float、Tensor、Iterable 或 LearningRateSchedule。
- **TypeError** - *parameters* 的元素不是 Parameter 或 dict。
- **TypeError** - *beta1*、*beta2* 或 *eps* 不是 float。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **ValueError** - *eps* 小于等于 0。
- **ValueError** - *beta1*、*beta2* 不在 (0.0,1.0) 范围内。
- **ValueError** - *weight_decay* 小于 0。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.Lamb(params=net.trainable_params(), learning_rate=0.1)
>>>
>>> #2) Use parameter groups and set different values
>>> poly_decay_lr = nn.PolynomialDecayLR(learning_rate=0.1, end_learning_rate=0.01,
...                                     decay_steps=4, power = 0.5)
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
...                 {'params': no_conv_params, 'lr': poly_decay_lr,
...                  'order_params': net.trainable_params(0.01)}]
>>> optim = nn.Lamb(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of
>>> # 0.01 and grad
>>> # centralization of True.
```

(下页继续)

(续上页)

```
>>> # The no_conv_params's parameters will use dynamic learning rate of poly decay learning.
↳rate and default
>>> # weight decay of 0.0 and grad centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
↳'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.11 mindspore.nn.LARS

class mindspore.nn.LARS (optimizer, epsilon=1e-05, coefficient=0.001, use_clip=False, lars_filter=lambda x: ...)

LARS 算法的实现。

LARS 算法采用大量的优化技术。详见论文 [LARGE BATCH TRAINING OF CONVOLUTIONAL NETWORKS](#)。

更新公式如下：

Parameters : base learning rate γ_0 , momentum m , weight decay λ ,
LARS coefficient η , number of steps T
Init : $t=0$, $v=0$, init weight w_0^l for each layer l

```
while t<T for each layer l do
     $g_t^l \leftarrow \nabla L(w_t^l)$ 
     $\gamma_t \leftarrow \gamma_0 * (1 - \frac{t}{T})^2$ 
     $\gamma^l \leftarrow \eta * \frac{\|w_t^l\|}{\|g_t^l\| + \lambda \|w_t^l\|}$  (compute the local LR  $\gamma^l$ )
     $v_{t+1}^l \leftarrow mv_t^l + \gamma_{t+1} * \gamma^l * (g_t^l + \lambda w_t^l)$ 
     $w_{t+1}^l \leftarrow w_t^l - v_{t+1}^l$ 
end while
```

w 表示网络中的 param, g 表示 gradients, t 表示当前 step, λ 表示 optimizer 配置的 weight_decay, γ 表示 optimizer 配置的 learning_rate, η 表示 coefficient。

参数：

- **optimizer** (mindspore.nn.Optimizer) - 待封装和修改梯度的 MindSpore 优化器。
- **epsilon** (float) - 将添加到分母中，提高数值稳定性。默认值：1e-05。
- **coefficient** (float) - 计算局部学习速率的信任系数。默认值：0.001。
- **use_clip** (bool) - 计算局部学习速率时是否裁剪。默认值：False。
- **lars_filter** (Function) - 用于指定使用 LARS 算法的网络参数。默认值：lambda x: 'LayerNorm' not in x.name and 'bias' not in x.name。

输入：

- **gradients** (tuple[Tensor]) - 优化器中 params 的梯度，shape 与优化器中的 params 相同。

支持平台：

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> opt = nn.Momentum(net.trainable_params(), 0.1, 0.9)
>>> opt_lars = nn.LARS(opt, epsilon=1e-08, coefficient=0.02)
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=opt_lars, metrics=None)
```

3.15.12 mindspore.nn.LazyAdam

class mindspore.nn.LazyAdam (params, learning_rate=1e-3, beta1=0.9, beta2=0.999, eps=1e-8, use_locking=False, use_nesterov=False, weight_decay=0.0, loss_scale=1.0)

Adaptive Moment Estimation (Adam) 算法的实现。请参阅论文 [Adam: A Method for Stochastic Optimization](#)。

当梯度稀疏时，此优化器将使用 Lazy Adam 算法。

更新公式如下：

$$\begin{aligned} m_{t+1} &= \beta_1 * m_t + (1 - \beta_1) * g \\ v_{t+1} &= \beta_2 * v_t + (1 - \beta_2) * g * g \\ \widehat{m}_{t+1} &= \frac{m_{t+1}}{1 - \beta_1^t} \\ \widehat{v}_{t+1} &= \frac{v_{t+1}}{1 - \beta_2^t} \\ w_{t+1} &= w_t - \gamma * \frac{\widehat{m}_{t+1}}{\sqrt{\widehat{v}_{t+1} + \epsilon}} \end{aligned}$$

m 代表一阶矩变量 *moment1*， v 代表二阶矩变量 *moment2*， g 代表梯度 *gradients*， γ 代表学习率 *learning_rate*， β_1, β_2 代表衰减速率 *beta1* 和 *beta2*， t 代表当前 *step*， β_1^t 和 β_2^t 代表 *beta1_power* 和 *beta2_power*， w 代表 *params*， ϵ 代表 *eps*。

说明：

- 如果前向网络使用了 SparseGatherV2 等算子，优化器会执行稀疏运算。通过设置 *target* 为 CPU，可在主机 (host) 上进行稀疏运算。稀疏特性在持续开发中。

需要注意的是，梯度稀疏时，该优化器只更新网络参数当前的索引位置，稀疏行为不等同于 Adam 算法。如果想执行稀疏策略，那么需要把 *target* 设置为 CPU。

- 在参数未分组时，优化器配置的 *weight_decay* 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时，每组网络参数均可配置 *weight_decay*。若未配置，则该组网络参数使用优化器中配置的 *weight_decay*。

参数：

- params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时，字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”：
 - **params** - 必填。当前组别的权重，该值必须是 *Parameter* 列表。

- **lr** - 可选。如果键中存在“lr”，则使用对应的值作为学习率。如果没有，则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
- **weight_decay** - 可选。如果键中存在“weight_decay”，则使用对应的值作为权重衰减值。如果没有，则使用优化器中配置的 *weight_decay* 作为权重衰减值。值得注意的是，*weight_decay* 可以是常量，也可以是 Cell 类型。Cell 类型的 *weight_decay* 用于实现动态 *weight_decay* 算法。动态权重衰减和动态学习率相似，用户需要自定义一个输入为 *global step* 的 *weight_decay_schedule*。在训练的过程中，优化器会调用 *WeightDecaySchedule* 的实例来获取当前 *step* 的 *weight_decay* 值。
- **grad_centralization** - 可选。如果键中存在“grad_centralization”，则使用对应的值，该值必须为布尔类型。如果没有，则认为 *grad_centralization* 为 *False*。该参数仅适用于卷积层。
- **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 默认值: 1e-3。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率，第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中，优化器将使用步数 (*step*) 作为输入，调用 *LearningRateSchedule* 实例来计算当前学习率。
- **beta1** (float, 可选) - *moment1* 的指数衰减率。参数范围 (0.0,1.0)。默认值: 0.9。
- **beta2** (float, 可选) - *moment2* 的指数衰减率。参数范围 (0.0,1.0)。默认值: 0.999。
- **eps** (float, 可选) - 将添加到分母中，以提高数值稳定性。必须大于 0。默认值: 1e-8。
- **use_locking** (bool, 可选) - 是否对参数更新加锁保护。如果为 *True*，则 *w*、*m* 和 *v* 的 *Tensor* 更新将受到锁的保护。如果为 *False*，则结果不可预测。默认值: *False*。
- **use_nesterov** (bool, 可选) - 是否使用 Nesterov Accelerated Gradient (NAG) 算法更新梯度。如果为 *True*，使用 NAG 更新梯度。如果为 *False*，则在不使用 NAG 的情况下更新梯度。默认值: *False*。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减 (L2 penalty)。默认值: 0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 *float*。
 - **Cell**: 动态 *weight_decay*。在训练过程中，优化器会使用步数 (*step*) 作为输入，调用该 *Cell* 实例来计算当前 *weight_decay* 值。
- **loss_scale** (float, 可选) - 梯度缩放系数，必须大于 0.0。如果 *loss_scale* 是整数，它将被转换为浮点数。通常使用默认值，仅当训练时使用了 *FixedLossScaleManager*，且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 *False* 时，此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息，请参阅 *mindspore.amp.FixedLossScaleManager*。默认值: 1.0。

输入:

- **gradients** (tuple[Tensor]) - *params* 的梯度，*shape* 与 *params* 相同。

输出:

Tensor[bool], 值为 True。

异常：

- **TypeError** - *learning_rate* 不是 int、float、Tensor、Iterable 或 *LearningRateSchedule*。
- **TypeError** - *parameters* 的元素不是 Parameter 或字典。
- **TypeError** - *beta1*、*beta2*、*eps* 或 *loss_scale* 不是 float。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **TypeError** - *use_locking* 或 *use_nesterov* 不是 bool。
- **ValueError** - *loss_scale* 或 *eps* 小于或等于 0。
- **ValueError** - *beta1*、*beta2* 不在 (0.0,1.0) 范围内。
- **ValueError** - *weight_decay* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.LazyAdam(parameters=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
>>>                  {'params': no_conv_params, 'lr': 0.01},
>>>                  {'order_params': net.trainable_params()}]
>>> optim = nn.LazyAdam(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of_
>>> ↪ 0.01 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight_
>>> ↪ decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
>>> ↪ 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.13 mindspore.nn.Momentum

class mindspore.nn.Momentum (params, learning_rate, momentum, weight_decay=0.0, loss_scale=1.0, use_nesterov=False)
Momentum 算法的实现。

有关更多详细信息, 请参阅论文 [On the importance of initialization and momentum in deep learning](#)。

$$v_{t+1} = v_t * u + grad$$

如果 `use_nesterov` 为 `True` :

$$p_{t+1} = p_t - (grad * lr + v_{t+1} * u * lr)$$

如果 `use_nesterov` 为 `False` :

$$p_{t+1} = p_t - lr * v_{t+1}$$

其中, `grad`、`lr`、`p`、`v` 和 `u` 分别表示梯度、学习率、参数、矩 (Moment) 和动量 (Momentum)。

说明:

- 在参数未分组时, 优化器配置的 `weight_decay` 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 `weight_decay`。若未配置, 则该组网络参数使用优化器中配置的 `weight_decay`。

参数:

- params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是“params”、“lr”、“weight_decay”、“grad_centralization”和“order_params”:
 - params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - lr** - 可选。如果键中存在“lr”, 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 `learning_rate` 作为学习率。支持固定和动态学习率。
 - weight_decay** - 可选。如果键中存在“weight_decay”, 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 `weight_decay` 作为权重衰减值。值得注意的是, `weight_decay` 可以是常量, 也可以是 *Cell* 类型。*Cell* 类型的 `weight decay` 用于实现动态 `weight decay` 算法。动态权重衰减和动态学习率相似, 用户需要自定义一个输入为 `global step` 的 `weight_decay_schedule`。在训练的过程中, 优化器会调用 `WeightDecaySchedule` 的实例来获取当前 `step` 的 `weight decay` 值。
 - grad_centralization** - 可选。如果键中存在“grad_centralization”, 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 `grad_centralization` 为 `False`。该参数仅适用于卷积层。
 - order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时, 通常使用该配置项保持 `parameters` 的顺序以提升性能。如果键中存在“order_params”, 则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 `params` 参数中。
- learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) -
 - float** - 固定的学习率。必须大于等于零。
 - int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 *i* 步将取向量中第 *i* 个值作为学习率。

- **Iterable** - 动态的学习率。第 i 步将取迭代器第 i 个值作为学习率。
- **LearningRateSchedule** - 动态的学习率。在训练过程中，优化器将使用步数 (step) 作为输入，调用 `LearningRateSchedule` 实例来计算当前学习率。
- **momentum** (float, 可选) - 浮点数类型的超参，表示移动平均的动量。必须等于或大于 0.0。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减 (L2 penalty)。默认值: 0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 float。
 - **Cell**: 动态 weight decay。在训练过程中，优化器会使用步数 (step) 作为输入，调用该 Cell 实例来计算当前 weight decay 值。
- **loss_scale** (float, 可选) - 梯度缩放系数，必须大于 0.0。如果 `loss_scale` 是整数，它将被转换为浮点数。通常使用默认值，仅当训练时使用了 `FixedLossScaleManager`，且 `FixedLossScaleManager` 的 `drop_overflow_update` 属性配置为 `False` 时，此值需要与 `FixedLossScaleManager` 中的 `loss_scale` 相同。有关更多详细信息，请参阅 `mindspore.amp.FixedLossScaleManager`。默认值: 1.0。
- **use_nesterov** (bool, 可选) - 是否使用 Nesterov Accelerated Gradient (NAG) 算法更新梯度。默认值: `False`。

输入:

- **gradients** (tuple[Tensor]) - `params` 的梯度，形状 (shape) 与 `params` 相同。

输出:

tuple[bool], 所有元素都为 `True`。

异常:

- **TypeError** - `learning_rate` 不是 int、float、Tensor、Iterable 或 `LearningRateSchedule`。
- **TypeError** - `parameters` 的元素不是 `Parameter` 或字典。
- **TypeError** - `loss_scale` 或 `momentum` 不是 float。
- **TypeError** - `weight_decay` 不是 float 或 int。
- **TypeError** - `use_nesterov` 不是 bool。
- **ValueError** - `loss_scale` 小于或等于 0。
- **ValueError** - `weight_decay` 或 `momentum` 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
```

(下页继续)

(续上页)

```

>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True}
    ↪,
    ...           {'params': no_conv_params, 'lr': 0.01},
    ...           {'order_params': net.trainable_params()}]
>>> optim = nn.Momentum(group_params, learning_rate=0.1, momentum=0.9, weight_decay=0.0)
>>> # The conv_params's parameters will use a learning rate of default value 0.1 and a
    ↪weight decay of 0.01 and
>>> # grad centralization of True.
>>> # The no_conv_params's parameters will use a learning rate of 0.01 and a weight decay of
    ↪default value 0.0
>>> # and grad centralization of False..
>>> # The final parameters order in which the optimizer will be followed is the value of
    ↪'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim, metrics=None)

```

3.15.14 mindspore.nn.OptTFTWrapper

class mindspore.nn.OptTFTWrapper (opt, **kwargs)

实现 TFT 优化器封装器。该封装器将在优化器更新前向 MindIO TFT 上报状态。

说明：该优化器依赖于 MindIO TFT 特性。当前只支持 Ascend 后端的图模式，并且 sink_size 的配置必须小于等于 1。

参数：

- **opt** (Optimizer) - 该参数必须为 Optimizer 的子类。

输入：

- **gradients** (tuple[Tensor]) - 参数 opt 的 *params* 的梯度，shape 与 opt 的 *params* shape 相同。

输出：

Tensor，优化器 opt 执行返回的结果。

异常：

- **TypeError** - 如果 opt 不是 Optimizer 的子类。
- **ValueError** - 如果不是运行在 Ascend 后端的图模式，或者用户不开启 TFT 特性。

支持平台：

Ascend

样例：

```

>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay

```

(下页继续)

(续上页)

```
>>> optim = nn.SGD(params=net.trainable_params())
>>> optim_wrapper = nn.OptTFTWrapper(optim)
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.15 mindspore.nn.ProximalAdagrad

class mindspore.nn.ProximalAdagrad (params, accum=0.1, learning_rate=0.001, l1=0.0, l2=0.0, use_locking=False, loss_scale=1.0, weight_decay=0.0)

ProximalAdagrad 算法的实现, 用于在线学习和随机优化。请参阅论文 [Efficient Learning using Forward-Backward Splitting](#)。

$$accum_{t+1} = accum_t + g * g$$

$$prox\_v = w_t - \gamma * g * \frac{1}{\sqrt{accum_{t+1}}}$$

$$w_{t+1} = \frac{sign(prox\_v)}{1 + \gamma * l2} * \max(|prox\_v| - \gamma * l1, 0)$$

其中, g 、 γ 、 w 、 $accum$ 和 t 分别表示 *grads*、*learning_rate*、*params*、累加器和当前 step。

说明:

- 如果前向网络使用了 `SparseGatherV2` 等算子, 优化器会执行稀疏运算。通过设置 *target* 为 CPU, 可在主机 (host) 上进行稀疏运算。稀疏特性在持续开发中。
- 在参数未分组时, 优化器配置的 *weight_decay* 应用于名称不含 "beta" 或 "gamma" 的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 *weight_decay*。若未配置, 则该组网络参数使用优化器中配置的 *weight_decay*。

参数:

- **params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是 "params"、"lr"、"weight_decay"、"grad_centralization" 和 "order_params":
 - **params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - **lr** - 可选。如果键中存在 "lr", 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在 "weight_decay", 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 *weight_decay* 作为权重衰减值。值得注意的是, *weight_decay* 可以是常量, 也可以是 Cell 类型。Cell 类型的 *weight_decay* 用于实现动态 *weight_decay* 算法。动态权重衰减和动态学习率相似, 用户需要自定义一个输入为 global step 的 *weight_decay_schedule*。在训练的过程中, 优化器会调用 *WeightDecaySchedule* 的实例来获取当前 step 的 *weight_decay* 值。
 - **grad_centralization** - 可选。如果键中存在 "grad_centralization", 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 *grad_centralization* 为 False。该参数仅适用于卷积层。

- **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **accum** (float, 可选) - 累加器 *accum* 的初始值，起始值必须为零或正值。默认值：0.1。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 默认值：0.001。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率，第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中，优化器将使用步数 (step) 作为输入，调用 [LearningRateSchedule](#) 实例来计算当前学习率。
- **l1** (float, 可选) - l1 正则化强度，必须大于或等于零。默认值：0.0。
- **l2** (float, 可选) - l2 正则化强度，必须大于或等于零。默认值：0.0。
- **use_locking** (bool, 可选) - 如果为 True，则更新操作使用锁保护。默认值：False。
- **loss_scale** (float, 可选) - 梯度缩放系数，必须大于 0.0。如果 *loss_scale* 是整数，它将被转换为浮点数。通常使用默认值，仅当训练时使用了 *FixedLossScaleManager*，且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 False 时，此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息，请参阅 [mindspore.amp.FixedLossScaleManager](#)。默认值：1.0。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减 (L2 penalty)。默认值：0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 float。
 - **Cell**: 动态 weight decay。在训练过程中，优化器会使用步数 (step) 作为输入，调用该 Cell 实例来计算当前 weight decay 值。

输入：

- **grads** (tuple[Tensor]) - 优化器中 *params* 的梯度，shape 与优化器中的 *params* 相同。

输出：

Tensor[bool]，值为 True。

异常：

- **TypeError** - *learning_rate* 不是 int、float、Tensor、Iterable 或 LearningRateSchedule。
- **TypeError** - *parameters* 的元素不是 Parameter 或字典。
- **TypeError** - *accum*、*l1*、*l2* 或 *loss_scale* 不是 float。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **ValueError** - *loss_scale* 小于或等于 0。
- **ValueError** - *accum*、*l1*、*l2* 或 *weight_decay* 小于 0。

支持平台：

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.ProximalAdagrad(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
>>> ...,
>>>                    {'params': no_conv_params, 'lr': 0.01},
>>>                    {'order_params': net.trainable_params()}]
>>> optim = nn.ProximalAdagrad(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay
>>> of 0.01 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight
>>> decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
>>> 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)
```

3.15.16 mindspore.nn.RMSProp

class mindspore.nn.**RMSProp** (*params, learning_rate=0.1, decay=0.9, momentum=0.0, epsilon=1e-10, use_locking=False, centered=False, loss_scale=1.0, weight_decay=0.0*)

均方根传播 (RMSProp) 算法的实现。

根据 RMSProp 算法更新 *params*。算法详见 讲义 第 29 页。

公式如下:

$$s_{t+1} = \rho s_t + (1 - \rho)(\nabla Q_i(w))^2$$

$$m_{t+1} = \beta m_t + \frac{\eta}{\sqrt{s_{t+1} + \epsilon}} \nabla Q_i(w)$$

$$w = w - m_{t+1}$$

第一个方程计算每个权重的平方梯度的移动平均。然后将梯度除以 $\sqrt{\rho s_{t+1} + \epsilon}$ 。

如果 *centered* 为 *True*:

$$g_{t+1} = \rho g_t + (1 - \rho) \nabla Q_i(w)$$

$$s_{t+1} = \rho s_t + (1 - \rho)(\nabla Q_i(w))^2$$

$$m_{t+1} = \beta m_t + \frac{\eta}{\sqrt{s_{t+1} - g_{t+1}^2 + \epsilon}} \nabla Q_i(w)$$

$$w = w - m_{t+1}$$

其中 w 代表待更新的网络参数 $params$ 。 g_{t+1} 是平均梯度。 s_{t+1} 是均方梯度。 m_{t+1} 是 moment, w 的 delta。 ρ 代表 $decay$ 。 β 是动量项, 表示 $momentum$ 。 ϵ 是平滑项, 可以避免除以零, 表示 $epsilon$ 。 η 是学习率, 表示 $learning\_rate$ 。 $\nabla Q_i(w)$ 是梯度, 表示 $gradients$ 。 t 表示当前 step。

说明:

- 在参数未分组时, 优化器配置的 `weight_decay` 应用于名称不含 "beta" 或 "gamma" 的网络参数。
- 用户可以分组调整权重衰减策略。分组时, 每组网络参数均可配置 `weight_decay`。若未配置, 则该组网络参数使用优化器中配置的 `weight_decay`。

参数:

- params** (Union[list[Parameter], list[dict]]) - 必须是 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时, 字典的键可以是 "params"、"lr"、"weight_decay"、"grad_centralization" 和 "order_params":
 - params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - lr** - 可选。如果键中存在 "lr", 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 `learning_rate` 作为学习率。支持固定和动态学习率。
 - weight_decay** - 可选。如果键中存在 "weight_decay", 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 `weight_decay` 作为权重衰减值。值得注意的是, `weight_decay` 可以是常量, 也可以是 Cell 类型。Cell 类型的 `weight decay` 用于实现动态 `weight decay` 算法。动态权重衰减和动态学习率相似, 用户需要自定义一个输入为 `global step` 的 `weight_decay_schedule`。在训练的过程中, 优化器会调用 `WeightDecaySchedule` 的实例来获取当前 step 的 `weight decay` 值。
 - grad_centralization** - 可选。如果键中存在 "grad_centralization", 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 `grad_centralization` 为 `False`。该参数仅适用于卷积层。
 - order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时, 通常使用该配置项保持 `parameters` 的顺序以提升性能。如果键中存在 "order_params", 则会忽略该组配置中的其他键。"order_params" 中的参数必须在某一组 `params` 参数中。
- learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 默认值: 0.1。
 - float** - 固定的学习率。必须大于等于零。
 - int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 i 步将取向量中第 i 个值作为学习率。
 - Iterable** - 动态的学习率。第 i 步将取迭代器第 i 个值作为学习率。
 - LearningRateSchedule** - 动态的学习率。在训练过程中, 优化器将使用步数 (step) 作为输入, 调用 `LearningRateSchedule` 实例来计算当前学习率。
- decay** (float, 可选) - 衰减率。必须大于等于 0。默认值: 0.9。
- momentum** (float, 可选) - float 类型的超参数, 表示移动平均的动量 (momentum)。必须大于等于 0。默认值: 0.0。
- epsilon** (float, 可选) - 将添加到分母中, 以提高数值稳定性。取值大于 0。默认值: $1e-10$ 。
- use_locking** (bool, 可选) - 是否对参数更新加锁保护。默认值: `False`。
- centered** (bool, 可选) - 如果为 `True`, 则梯度将通过梯度的估计方差进行归一。默认值: `False`。

- **loss_scale** (float, 可选) - 梯度缩放系数, 必须大于 0.0。如果 *loss_scale* 是整数, 它将被转换为浮点数。通常使用默认值, 仅当训练时使用了 *FixedLossScaleManager*, 且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 *False* 时, 此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息, 请参阅 *mindspore.amp.FixedLossScaleManager*。默认值: 1.0。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减 (L2 penalty)。默认值: 0.0。
 - **float**: 固定值, 必须大于或者等于 0。
 - **int**: 固定值, 必须大于或者等于 0, 会被转换成 float。
 - **Cell**: 动态 weight decay。在训练过程中, 优化器会使用步数 (step) 作为输入, 调用该 Cell 实例来计算当前 weight decay 值。

输入:

- **gradients** (tuple[Tensor]) - *params* 的梯度, shape 与 *params* 相同。

输出:

Tensor[bool], 值为 True。

异常:

- **TypeError** - *learning_rate* 不是 int、float、Tensor、Iterable 或 LearningRateSchedule。
- **TypeError** - *decay*、*momentum*、*epsilon* 或 *loss_scale* 不是 float。
- **TypeError** - *parameters* 的元素不是 Parameter 或字典。
- **TypeError** - *weight_decay* 不是 float 或 int。
- **TypeError** - *use_locking* 或 *centered* 不是 bool。
- **ValueError** - *epsilon* 小于或等于 0。
- **ValueError** - *decay* 或 *momentum* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.RMSProp(params=net.trainable_params(), learning_rate=0.1)
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'weight_decay': 0.01, 'grad_centralization': True},
>>>                  {'params': no_conv_params, 'lr': 0.01},
>>>                  {'order_params': net.trainable_params()}]
>>> optim = nn.RMSProp(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and weight decay of_
>>> 0.01 and grad
```

(下页继续)

(续上页)

```

>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight_
    ↳decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
    ↳'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)

```

3.15.17 mindspore.nn.Rprop

class mindspore.nn.Rprop (params, learning_rate=0.1, etas=(0.5, 1.2), step_sizes=(1e-6, 50.), weight_decay=0.)

弹性反向传播 (Rprop) 算法的实现。

请参阅论文 [A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm.](#) 。

更新公式如下：

$$\begin{aligned}
 & \text{if } g_{t-1}g_t > 0 \\
 & \quad \Delta_t \leftarrow \min(\Delta_{t-1}\eta_+, \Delta_{\max}) \\
 & \text{else if } g_{t-1}g_t < 0 \\
 & \quad \Delta_t \leftarrow \max(\Delta_{t-1}\eta_-, \Delta_{\min}) \\
 & \text{else} \\
 & \quad \Delta_t \leftarrow \Delta_{t-1} \\
 & \quad w_t \leftarrow w_{t-1} - \Delta_t \text{sign}(g_t)
 \end{aligned}$$

g 表示 *gradients* , w 表示 *parameters* , $\Delta_{\min/\max}$ 表示最小或者最大步长, $\eta_{+/-}$ 表示加速和减速因子。

说明：

- 在参数未分组时，优化器配置的 `weight_decay` 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时，每组网络参数均可配置 `weight_decay` 。若未配置，则该组网络参数使用优化器中配置的 `weight_decay` 。

参数：

- params** (Union[list[Parameter], list[dict]]) - 需为 *Parameter* 组成的列表或字典组成的列表。当列表元素是字典时，字典的键可以是 “params”、“lr”、“weight_decay”、“grad_centralization” 和 “order_params”：
 - params** - 必填。当前组别的权重，该值必须是 *Parameter* 列表。
 - lr** - 可选。如果键中存在“lr”，则使用对应的值作为学习率。如果没有，则使用优化器中的参数 `learning_rate` 作为学习率。支持固定和动态学习率。
 - weight_decay** - 可选。如果键中存在“weight_decay”，则使用对应的值作为权重衰减值。如果没有，则使用优化器中配置的 `weight_decay` 作为权重衰减值。值得注意的是，`weight_decay` 可以是常量，也可以是 *Cell* 类型。*Cell* 类型的 `weight decay` 用于实现动态 `weight decay` 算法。动态权重衰减和动态学习率相似，用户需要自定义一个输入为 `global step` 的 `weight_decay_schedule`。在训练的过程中，优化器会调用 `WeightDecaySchedule` 的实例来获取当前 `step` 的 `weight decay` 值。
 - grad_centralization** - 可选。如果键中存在“grad_centralization”，则使用对应的值，该值必须为布尔类型。如果没有，则认为 `grad_centralization` 为 `False`。该参数仅适用于卷积层。

- **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时，通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在“order_params”，则会忽略该组配置中的其他键。“order_params”中的参数必须在某一组 *params* 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 学习率。默认值：0.1。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率，第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中，优化器将使用步数 (step) 作为输入，调用 [LearningRateSchedule](#) 实例来计算当前学习率。
- **etas** (tuple[float, float], 可选) - 乘法的增加或减少的因子 (etaminus, etaplus)。默认值：(0.5, 1.2)。
- **step_sizes** (tuple[float, float], 可选) - 允许的最小和最大步长 (min_step_sizes, max_step_size)。默认值：(1e-6, 50.)。
- **weight_decay** (Union[float, int, Cell], 可选) - 权重衰减 (L2 penalty)。默认值：0.0。
 - **float**: 固定值，必须大于或者等于 0。
 - **int**: 固定值，必须大于或者等于 0，会被转换成 float。
 - **Cell**: 动态 weight decay。在训练过程中，优化器会使用步数 (step) 作为输入，调用该 Cell 实例来计算当前 weight decay 值。

输入：

- **gradients** (tuple[Tensor]) - *params* 的梯度，shape 与 *params* 相同。

输出：

Tensor[bool]，值为 True。

异常：

- **TypeError** - 如果 *learning_rate* 不是 int、float、Tensor、Iterable 或 LearningRateSchedule。
- **TypeError** - 如果 *parameters* 的元素不是 Parameter 或字典。
- **TypeError** - 如果 *step_sizes* 或 *etas* 不是 tuple。
- **ValueError** - 如果最大步长小于最小步长。
- **ValueError** - 如果 *step_sizes* 或 *etas* 的长度不等于 2。
- **TypeError** - 如果 *etas* 或 *step_sizes* 中的元素不是 float。
- **ValueError** - 如果 *etaminus* 不在 (0, 1) 的范围内，或者 *etaplus* 不大于 1。
- **TypeError** - 如果 *weight_decay* 既不是 float 也不是 int。
- **ValueError** - 如果 *weight_decay* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.Rprop(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'grad_centralization': True},
...                 {'params': no_conv_params, 'lr': 0.01},
...                 {'order_params': net.trainable_params()}]
>>> optim = nn.Rprop(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 default weight_
    ↳ decay of 0.0 and grad
>>> # centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight_
    ↳ decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
    ↳ 'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)

```

3.15.18 mindspore.nn.SGD

class mindspore.nn.SGD (params, learning_rate=0.1, momentum=0.0, dampening=0.0, weight_decay=0.0, nesterov=False, loss_scale=1.0)

随机梯度下降的实现。动量可选。

SGD 相关介绍参见 [SGD](#)。

Nesterov 动量公式参见论文 [On the importance of initialization and momentum in deep learning](#)。

$$v_{t+1} = u * v_t + gradient * (1 - dampening)$$

如果 nesterov 为 True：

$$p_{t+1} = p_t - lr * (gradient + u * v_{t+1})$$

如果 nesterov 为 False：

$$p_{t+1} = p_t - lr * v_{t+1}$$

需要注意的是，对于训练的第一步 $v_{t+1} = gradient$ 。其中，p、v 和 u 分别表示 parameters、accum 和 momentum。

说明：

- 在参数未分组时，优化器配置的 `weight_decay` 应用于名称不含“beta”或“gamma”的网络参数。
- 用户可以分组调整权重衰减策略。分组时，每组网络参数均可配置 `weight_decay`。若未配置，则该组网络参数使用优化器中配置的 `weight_decay`。

参数:

- **params** (Union[list[Parameter], list[dict]]) - 当 *params* 为会更新的 *Parameter* 列表时, *params* 中的元素必须为类 *Parameter*。当 *params* 为 *dict* 列表时, "params"、"lr"、"weight_decay"、"grad_centralization" 和 "order_params" 为可以解析的键。
 - **params** - 必填。当前组别的权重, 该值必须是 *Parameter* 列表。
 - **lr** - 可选。如果键中存在 "lr", 则使用对应的值作为学习率。如果没有, 则使用优化器中的参数 *learning_rate* 作为学习率。支持固定和动态学习率。
 - **weight_decay** - 可选。如果键中存在 "weight_decay", 则使用对应的值作为权重衰减值。如果没有, 则使用优化器中配置的 *weight_decay* 作为权重衰减值。当前 *weight_decay* 仅支持 *float* 类型, 不支持动态变化。
 - **grad_centralization** - 可选。如果键中存在 "grad_centralization", 则使用对应的值, 该值必须为布尔类型。如果没有, 则认为 *grad_centralization* 为 *False*。该参数仅适用于卷积层。
 - **order_params** - 可选。值的顺序是参数更新的顺序。当使用参数分组功能时, 通常使用该配置项保持 *parameters* 的顺序以提升性能。如果键中存在 "order_params", 则会忽略该组配置中的其他键。"order_params" 中的参数必须在某一组 *params* 参数中。
- **learning_rate** (Union[float, int, Tensor, Iterable, LearningRateSchedule], 可选) - 默认值: 0.1。
 - **float** - 固定的学习率。必须大于等于零。
 - **int** - 固定的学习率。必须大于等于零。整数类型会被转换为浮点数。
 - **Tensor** - 可以是标量或一维向量。标量是固定的学习率。一维向量是动态的学习率, 第 *i* 步将取向量中第 *i* 个值作为学习率。
 - **Iterable** - 动态的学习率。第 *i* 步将取迭代器第 *i* 个值作为学习率。
 - **LearningRateSchedule** - 动态的学习率。在训练过程中, 优化器将使用步数 (step) 作为输入, 调用 *LearningRateSchedule* 实例来计算当前学习率。
- **momentum** (float, 可选) - 浮点动量, 必须大于等于 0.0。默认值: 0.0。
- **dampening** (float, 可选) - 浮点动量阻尼值, 必须大于等于 0.0。默认值: 0.0。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty), 必须大于等于 0。默认值: 0.0。
- **nesterov** (bool, 可选) - 启用 Nesterov 动量。如果使用 Nesterov, 动量必须为正, 阻尼必须等于 0.0。默认值: *False*。
- **loss_scale** (float, 可选) - 梯度缩放系数, 必须大于 0.0。如果 *loss_scale* 是整数, 它将被转换为浮点数。通常使用默认值, 仅当训练时使用了 *FixedLossScaleManager*, 且 *FixedLossScaleManager* 的 *drop_overflow_update* 属性配置为 *False* 时, 此值需要与 *FixedLossScaleManager* 中的 *loss_scale* 相同。有关更多详细信息, 请参阅 *mindspore.amp.FixedLossScaleManager*。默认值: 1.0。

输入:

- **gradients** (tuple[Tensor]) - *params* 的梯度, *shape* 与 *params* 相同。

输出:

Tensor[bool], 值为 *True*。

异常:

- **ValueError** - 动量、阻尼或重量衰减值小于 0.0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> #1) All parameters use the same learning rate and weight decay
>>> optim = nn.SGD(params=net.trainable_params())
>>>
>>> #2) Use parameter groups and set different values
>>> conv_params = list(filter(lambda x: 'conv' in x.name, net.trainable_params()))
>>> no_conv_params = list(filter(lambda x: 'conv' not in x.name, net.trainable_params()))
>>> group_params = [{'params': conv_params, 'grad_centralization': True},
...                 {'params': no_conv_params, 'lr': 0.01},
...                 {'order_params': net.trainable_params()}]
>>> optim = nn.SGD(group_params, learning_rate=0.1, weight_decay=0.0)
>>> # The conv_params's parameters will use default learning rate of 0.1 and default weight_
↳decay of 0.0
>>> # and grad centralization of True.
>>> # The no_conv_params's parameters will use learning rate of 0.01 and default weight_
↳decay of 0.0 and grad
>>> # centralization of False.
>>> # The final parameters order in which the optimizer will be followed is the value of
↳'order_params'.
>>>
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim)
```

3.16 动态学习率

3.16.1 LearningRateSchedule 类

本模块中的动态学习率都是 LearningRateSchedule 的子类，将 LearningRateSchedule 的实例传递给优化器。在训练过程中，优化器以当前 step 为输入调用该实例，得到当前的学习率。

```
import mindspore.nn as nn

min_lr = 0.01
max_lr = 0.1
decay_steps = 4
cosine_decay_lr = nn.CosineDecayLR(min_lr, max_lr, decay_steps)

net = Net()
optim = nn.Momentum(net.trainable_params(), learning_rate=cosine_decay_lr, momentum=0.9)
```

接口名	概述	支持平台
<code>mindspore.nn.CosineDecayLR</code>	基于余弦衰减函数计算学习率。	Ascend GPU

下页继续

表 16 – 续上页

<code>mindspore.nn.ExponentialDecayLR</code>	基于指数衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.InverseDecayLR</code>	基于逆时衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.NaturalExpDecayLR</code>	基于自然指数衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.PolynomialDecayLR</code>	基于多项式衰减函数计算学习率。	Ascend GPU
<code>mindspore.nn.WarmUpLR</code>	预热学习率。	Ascend GPU CPU

`mindspore.nn.CosineDecayLR`

class `mindspore.nn.CosineDecayLR` (*min_lr*, *max_lr*, *decay_steps*)

基于余弦衰减函数计算学习率。

对于当前 *step*，计算学习率的公式为：

$$\text{decayed\_learning\_rate} = \text{min\_lr} + 0.5 * (\text{max\_lr} - \text{min\_lr}) * (1 + \cos(\frac{\text{current\_step}}{\text{decay\_steps}} \pi))$$

参数：

- **min_lr** (float) - 学习率的最小值。
- **max_lr** (float) - 学习率的最大值。
- **decay_steps** (int) - 进行衰减的 *step* 数。

输入：

- **global_step** (Tensor) - 当前 *step* 数，即 *current_step*。

输出：

标量 Tensor。当前 *step* 的学习率值，*shape* 为 *()*。

异常：

- **TypeError** - *min_lr* 或 *max_lr* 不是 float。
- **TypeError** - *decay_steps* 不是整数。
- **ValueError** - *min_lr* 小于 0 或 *decay_steps* 小于 1。
- **ValueError** - *max_lr* 小于或等于 0。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>>
>>> min_lr = 0.01
>>> max_lr = 0.1
>>> decay_steps = 4
>>> global_steps = Tensor(2, mindspore.int32)
>>> cosine_decay_lr = nn.CosineDecayLR(min_lr, max_lr, decay_steps)
>>> lr = cosine_decay_lr(global_steps)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

mindspore.nn.ExponentialDecayLR

class mindspore.nn.ExponentialDecayLR (*learning_rate, decay_rate, decay_steps, is_stair=False*)

基于指数衰减函数计算学习率。

对于当前 *step*，计算学习率的公式为：

$$decayed\_learning\_rate = learning\_rate * decay\_rate^p$$

其中，

$$p = \frac{current\_step}{decay\_steps}$$

如果 *is_stair* 为 *True*，则公式为：

$$p = floor(\frac{current\_step}{decay\_steps})$$

参数：

- **learning_rate** (float) - 学习率的初始值。
- **decay_rate** (float) - 衰减率。
- **decay_steps** (int) - 进行衰减的 *step* 数。
- **is_stair** (bool, 可选) - 如果为 *True*，则学习率每 *decay_steps* 步衰减一次。默认值：*False*。

输入：

- **global_step** (Tensor) - 当前 *step* 数，即上述公式中的 *current_step*。

输出：

标量 Tensor。当前 *step* 的学习率值，*shape* 为 *()*。

异常：

- **TypeError** - *learning_rate* 或 *decay_rate* 不是 float。
- **TypeError** - *decay_steps* 不是 int 或 *is_stair* 不是 bool。
- **ValueError** - *decay_steps* 小于 1。
- **ValueError** - *learning_rate* 或 *decay_rate* 小于或等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>>
>>> learning_rate = 0.1
>>> decay_rate = 0.9
>>> decay_steps = 4
>>> global_step = Tensor(2, mindspore.int32)
>>> exponential_decay_lr = nn.ExponentialDecayLR(learning_rate, decay_rate, decay_steps)
>>> lr = exponential_decay_lr(global_step)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```


mindspore.nn.InverseDecayLR

class mindspore.nn.InverseDecayLR (*learning_rate, decay_rate, decay_steps, is_stair=False*)

基于逆时衰减函数计算学习率。

对于当前 *step*，计算学习率的公式为：

$$decayed\_learning\_rate = learning\_rate / (1 + decay\_rate * p)$$

其中，

$$p = \frac{current\_step}{decay\_steps}$$

如果 *is_stair* 为 *True*，则公式为：

$$p = floor(\frac{current\_step}{decay\_steps})$$

参数：

- **learning_rate** (float) - 学习率的初始值。
- **decay_rate** (float) - 衰减率。
- **decay_steps** (int) - 进行衰减的 *step* 数。
- **is_stair** (bool, 可选) - 如果为 *True*，则学习率每 *decay_steps* 次衰减一次；如果为 *False*，则学习率每个 *step* 均衰减。默认值：*False*。

输入：

- **global_step** (Tensor) - 当前 *step* 数，即 *current_step*。

输出：

标量 Tensor。当前 *step* 的学习率值，*shape* 为 *()*。

异常：

- **TypeError** - *learning_rate* 或 *decay_rate* 不是 float。
- **TypeError** - *decay_steps* 不是 int，或 *is_stair* 不是 bool。
- **ValueError** - *decay_steps* 小于 1。
- **ValueError** - *learning_rate* 或 *decay_rate* 小于或等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>>
>>> learning_rate = 0.1
>>> decay_rate = 0.9
>>> decay_steps = 4
>>> global_step = Tensor(2, mindspore.int32)
>>> inverse_decay_lr = nn.InverseDecayLR(learning_rate, decay_rate, decay_steps, True)
>>> lr = inverse_decay_lr(global_step)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

mindspore.nn.NaturalExpDecayLR

class mindspore.nn.NaturalExpDecayLR (*learning_rate, decay_rate, decay_steps, is_stair=False*)

基于自然指数衰减函数计算学习率。

对于当前 step，计算学习率的公式为：

$$decayed\_learning\_rate = learning\_rate * e^{-decay\_rate * p}$$

其中，

$$p = \frac{current\_step}{decay\_steps}$$

如果 *is_stair* 为 True，则公式为：

$$p = floor(\frac{current\_step}{decay\_steps})$$

参数：

- **learning_rate** (float) - 学习率的初始值。
- **decay_rate** (float) - 衰减率。
- **decay_steps** (int) - 进行衰减的 step 数。
- **is_stair** (bool) - 如果为 True，则学习率每 *decay_steps* 次衰减一次。默认值：False。

输入：

- **global_step** (Tensor) - 当前 step 数，即 *current_step*，shape 为 ()。

输出：

标量 Tensor。当前 step 的学习率值。

异常：

- **TypeError** - *learning_rate* 或 *decay_rate* 不是 float。
- **TypeError** - *decay_steps* 不是 int 或 *is_stair* 不是 bool。
- **ValueError** - *decay_steps* 小于 1。
- **ValueError** - *learning_rate* 或 *decay_rate* 小于或等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>>
>>> learning_rate = 0.1
>>> decay_rate = 0.9
>>> decay_steps = 4
>>> global_step = Tensor(2, mindspore.int32)
>>> natural_exp_decay_lr = nn.NaturalExpDecayLR(learning_rate, decay_rate, decay_steps, True)
>>> lr = natural_exp_decay_lr(global_step)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

`mindspore.nn.PolynomialDecayLR`

class `mindspore.nn.PolynomialDecayLR` (*learning_rate, end_learning_rate, decay_steps, power, update_decay_steps=False*)

基于多项式衰减函数计算学习率。

对于当前 `step`，计算学习率的公式为：

$$\begin{aligned} \text{decayed\_learning\_rate} = & (\text{learning\_rate} - \text{end\_learning\_rate}) * \\ & (1 - \text{tmp\_step} / \text{tmp\_decay\_steps})^{\text{power}} \\ & + \text{end\_learning\_rate} \end{aligned}$$

其中，

$$\text{tmp\_step} = \min(\text{current\_step}, \text{decay\_steps})$$

如果 `update_decay_steps` 为 `true`，则每 `decay_steps` 更新 `tmp_decay_steps` 的值。公式为：

$$\text{tmp\_decay\_steps} = \text{decay\_steps} * \text{ceil}(\text{current\_step} / \text{decay\_steps})$$

参数：

- **learning_rate** (float) - 学习率的初始值。
- **end_learning_rate** (float) - 学习率的最终值。
- **decay_steps** (int) - 进行衰减的 `step` 数。
- **power** (float) - 多项式的幂，必须大于 0。
- **update_decay_steps** (bool) - 如果为 `True`，则学习率每 `decay_steps` 次衰减一次。默认值：`False`。

输入：

- **global_step** (Tensor) - 当前 `step` 数，即 `current_step`。shape 为 `()`。

输出：

标量 Tensor。当前 `step` 的学习率值，shape 为 `()`。

异常：

- **TypeError** - `learning_rate`, `end_learning_rate` 或 `power` 不是 float。
- **TypeError** - `decay_steps` 不是 int 或 `update_decay_steps` 不是 bool。
- **ValueError** - `end_learning_rate` 小于 0 或 `decay_steps` 小于 1。
- **ValueError** - `learning_rate` 或 `power` 小于或等于 0。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>>
>>> learning_rate = 0.1
>>> end_learning_rate = 0.01
>>> decay_steps = 4
>>> power = 0.5
```

(下页继续)

(续上页)

```
>>> global_step = Tensor(2, mindspore.int32)
>>> polynomial_decay_lr = nn.PolynomialDecayLR(learning_rate, end_learning_rate, decay_steps,
↪ power)
>>> lr = polynomial_decay_lr(global_step)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

mindspore.nn.WarmUpLR

class mindspore.nn.WarmUpLR(*learning_rate*, *warmup_steps*)

预热学习率。

对于当前 *step*，计算学习率的公式为：

$$warmup\_learning\_rate = learning\_rate * tmp\_step / warmup\_steps$$

其中，

$$tmp\_step = \min(current\_step, warmup\_steps)$$

参数：

- **learning_rate** (float) - 学习率的初始值，*learning_rate* 的值必须大于 0。
- **warmup_steps** (int) - 学习率 warmup 的 *step* 数，*warmup_steps* 的值必须大于或等于 1。

输入：

- **global_step** (Tensor) - 当前 *step* 数，即 *current_step*。shape 为 ()。

输出：

标量 Tensor。当前 *step* 的学习率值，shape 为 ()。

异常：

- **TypeError** - *learning_rate* 不是 float。
- **TypeError** - *warmup_steps* 不是 int。
- **ValueError** - *warmup_steps* 小于 1。
- **ValueError** - *learning_rate* 小于或等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>>
>>> learning_rate = 0.1
>>> warmup_steps = 2
>>> global_step = Tensor(2, mindspore.int32)
>>> warmup_lr = nn.WarmUpLR(learning_rate, warmup_steps)
>>> lr = warmup_lr(global_step)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

3.16.2 Dynamic LR 函数

本模块中的动态学习率都是 `function`，调用 `function` 并将结果传递给优化器。在训练过程中，优化器将 `result[current step]` 作为当前学习率。

```
import mindspore.nn as nn

min_lr = 0.01
max_lr = 0.1
total_step = 6
step_per_epoch = 1
decay_epoch = 4

lr = nn.cosine_decay_lr(min_lr, max_lr, total_step, step_per_epoch, decay_epoch)

net = Net()
optim = nn.Momentum(net.trainable_params(), learning_rate=lr, momentum=0.9)
```

接口名	概述	支持平台
<code>mindspore.nn.cosine_decay_lr</code>	基于余弦衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.exponential_decay_lr</code>	基于指数衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.inverse_decay_lr</code>	基于逆时衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.natural_exp_decay_lr</code>	基于自然指数衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.piecewise_constant_lr</code>	获取分段常量学习率。	Ascend GPU CPU
<code>mindspore.nn.polynomial_decay_lr</code>	基于多项式衰减函数计算学习率。	Ascend GPU CPU
<code>mindspore.nn.warmup_lr</code>	预热学习率。	Ascend GPU CPU

`mindspore.nn.cosine_decay_lr`

`mindspore.nn.cosine_decay_lr(min_lr, max_lr, total_step, step_per_epoch, decay_epoch)`

基于余弦衰减函数计算学习率。每个 `step` 的学习率将会被存放在一个列表中。

对于第 `i` 步，计算 `decayed_learning_rate[i]` 的公式为：

$$decayed\_learning\_rate[i] = min\_lr + 0.5 * (max\_lr - min\_lr) * (1 + \cos(\frac{current\_epoch}{decay\_epoch} \pi))$$

其中 $current\_epoch = floor(\frac{i}{step\_per\_epoch})$ 。

参数：

- **min_lr** (float) - 学习率的最小值。
- **max_lr** (float) - 学习率的最大值。
- **total_step** (int) - `step` 总数。
- **step_per_epoch** (int) - 每个 `epoch` 的 `step` 数。
- **decay_epoch** (int) - 进行衰减的 `epoch` 数。

返回：

`list[float]`。列表大小为 `total_step`。

异常：

- **TypeError** - `min_lr` 或 `max_lr` 不是 `float`。

- **TypeError** - *total_step* 或 *step_per_epoch* 或 *decay_epoch* 不是 int。
- **ValueError** - *max_lr* 不大于 0 或 *min_lr* 小于 0。
- **ValueError** - *total_step* 或 *step_per_epoch* 或 *decay_epoch* 小于 0。
- **ValueError** - *max_lr* 大于或等于 *min_lr*。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.nn as nn
>>>
>>> min_lr = 0.01
>>> max_lr = 0.1
>>> total_step = 6
>>> step_per_epoch = 2
>>> decay_epoch = 2
>>> lr = nn.cosine_decay_lr(min_lr, max_lr, total_step, step_per_epoch, decay_epoch)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

mindspore.nn.exponential_decay_lr

mindspore.nn.exponential_decay_lr (*learning_rate*, *decay_rate*, *total_step*, *step_per_epoch*, *decay_epoch*, *is_stair=False*)

基于指数衰减函数计算学习率。每个 *step* 的学习率将会被存放在一个列表中。

对于第 *i* 步，计算 *decayed_learning_rate[i]* 的公式为：

$$decayed\_learning\_rate[i] = learning\_rate * decay\_rate^{\frac{current\_epoch}{decay\_epoch}}$$

其中 $current\_epoch = floor(\frac{i}{step\_per\_epoch})$ 。

参数：

- **learning_rate** (float) - 学习率的初始值。
- **decay_rate** (float) - 衰减率。
- **total_step** (int) - *step* 总数。
- **step_per_epoch** (int) - 每个 *epoch* 的 *step* 数。
- **decay_epoch** (int) - 进行衰减的 *epoch* 数。
- **is_stair** (bool) - 如果为 *True*，则学习率每 *decay_epoch* 次衰减一次。默认值：False。

返回：

list[float]。列表的大小为 *total_step*。

异常：

- **TypeError** - *total_step* 或 *step_per_epoch* 或 *decay_epoch* 不是 int。
- **TypeError** - *is_stair* 不是 bool。
- **TypeError** - *learning_rate* 或 *decay_rate* 不是 float。
- **ValueError** - *learning_rate* 或 *decay_rate* 小于等于 0。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.nn as nn
>>>
>>> learning_rate = 0.1
>>> decay_rate = 0.9
>>> total_step = 6
>>> step_per_epoch = 2
>>> decay_epoch = 1
>>> lr = nn.exponential_decay_lr(learning_rate, decay_rate, total_step, step_per_epoch,
    ↳decay_epoch)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

`mindspore.nn.inverse_decay_lr`

`mindspore.nn.inverse_decay_lr(learning_rate, decay_rate, total_step, step_per_epoch, decay_epoch, is_stair=False)`

基于逆时衰减函数计算学习率。每个 step 的学习率将会被存放在一个列表中。

对于第 i 步, 计算 `decayed_learning_rate[i]` 的公式为:

$$\text{decayed\_learning\_rate}[i] = \text{learning\_rate} / (1 + \text{decay\_rate} * \text{current\_epoch} / \text{decay\_epoch})$$

其中, $\text{current\_epoch} = \text{floor}(\frac{i}{\text{step\_per\_epoch}})$ 。

参数:

- **learning_rate** (float) - 学习率的初始值。
- **decay_rate** (float) - 衰减率。
- **total_step** (int) - step 总数。
- **step_per_epoch** (int) - 每个 epoch 的 step 数。
- **decay_epoch** (int) - 进行衰减的 epoch 数。
- **is_stair** (bool) - 如果为 True, 则学习率每 `decay_epoch` 次衰减一次。如果为 False, 则学习率每个 epoch 均衰减。默认值: False。

返回:

`list[float]`。列表大小为 `total_step`。

异常:

- **TypeError** - `total_step` 或 `step_per_epoch` 或 `decay_epoch` 不是 int。
- **TypeError** - `is_stair` 不是 bool。
- **TypeError** - `learning_rate` 或 `decay_rate` 不是 float。
- **ValueError** - `learning_rate` 或 `decay_rate` 小于等于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.nn as nn
>>>
>>> learning_rate = 0.1
>>> decay_rate = 0.5
>>> total_step = 6
>>> step_per_epoch = 1
>>> decay_epoch = 1
>>> lr = nn.inverse_decay_lr(learning_rate, decay_rate, total_step, step_per_epoch, decay_
    epoch, True)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

`mindspore.nn.natural_exp_decay_lr`

`mindspore.nn.natural_exp_decay_lr` (*learning_rate, decay_rate, total_step, step_per_epoch, decay_epoch, is_stair=False*)

基于自然指数衰减函数计算学习率。每个 `step` 的学习率将会被存放在一个列表中。

对于第 i 步，计算 `decayed_learning_rate[i]` 的公式为：

$$\text{decayed\_learning\_rate}[i] = \text{learning\_rate} * e^{-\text{decay\_rate} * \text{current\_epoch}}$$

其中 $\text{current\_epoch} = \text{floor}(\frac{i}{\text{step\_per\_epoch}})$ 。

参数：

- **learning_rate** (float) - 学习率的初始值。
- **decay_rate** (float) - 衰减率。
- **total_step** (int) - step 总数。
- **step_per_epoch** (int) - 每个 epoch 的 step 数。
- **decay_epoch** (int) - 进行衰减的 epoch 数。
- **is_stair** (bool) - 如果为 `True`，则学习率每 `decay_epoch` 次衰减一次。默认值：`False`。

返回：

`list[float]`。`total_step` 表示列表的大小。

异常：

- **TypeError** - `total_step` 或 `step_per_epoch` 或 `decay_epoch` 不是 `int`。
- **TypeError** - `is_stair` 不是 `bool`。
- **TypeError** - `learning_rate` 或 `decay_rate` 不是 `float`。
- **ValueError** - `learning_rate` 或 `decay_rate` 小于等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.nn as nn
>>>
>>> learning_rate = 0.1
>>> decay_rate = 0.9
>>> total_step = 6
```

(下页继续)

(续上页)

```
>>> step_per_epoch = 2
>>> decay_epoch = 2
>>> lr = nn.natural_exp_decay_lr(learning_rate, decay_rate, total_step, step_per_epoch,
    ↳decay_epoch, True)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

mindspore.nn.piecewise_constant_lr

`mindspore.nn.piecewise_constant_lr(milestone, learning_rates)`

获取分段常量学习率。每个 step 的学习率将会被存放在一个列表中。

通过给定的 *milestone* 和 *learning_rates* 计算学习率。设 *milestone* 的值为 $(M_1, M_2, \dots, M_t, \dots, M_N)$ ，*learning_rates* 的值为 $(x_1, x_2, \dots, x_t, \dots, x_N)$ 。N 是 *milestone* 的长度。设 *y* 为输出学习率，那么对于第 *i* 步，计算 $y[i]$ 的公式为：

$$y[i] = x_t, \text{ for } i \in [M_{t-1}, M_t)$$

参数：

- **milestone** (Union[list[int], tuple[int]]) - milestone 列表。当达到指定的 step 时，使用对应的 *learning_rates*。此列表是一个单调递增的列表。列表中的元素必须大于 0。
- **learning_rates** (Union[list[float], tuple[float]]) - 学习率列表。

返回：

list[float]。列表的大小为 M_N 。

异常：

- **TypeError** - *milestone* 或 *learning_rates* 既不是 tuple 也不是 list。
- **ValueError** - *milestone* 和 *learning_rates* 的长度不相等。
- **ValueError** - *milestone* 中的不是单调递增的。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.nn as nn
>>>
>>> milestone = [2, 5, 10]
>>> learning_rates = [0.1, 0.05, 0.01]
>>> lr = nn.piecewise_constant_lr(milestone, learning_rates)
>>> # learning_rates = 0.1 if step <= 2
>>> # learning_rates = 0.05 if 2 < step <= 5
>>> # learning_rates = 0.01 if 5 < step <= 10
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

mindspore.nn.polynomial_decay_lr

`mindspore.nn.polynomial_decay_lr` (*learning_rate*, *end_learning_rate*, *total_step*, *step_per_epoch*, *decay_epoch*, *power*, *update_decay_epoch=False*)

基于多项式衰减函数计算学习率。每个 *step* 的学习率将会被存放在一个列表中。

对于第 *i* 步, 计算 `decayed_learning_rate[i]` 的公式为:

$$\text{decayed\_learning\_rate}[i] = (\text{learning\_rate} - \text{end\_learning\_rate}) * (1 - \text{tmp\_epoch}/\text{tmp\_decay\_epoch})^{\text{power}} + \text{end\_learning\_rate}$$

其中,

$$\text{tmp\_epoch} = \min(\text{current\_epoch}, \text{decay\_epoch})$$

$$\text{current\_epoch} = \text{floor}\left(\frac{i}{\text{step\_per\_epoch}}\right)$$

$$\text{tmp\_decay\_epoch} = \text{decay\_epoch}$$

如果 *update_decay_epoch* 为 `True`, 则每个 *epoch* 更新 *tmp_decay_epoch* 的值。公式为:

$$\text{tmp\_decay\_epoch} = \text{decay\_epoch} * \text{ceil}(\text{current\_epoch}/\text{decay\_epoch})$$

参数:

- **learning_rate** (float) - 学习率的初始值。
- **end_learning_rate** (float) - 学习率的最终值。
- **total_step** (int) - *step* 总数。
- **step_per_epoch** (int) - 每个 *epoch* 的 *step* 数。
- **decay_epoch** (int) - 进行衰减的 *epoch* 数。
- **power** (float) - 多项式的幂, 必须大于 0。
- **update_decay_epoch** (bool) - 如果为 `True`, 则更新 *decay_epoch*。默认值: `False`。

返回:

`list[float]`。列表的大小为 *total_step*。

异常:

- **TypeError** - *learning_rate* 或 *end_learning_rate* 或 *power* 不是 float。
- **TypeError** - *total_step* 或 *step_per_epoch* 或 *decay_epoch* 不是 int。
- **TypeError** - *update_decay_epoch* 不是 bool。
- **ValueError** - *learning_rate* 或 *power* 小于等于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.nn as nn
>>>
>>> lr = 0.1
>>> end_learning_rate = 0.01
>>> total_step = 6
```

(下页继续)

(续上页)

```
>>> step_per_epoch = 2
>>> decay_epoch = 2
>>> power = 0.5
>>> lr = nn.polynomial_decay_lr(lr, end_learning_rate, total_step, step_per_epoch, decay_
    epoch, power)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

mindspore.nn.warmup_lr

`mindspore.nn.warmup_lr(learning_rate, total_step, step_per_epoch, warmup_epoch)`

预热学习率。每个 step 的学习率将会被存放在一个列表中。

对于第 i 步，计算 `warmup_learning_rate[i]` 的公式为：

$$\text{warmup\_learning\_rate}[i] = \text{learning\_rate} * \text{tmp\_epoch} / \text{warmup\_epoch}$$

其中 $\text{tmp\_epoch} = \min(\text{current\_epoch}, \text{warmup\_epoch})$, $\text{current\_epoch} = \text{floor}(\frac{i}{\text{step\_per\_epoch}})$

参数：

- **learning_rate** (float) - 学习率的初始值。
- **total_step** (int) - step 总数。
- **step_per_epoch** (int) - 每个 epoch 的 step 数。
- **warmup_epoch** (int) - 预热学习率的 epoch 数。

返回：

`list[float]`。 `total_step` 表示列表的大小。

异常：

- **TypeError** - `learning_rate` 不是 float。
- **TypeError** - `total_step` 或 `step_per_epoch` 或 `decay_epoch` 不是 int。
- **ValueError** - `learning_rate` 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.nn as nn
>>>
>>> learning_rate = 0.1
>>> total_step = 6
>>> step_per_epoch = 2
>>> warmup_epoch = 2
>>> lr = nn.warmup_lr(learning_rate, total_step, step_per_epoch, warmup_epoch)
>>> net = nn.Dense(2, 3)
>>> optim = nn.SGD(net.trainable_params(), learning_rate=lr)
```

3.17 图像处理层

接口名	概述	支持平台
<code>mindspore.nn.PixelShuffle</code>	对 <i>input</i> 应用像素重组操作，它实现了步长为 $1/r$ 的子像素卷积。	Ascend GPU CPU
<code>mindspore.nn.PixelUnshuffle</code>	对 <i>input</i> 应用逆像素重组操作，这是像素重组的逆操作。	Ascend GPU CPU
<code>mindspore.nn.Upsample</code>	详情请参考 <code>mindspore.ops.interpolate()</code> 。	Ascend GPU CPU

3.17.1 mindspore.nn.PixelShuffle

class mindspore.nn.PixelShuffle(*upscale_factor*)
对 *input* 应用像素重组操作，它实现了步长为 $1/r$ 的子像素卷积。关于 PixelShuffle 算法详细介绍，请参考 [Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network](#)。
通常情况下，输入 shape $(*, C \times r^2, H, W)$ ，输出 shape $(*, C, H \times r, W \times r)$ 。 r 是缩小因子。 $*$ 是大于等于 0 的维度。

说明：Ascend 上输入 Tensor 的维度要小于 7。

- 参数：
- upscale_factor** (int) - 打乱输入 Tensor 的因子，是正整数。*upscale_factor* 是上面提到的 r 。
- 输入：
- input** (Tensor) - Tensor，shape 为 $(*, C \times r^2, H, W)$ 。输入 Tensor 的维度需要大于 2，并且倒数第三维 length 可以被 *upscale_factor* 的平方整除。
- 输出：
- output** (Tensor) - Tensor，shape 为 $(*, C, H \times r, W \times r)$ 。
- 异常：
- ValueError** - *upscale_factor* 不是正整数。
 - ValueError** - 输入 *input* 倒数第三维度的 length 不能被 *upscale_factor* 的平方整除。
 - ValueError** - *input* 维度小于 3。
 - TypeError** - *input* 不是 Tensor。
- 支持平台：
- Ascend GPU CPU
- 样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> input_x = np.arange(3 * 2 * 8 * 4 * 4).reshape((3, 2, 8, 4, 4))
>>> input_x = ms.Tensor(input_x, ms.dtype.int32)
>>> pixel_shuffle = ms.nn.PixelShuffle(2)
>>> output = pixel_shuffle(input_x)
```

(下页继续)

(续上页)

```
>>> print(output.shape)
(3, 2, 2, 8, 8)
```

3.17.2 mindspore.nn.PixelUnshuffle

class mindspore.nn.PixelUnshuffle(*downscale_factor*)

对 *input* 应用逆像素重组操作，这是像素重组的逆操作。关于 PixelUnshuffle 算法详细介绍，请参考 [Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network](#)。

通常情况下，输入 shape $(*, C, H \times r, W \times r)$ ，输出 shape $(*, C \times r^2, H, W)$ 。 r 是缩小因子。 $*$ 是大于等于 0 的维度。

参数：

- **downscale_factor** (int) - 恢复输入 Tensor 的因子，是正整数。*downscale_factor* 是上面提到的 r 。

输入：

- **input** (Tensor) - Tensor，shape 为 $(*, C, H \times r, W \times r)$ 。输入 Tensor 的维度需要大于 2，并且倒数第一和倒数第二维对应的值可以被 *downscale_factor* 整除。

输出：

- **output** (Tensor) - Tensor，shape 为 $(*, C \times r^2, H, W)$ 。

异常：

- **ValueError** - *downscale_factor* 不是正整数。
- **ValueError** - 输入 *input* 倒数第一和倒数第二维度对应的值不能被 *downscale_factor* 整除。
- **ValueError** - *input* 维度小于 3。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> pixel_unshuffle = ms.nn.PixelUnshuffle(2)
>>> input_x = np.arange(8 * 8).reshape((1, 1, 8, 8))
>>> input_x = ms.Tensor(input_x, ms.dtype.int32)
>>> output = pixel_unshuffle(input_x)
>>> print(output.shape)
(1, 4, 4, 4)
```

3.17.3 mindspore.nn.Upsample

class mindspore.nn.Upsample (size=None, scale_factor=None, mode='nearest', align_corners=None,
 recompute_scale_factor=None)

详情请参考mindspore.ops.interpolate()。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> x = ms.Tensor([[[[1.0, 2.0, 3.0, 4.0], [5.0, 6.0, 7.0, 8.0]]]])
>>> upsample = ms.nn.Upsample(size=(5, 5))
>>> out = upsample(x)
>>> print(x.asnumpy())
[[[1. 2. 3. 4.]
  [5. 6. 7. 8.]]]
>>> print(out.asnumpy())
[[[1. 1. 2. 3. 4.]
  [1. 1. 2. 3. 4.]
  [1. 1. 2. 3. 4.]
  [5. 5. 6. 7. 8.]
  [5. 5. 6. 7. 8.]]]
>>> print(out.shape)
(1, 1, 5, 5)
```

3.18 公共层

接口名	概述	支持平台
<i>mindspore.nn.ChannelShuffle</i>	将 shape 为 $(*, C, H, W)$ 的 Tensor 的通道划分成 g 组, 并按如下方式重新排列 $(*, \frac{C}{g}, g, H * W)$, 同时在最终输出中保持原始 Tensor 的 shape。	Ascend GPU CPU
<i>mindspore.nn.Flatten</i>	沿着从 <i>start_dim</i> 到 <i>end_dim</i> 的维度, 对输入 Tensor 进行展平。	Ascend GPU CPU
<i>mindspore.nn.Identity</i>	网络占位符, 返回与输入完全一致。	Ascend GPU CPU
<i>mindspore.nn.Unflatten</i>	根据 <i>axis</i> 和 <i>unflattened_size</i> 折叠指定维度为给定形状。	Ascend GPU CPU

3.18.1 mindspore.nn.ChannelShuffle

class mindspore.nn.ChannelShuffle (groups)

将 shape 为 $(*, C, H, W)$ 的 Tensor 的通道划分成 g 组, 并按如下方式重新排列 $(*, \frac{C}{g}, g, H * W)$, 同时在最终输出中保持原始 Tensor 的 shape。

参数：

- **groups** (int) - 划分通道的组数, 必须大于 0。在上述公式中表示为 g 。

输入：

- **x** (Tensor) - Tensor 的 shape $(*, C_{in}, H_{in}, W_{in})$ 。

输出:

Tensor, 数据类型和 shape 与 x 相同。

异常:

- **TypeError** - $groups$ 非正整数。
- **ValueError** - $groups$ 小于 1。
- **ValueError** - x 的维度小于 3。
- **ValueError** - x 的通道数不能被 $groups$ 整除。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> channel_shuffle = ms.nn.ChannelShuffle(2)
>>> x = ms.Tensor(np.arange(16).astype(np.int32).reshape(1, 4, 2, 2))
>>> print(x)
[[[ [ 0  1]
      [ 2  3]
      [ 4  5]
      [ 6  7]
      [ 8  9]
      [10 11]
      [12 13]
      [14 15]]]]
>>> output = channel_shuffle(x)
>>> print(output)
[[[ [ 0  1]
      [ 2  3]
      [ 8  9]
      [10 11]
      [ 4  5]
      [ 6  7]
      [12 13]
      [14 15]]]]]
```

3.18.2 mindspore.nn.Flatten

class mindspore.nn.Flatten ($start\_dim=1, end\_dim=-1$)

沿着从 $start\_dim$ 到 end_dim 的维度, 对输入 Tensor 进行展平。

参数:

- **start_dim** (int, 可选) - 要展平的第一个维度。默认值: 1。
- **end_dim** (int, 可选) - 要展平的最后一个维度。默认值: -1。

输入:

- **x** (Tensor) - 要展平的输入 Tensor。

输出:

Tensor。如果没有维度被展平，返回原始的输入 *input*，否则返回展平后的 Tensor。如果 *input* 是零维 Tensor，将会返回一个一维 Tensor。

异常：

- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *start_dim* 或 *end_dim* 不是 int 类型。
- **ValueError** - 规范化后，*start_dim* 大于 *end_dim*。
- **ValueError** - *start_dim* 或 *end_dim* 不在 $[-input.dim, input.dim-1]$ 范围内。例如参数使用默认值且输入为零维或一维 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> x = Tensor(np.array([[1.2, 1.2], [2.1, 2.1]], [[2.2, 2.2], [3.2, 3.2]]), mindspore.
↳float32)
>>> net = nn.Flatten()
>>> output = net(x)
>>> print(output)
[[1.2 1.2 2.1 2.1]
 [2.2 2.2 3.2 3.2]]
>>> print(f"before flatten the x shape is {x.shape}")
before flatten the x shape is (2, 2, 2)
>>> print(f"after flatten the output shape is {output.shape}")
after flatten the output shape is (2, 4)
```

3.18.3 mindspore.nn.Identity

class mindspore.nn.Identity(*args, **kwargs)

网络占位符，返回与输入完全一致。

参数：

- **args** (Any) - 任意参数。
- **kwargs** (Any) - 任意关键字参数。

输入：

- **input** (Any) - Identity 的输入。

输出：

与 *input* 完全相同。

支持平台：

Ascend GPU CPU

样例：


```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.array([1, 2, 3, 4]), mindspore.int64)
>>> net = nn.Identity()
>>> output = net(input)
>>> print(output)
[1 2 3 4]
```

3.18.4 mindspore.nn.Unflatten

class mindspore.nn.Unflatten(*axis*, *unflattened_size*)

根据 *axis* 和 *unflattened_size* 折叠指定维度为给定形状。

参数：

- **axis** (int) - 指定输入 Tensor 被折叠维度。
- **unflattened_size** (Union(tuple[int], list[int])) - 指定维度折叠后的新 shape，可以为 tuple[int] 或者 list[int]。
unflattened_size 中各元素的乘积必须等于 *input_shape[axis]*。

输入：

- **input** (Tensor) - 进行折叠操作的 Tensor。

输出：

折叠操作后的 Tensor。

异常：

- **TypeError** - *axis* 不是 int。
- **TypeError** - *unflattened_size* 既不是 tuple[int] 也不是 list[int]。
- **TypeError** - *unflattened_size* 中各元素的乘积不等于 *input_shape[axis]*。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, nn
>>> import numpy as np
>>> input = Tensor(np.arange(0, 100).reshape(2, 10, 5), mindspore.float32)
>>> net = nn.Unflatten(1, (2, 5))
>>> output = net(input)
>>> print(f"before unflatten the input shape is {input.shape}")
before unflatten the input shape is (2, 10, 5)
>>> print(f"after unflatten the output shape is {output.shape}")
after unflatten the output shape is (2, 2, 5, 5)
```

3.19 工具

`mindspore.nn.utils.no_init_parameters`

使用该接口可跳过 `parameter` 初始化。

3.19.1 mindspore.nn.utils.no_init_parameters

`mindspore.nn.utils.no_init_parameters()`

使用该接口可跳过 `parameter` 初始化。

加载 `checkpoint` 的场景下网络实例化中 `parameter` 会实例化并占用物理内存, 加载 `checkpoint` 会替换 `parameter` 值, 使用该接口在网络实例化时用装饰器给当前 `Cell` 里所有 `parameter` 添加一个属性: `init_param`, 并设为 `init_param=False`, 检测到 `init_param=False` 时跳过 `parameter` 初始化, 加载 `checkpoint` 时从 `checkpoint` 给 `parameter` 赋值, 可优化性能和减少物理内存。

说明: 只能跳过使用 `initializer` 创建的 `parameter` 的初始化, 由 `Tensor` 或 `numpy` 创建的 `parameter` 无法跳过。

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn, ops, load_checkpoint
>>> from mindspore.common.initializer import initializer
>>> from mindspore.nn.utils import no_init_parameters
>>> # 1. Add a decorator to the network that requires delayed initialization
>>> class Net(nn.Cell):
...     def __init__(self, in_channels, out_channels):
...         super().__init__()
...         self.weight = ms.Parameter(initializer("normal", [in_channels, out_channels], ms.
float32))
...         self.bias = ms.Parameter(initializer("normal", [out_channels], ms.float32))
...         self.matmul = ops.MatMul()
...         self.add = ops.Add()
...     def construct(self, x):
...         x = self.matmul(x, self.weight)
...         x = self.add(x, self.bias)
...         return x
>>> with no_init_parameters():
...     # After instantiation, all parameters in the net are not initialized
...     net = Net(28*28, 64)
>>> # 2. Load checkpoint parameters to the net
>>> load_checkpoint('./checkpoint/test_net.ckpt', net=net)
>>> # 3. After loading the checkpoint, manually call init_parameters_data() to initialize
>>> #     the uninitialized parameters in the net if need. If the network is executed,
>>> #     the framework will automatically call this interface.
>>> net.init_parameters_data()
```

mindspore.mint 提供了大量的 functional、nn、优化器接口，API 用法及功能等与业界主流用法一致，方便用户参考使用。mint 接口当前是实验性接口，在图编译模式为 O0 和 PyNative 模式下性能比 ops 更优。当前暂不支持 O2（图下沉模式）及 CPU、GPU 后端，后续会逐步完善。

模块导入方法如下：

```
from mindspore import mint
```

MindSpore 中 *mindspore.mint* 接口与上一版本相比，新增、删除和支持平台的变化信息请参考 [mindspore.mint API 接口变更](#)。

4.1 Tensor

4.1.1 创建运算

接口名	概述	支持平台
<i>mindspore.mint.arange</i>	创建一个从 <i>start</i> 开始到 <i>end</i> 结束（不含）、步长为 <i>step</i> 的序列（一维张量）。	Ascend
<i>mindspore.mint.bernoulli</i>	从伯努利分布中进行采样，并根据输入 <i>input</i> 中第 <i>i</i> 个元素给出的概率值，将输出 <i>output</i> 中的第 <i>i</i> 元素随机设置为 0 或 1。	Ascend
<i>mindspore.mint.bincount</i>	统计 <i>input</i> 中每个值的出现次数。	Ascend
<i>mindspore.mint.clone</i>	返回一个输入 Tensor 的副本。	Ascend
<i>mindspore.mint.eye</i>	返回一个主对角线上元素为 1，其余元素为 0 的 tensor。	Ascend
<i>mindspore.mint.einsum</i>	基于爱因斯坦求和约定（Einsum）符号，沿着指定维度对输入 Tensor 元素的乘积求和。	Ascend
<i>mindspore.mint.empty</i>	创建一个数据没有初始化的 Tensor。	Ascend

下页继续

表 1 - 续上页

<code>mindspore.mint.empty_like</code>	创建一个未初始化的 Tensor, shape 和 <i>input</i> 相同, dtype 由 <i>dtype</i> 决定, Tensor 使用的内存由 <i>device</i> 决定。	Ascend
<code>mindspore.mint.full</code>	创建一个指定 shape 的 Tensor, 并用指定值填充。	Ascend
<code>mindspore.mint.full_like</code>	返回一个 shape 与 <i>input</i> 相同并且使用 <i>fill_value</i> 填充的 Tensor。	Ascend
<code>mindspore.mint.linspace</code>	创建一个 steps 个元素的, 在 [start, end] 区间内均匀分布的一维 tensor。	Ascend
<code>mindspore.mint.ones</code>	创建一个值全为 1 的 Tensor。	Ascend
<code>mindspore.mint.ones_like</code>	创建一个数值全为 1 的 Tensor, shape 和 <i>input</i> 相同, dtype 由 <i>dtype</i> 决定。	Ascend
<code>mindspore.mint.randint</code>	返回一个 Tensor, shape 和 dtype 由输入决定, 其元素为 [<i>low</i> , <i>high</i>) 区间的随机整数。	Ascend
<code>mindspore.mint.randint_like</code>	返回一个 Tensor, 其元素为 [<i>low</i> , <i>high</i>) 区间的随机整数, 根据 <i>input</i> 决定 shape 和 dtype。	Ascend
<code>mindspore.mint.randn</code>	返回一个 Tensor, shape 和 dtype 由输入决定, 其元素为服从标准正态分布的数字。	Ascend
<code>mindspore.mint.randn_like</code>	返回 shape 与输入相同, 类型为 <i>dtype</i> 的 Tensor。	Ascend
<code>mindspore.mint.randperm</code>	生成从 0 到 n-1 的整数随机排列。	Ascend
<code>mindspore.mint.zeros</code>	创建一个值全为 0 的 Tensor。	Ascend
<code>mindspore.mint.zeros_like</code>	创建一个数值全为 0 的 Tensor, shape 和 <i>input</i> 相同, dtype 由 <i>dtype</i> 决定。	Ascend

mindspore.mint.arange

`mindspore.mint.arange` (*start=0*, *end=None*, *step=1*, *, *dtype=None*)

创建一个从 *start* 开始到 *end* 结束 (不含)、步长为 *step* 的序列 (一维张量)。

参数:

- **start** (Union[float, int], 可选) - 序列起始值, 默认值: 0。
- **end** (Union[float, int], 可选) - 序列终止值 (不含), 默认值: None, 如果是 None, 则将 *start* 作为终止值, 0 为起始值。
- **step** (Union[float, int], 可选) - 序列步长, 默认值: 1。

关键字参数:

- **dtype** (mindspore.dtype, 可选) - 输出 Tensor 的 dtype。默认值: None。

当该参数未设置或为 None 时:

如果 *start*、*end* 和 *step* 都是 int, 则输出 Tensor 的 dtype 为 int64。

如果 *start*、*end* 和 *step* 中包含 float, 则输出 Tensor 的 dtype 为 float32。

返回:

一维张量, 如果设置了 *dtype* 参数, 则会被 cast 成该类型的 Tensor, 有可能因此损失精度。

异常:

- **TypeError** - *start*, *end* 和 *step* 不是 int 或 float 类型。
- **ValueError** - *step* = 0。
- **ValueError** - *step* > 0 且 *start* >= *end*。

- **ValueError** - $step < 0$ 且 $start \leq end$ 。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, mint
>>> output = mint.arange(1, 6)
>>> print(output)
[1 2 3 4 5]
>>> print(output.dtype)
Int64
>>> output = mint.arange(0, 3, 1.2)
>>> print(output)
[0.  1.2 2.4]
>>> print(output.dtype)
Float32
>>> output = mint.arange(7, 1, -2)
>>> print(output)
[7 5 3]
>>> print(output.dtype)
Int64
>>> output = mint.arange(12, 2, -1, dtype=ms.bfloat16)
>>> print(output)
[12. 11. 10.  9.  8.  7.  6.  5.  4.  3.]
>>> print(output.dtype)
BFloat16
```

mindspore.mint.bernoulli

`mindspore.mint.bernoulli(input, *, generator=None)`

从伯努利分布中进行采样，并根据输入 *input* 中第 *i* 个元素给出的概率值，将输出 *output* 中的第 *i* 元素随机设置为 0 或 1。

$$output_i \sim \text{Bernoulli}(p = input_i)$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 伯努利分布的输入张量，其中元素 *input_i* 代表对应输出元素 *output_i* 被设为 1 的概率，因此 *input* 中每个元素的数值范围都应当在 $[0, 1]$ 之间。支持的数据类型：float16、float32、float64、bfloat16（仅 Atlas A2 训练系列产品支持）。

关键字参数：

- **generator** (*mindspore.Generator*, 可选) - 伪随机数生成器。默认值：None，使用默认伪随机数生成器。

返回：

- **output** (Tensor) - 输出张量，其 *shape* 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *input* 的数据类型不是 float16、float32、float64、bfloat16 之一。
- **ValueError** - *input* 中任意一个元素的数值范围不在 0 到 1 之间。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> input_x = Tensor(np.ones((3, 3)), mindspore.float32)
>>> output = mint.bernoulli(input_x)
>>> print(output)
[[ 1.  1.  1.]
 [ 1.  1.  1.]
 [ 1.  1.  1.]]
>>> input_x = Tensor(np.zeros((3, 3)), mindspore.float32)
>>> output = mint.bernoulli(input_x)
>>> print(output)
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
```

mindspore.mint.bincount

`mindspore.mint.bincount(input, weights=None, minlength=0)`

统计 *input* 中每个值的出现次数。

如果不指定 *minlength*，输出 Tensor 的长度为输入 *input* 中最大值加 1。如果指定了 *minlength*，则输出 Tensor 的长度为 *input* 中最大值加 1 和 *minlength* 之间的最大值。

输出 Tensor 中每个值标记了该索引值在 *input* 中的出现次数。如果指定了 *weights*，对输出的结果进行加权处理，即 $out[n] += weight[i]$ 而不是 $out[n] += 1$ 。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 一维的 Tensor。
- **weights** (Tensor, 可选) - 权重，与 *input* shape 相同。默认值：None。
- **minlength** (int, 可选) - 输出 Tensor 的最小长度，应为非负数。默认值：0。

返回：

Tensor，如果输入为非空，输出 shape 为 $(\max(\max(input) + 1, minlength),)$ ，否则 shape 为 (0,)。

异常：

- **TypeError** - 如果 *input* 或 *weights* 不是 Tensor。
- **ValueError** - 如果 *input* 中存在负数。
- **ValueError** - 如果 *input* 不是一维的，或者 *input* 和 *weights* 不具有相同的 shape。

支持平台:

Ascend

样例:

```
>>> from mindspore import mint, Tensor
>>> print(mint.bincount(Tensor(np.arange(5))))
[1 1 1 1 1]
>>> print(mint.bincount(Tensor(np.array([0, 1, 1, 3, 2, 1, 7]))))
[1 3 1 1 0 0 0 1]
>>> w = Tensor(np.array([0.3, 0.5, 0.2, 0.7, 1., -0.6])) # weights
>>> x = Tensor(np.array([0, 1, 1, 2, 2, 2]))
>>> print(mint.bincount(x, weights=w, minlength=5))
[0.3 0.7 1.1 0. 0.]
```

mindspore.mint.clone

`mindspore.mint.clone(input)`

返回一个输入 Tensor 的副本。

警告: 这是一个实验性 API, 后续可能修改或删除。

说明: 该函数是可微分的, 梯度将直接从该函数的计算结果流向 *input*。

参数:

- **input** (Tensor) - 输入 Tensor。

返回:

Tensor, 其数据、shape 和数据类型与输入 *input* 相同。

异常:

- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.ones((3,3)).astype("float32"))
>>> output = mint.clone(input)
>>> print(output)
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

mindspore.mint.eye

`mindspore.mint.eye(n, m=None, dtype=None)`

返回一个主对角线上元素为 1，其余元素为 0 的 `tensor`。

参数：

- **n** (int) - 返回的行数。
- **m** (int, 可选) - 返回的列数。如果为 `None`，与 **n** 相等。默认 `None`。
- **dtype** (`mindspore.dtype`, 可选) - 返回的数据类型。默认 `None`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> output = mindspore.mint.eye(3)
>>> print(output)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
>>>
>>> output = mindspore.mint.eye(3, 4)
>>> print(output)
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

mindspore.mint.einsum

`mindspore.mint.einsum(equation, *operands)`

基于爱因斯坦求和约定（Einsum）符号，沿着指定维度对输入 `Tensor` 元素的乘积求和。可以使用这个运算符来执行对角线、减法、转置、矩阵乘法、乘法、内积运算等等。

说明：现在支持子列表模式。例如，`mint.einsum(op1, sublist1, op2, sublist2, ..., sublist_out)`。在子列表模式中，*equation* 由子列表推导得到，Python 的省略号和介于 [0, 52) 的整数 `list` 组成子列表。每个操作数后面都有一个子列表，并且最后有一个表示输出的子列表。动态 `shape`、动态 `rank` 的输入不支持在图模式 (`mode=mindspore.GRAPH_MODE`) 下执行。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **equation** (str) - 基于爱因斯坦求和约定的符号，表示想要执行的操作。符号只能包含字母（必须在 [a-zA-Z] 中）、逗号、省略号和箭头。字母表示输入 `Tensor` 维数，逗号表示单独的 `Tensor`，省略号表示忽略的 `Tensor` 维数，箭头的左边表示输入 `Tensor`，右边表示期望输出的维度。如果方程中没有箭头，则方程中仅出现一次的字母将代表一部分的输出的维度，并按字母顺序升序排序。此时输出是先将输入操作数维度根据字母

对齐后按元素相乘，然后将不属于输出字母对应的维度进行相加。如果方程中有一个箭头，则代表输出的字母必须在输入字母中出现至少一次，而在输出字母中只能最多出现一次。

- **operands** (Tensor) - 用于计算的输入 Tensor。Tensor 的数据类型必须相同。

返回：

Tensor, shape 可以根据 *equation* 得到。数据类型和输入 Tensor 相同。

异常：

- **TypeError** - *equation* 无效或者不匹配输入 Tensor。
- **ValueError** - 子列表模式下子列表的数字不介于 [0, 52) 之间。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([1.0, 2.0, 4.0]), mindspore.float32)
>>> equation = "i->"
>>> output = mint.einsum(equation, x)
>>> print(output)
7.0
>>> x = Tensor(np.array([1.0, 2.0, 4.0]), mindspore.float32)
>>> y = Tensor(np.array([2.0, 4.0, 3.0]), mindspore.float32)
>>> equation = "i,i->i"
>>> output = mint.einsum(equation, x, y)
>>> print(output)
[ 2.  8. 12.]
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> y = Tensor(np.array([[2.0, 3.0], [1.0, 2.0], [4.0, 5.0]]), mindspore.float32)
>>> equation = "ij,jk->ik"
>>> output = mint.einsum(equation, x, y)
>>> print(output)
[[16. 22.]
 [37. 52.]]
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> equation = "ij->ji"
>>> output = mint.einsum(equation, x)
>>> print(output)
[[1. 4.]
 [2. 5.]
 [3. 6.]]
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> equation = "ij->j"
>>> output = mint.einsum(equation, x)
>>> print(output)
[5. 7. 9.]
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> equation = "...->"
>>> output = mint.einsum(equation, x)
>>> print(output)
21.0
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> y = Tensor(np.array([2.0, 4.0, 1.0]), mindspore.float32)
```

(下页继续)

(续上页)

```

>>> equation = "j,i->ji"
>>> output = mint.einsum(equation, x, y)
>>> print(output)
[[ 2.  4.  1.]
 [ 4.  8.  2.]
 [ 6. 12.  3.]]
>>> x = mindspore.Tensor([1, 2, 3, 4], mindspore.float32)
>>> y = mindspore.Tensor([1, 2], mindspore.float32)
>>> output = mint.einsum(x, [..., 1], y, [..., 2], [..., 1, 2])
>>> print(output)
[[1. 2.]
 [2. 4.]
 [3. 6.]
 [4. 8.]]

```

mindspore.mint.empty

`mindspore.mint.empty(*size, dtype=None, device=None)`

创建一个数据没有初始化的 Tensor。参数 *size* 指定 Tensor 的 shape，参数 *dtype* 指定填充值的数据类型，参数 *device* 指定 Tensor 使用的内存来源。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **size** (Union[tuple[int], list[int], int]) - 指定输出 Tensor 的 shape，可以是可变数量的正整数或包含正整数的 tuple、list。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 用来描述所创建的 Tensor 的 dtype。如果为 None，那么将会使用 *mindspore.float32*。默认值：None。
- **device** (string, 可选) - 指定 Tensor 使用的内存来源。当前支持 CPU 和 Ascend。如果为 None，那么将会使用 *mindspore.set_device()* 设置的值。默认值 None。

返回：

Tensor，shape 和 dtype 由输入定义。

异常：

- **TypeError** - 如果 *size* 不是一个 int，或元素为 int 的元组、列表。

支持平台：

Ascend

样例：

```

>>> import mindspore
>>> from mindspore import mint
>>> output = mint.empty((2, 3), dtype=mindspore.float32)
>>> print(output)
[[0. 0. 0.]
 [0. 0. 0.]]

```

mindspore.mint.empty_like

`mindspore.mint.empty_like(input, *, dtype=None, device=None)`

创建一个未初始化的 Tensor，shape 和 `input` 相同，`dtype` 由 `dtype` 决定，Tensor 使用的内存由 `device` 决定。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 任意维度的 Tensor。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 用来描述所创建的 Tensor 的 `dtype`。如果为 `None`，那么将会使用 `input` 的 `dtype`。默认值：`None`。
- **device** (string, 可选) - 指定 Tensor 使用的内存来源。当前支持 CPU 和 Ascend。如果为 `None`，那么将会使用 `input` 的内存来源，如果 `input` 没有申请内存，将会使用 `mindspore.set_device()` 设置的值。默认值 `None`。

返回：

Tensor，具有与 `input` 相同的 shape，但是数据内容没有初始化（可能是任意值）。

异常：

- **TypeError** - `input` 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import mint, Tensor
>>> x = Tensor([[1, 2, 3], [4, 5, 6]])
>>> output1 = mint.empty_like(x)
>>> print(output1)
[[0 0 0]
 [0 0 0]]
>>> output2 = mint.empty_like(x, dtype=mindspore.float64)
>>> print(output2)
[[0. 0. 0.]
 [0. 0. 0.]]
```

mindspore.mint.full

`mindspore.mint.full(size, fill_value, *, dtype=None)`

创建一个指定 shape 的 Tensor，并用指定值填充。

参数：

- **size** (Union(tuple[int], list[int])) - 指定输出 Tensor 的 shape。
- **fill_value** (Union(number.Number, Tensor)) - 用来填充输出 Tensor 的值。可以是一个标量值、0-D 的 Tensor 或只有单个元素的 1-D 的 Tensor。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 指定输出 Tensor 的数据类型。数据类型只支持 *bool_* 和 *number* , 更多细节详见 *mindspore.dtype* 。默认值: `None` 。

返回:

Tensor。

异常:

- **TypeError** - *size* 不是元组或列表。
- **ValueError** - *size* 中包含小于 0 的成员。

支持平台:

Ascend

样例:

```
>>> from mindspore import mint
>>> output = mint.full((2, 2), 1)
>>> print(output)
[[1. 1.]
 [1. 1.]]
>>> output = mint.full((3, 3), 0)
>>> print(output)
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

mindspore.mint.full_like

`mindspore.mint.full_like(input, fill_value, *, dtype=None)`

返回一个 shape 与 *input* 相同并且使用 *fill_value* 填充的 Tensor。

参数:

- **input** (Tensor) - 输入 Tensor, 输出 Tensor 与 *input* 具有相同的 shape。
- **fill_value** (Number) - 用来填充输出 Tensor 的值。当前不支持复数类型。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 指定输出 Tensor 的数据类型。数据类型只支持 *bool_* 和 *number* , 更多细节详见 *mindspore.dtype* 。默认值: `None` 。

返回:

Tensor。

异常:

- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([[0, 1], [2, 1]], dtype=mindspore.int32)
```

(下页继续)

(续上页)

```
>>> output = mint.full_like(input, 1)
>>> print(output)
[[1 1]
 [1 1]]
>>> input = Tensor([[0, 1, 1], [2, 1, 2], [1, 3, 4]], dtype=mindspore.int32)
>>> output = mint.full_like(input, 0, dtype=mindspore.float32)
>>> print(output)
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

mindspore.mint.linspace

`mindspore.mint.linspace` (*start, end, steps, *, dtype=None*)

创建一个 `steps` 个元素的, 在 `[start, end]` 区间内均匀分布的一维 tensor。

$$step = (end - start) / (steps - 1)$$

$$output = [start, start + step, start + 2 * step, ..., end]$$

警告: Atlas 训练系列产品暂不支持 int16 数据类型。

参数:

- **start** (`Union[float, int]`) - 区间的起始值。
- **end** (`Union[float, int]`) - 区间的末尾值。
- **steps** (`int`) - 元素数量。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 返回的数据类型。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> output = mindspore.mint.linspace(3, 10, 5)
>>> print(output)
[ 3.    4.75  6.5   8.25 10. ]
>>> output = mindspore.mint.linspace(-10, 10, 5)
>>> print(output)
[-10.  -5.   0.   5.  10.]
>>> output = mindspore.mint.linspace(-10, 10, 1)
>>> print(output)
[-10.]
```

mindspore.mint.ones

`mindspore.mint.ones(size, *, dtype=None)`

创建一个值全为 1 的 Tensor。第一个参数 *size* 指定 Tensor 的 shape，第二个参数 *dtype* 指定填充值的数据类型。

参数：

- **size** (`Union[tuple[int], list[int], int, Tensor]`) - 指定输出 Tensor 的 shape，只允许正整数或者包含正整数的 tuple、list、Tensor。如果是一个 Tensor，必须是一个数据类型为 int32 或者 int64 的 0-D 或 1-D Tensor。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 用来描述所创建的 Tensor 的 dtype。如果为 None，那么将会使用 `mindspore.float32`。默认值：None。

返回：

Tensor，shape 和 dtype 由输入定义。

异常：

- **TypeError** - 如果 *size* 不是一个 int，或元素为 int 的元组、列表、Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import mint
>>> output = mint.ones((2, 2), dtype=mindspore.float32)
>>> print(output)
[[1. 1.]
 [1. 1.]]
```

mindspore.mint.ones_like

`mindspore.mint.ones_like(input, *, dtype=None)`

创建一个数值全为 1 的 Tensor，shape 和 *input* 相同，dtype 由 *dtype* 决定。

如果 *dtype* = None，输出 Tensor 的数据类型会和 *input* 一致。

参数：

- **input** (Tensor) - 任意维度的 Tensor。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 用来描述所创建的 Tensor 的 dtype。如果为 None，那么将会使用 *input* 的 dtype。默认值：None。

返回：

Tensor，具有与 *input* 相同的 shape 并填充了 1。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[0, 1], [2, 1]]).astype(np.int32))
>>> output = mint.ones_like(x)
>>> print(output)
[[1 1]
 [1 1]]
```

mindspore.mint.randint

`mindspore.mint.randint` (*low=0, high, size, *, generator=None, dtype=None*)

返回一个 Tensor, shape 和 dtype 由输入决定, 其元素为 [*low*, *high*) 区间的随机整数。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **low** (int, 可选) - 随机区间的起始值。默认值: 0。
- **high** (int) - 随机区间的结束值。
- **size** (Union[tuple(int), list(int)]) - 新 Tensor 的 shape, 如 (2, 3)。

关键字参数:

- **generator** (*mindspore.Generator*, 可选) - 伪随机数生成器。默认值: None, 使用默认伪随机数生成器。
- **dtype** (*mindspore.dtype*, 可选) - 指定的 Tensor dtype。如果是 None, 将会使用 *mindspore.int64*。默认值: None。

返回:

Tensor, shape 和 dtype 被输入指定, 其元素为 [*low*, *high*) 区间的随机整数。

异常:

- **TypeError** - 如果 *size* 不是 tuple。
- **TypeError** - 如果 *low* 或 *high* 不是整数。

支持平台:

Ascend

样例:

```
>>> from mindspore import mint
>>> print(mint.randint(0, 5, (2, 3)).shape)
(2, 3)
```

mindspore.mint.randint_like

`mindspore.mint.randint_like(input, low=0, high, *, dtype=None)`

返回一个 Tensor，其元素为 $[low, high)$ 区间的随机整数，根据 *input* 决定 shape 和 dtype。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入的 Tensor，用来决定输出 Tensor 的 shape 和默认的 dtype。
- **low** (int, 可选) - 随机区间的起始值。默认值：0。
- **high** (int) - 随机区间的结束值。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 指定的 Tensor dtype。如果是 None，将会使用 *input* 的 dtype。默认值：None。

返回：

Tensor，shape 和 dtype 被输入指定，其元素为 $[low, high)$ 区间的随机整数。

异常：

- **TypeError** - 如果 *low* 或 *high* 不是整数。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, mint
>>> a = Tensor([[2, 3, 4], [1, 2, 3]])
>>> low = 0
>>> high = 5
>>> print(mint.randint_like(a, low, high, dtype=ms.int32).shape)
(2, 3)
```

mindspore.mint.randn

`mindspore.mint.randn(*size, generator=None, dtype=None)`

返回一个 Tensor，shape 和 dtype 由输入决定，其元素为服从标准正态分布的数字。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **size** (Union[int, tuple(int), list(int)]) - 输出的 Tensor 的 shape，例如，(2,3) or 2。

关键字参数：

- **generator** (*mindspore.Generator*, 可选) - 伪随机数生成器。默认值：None，使用默认伪随机数生成器。

- **dtype** (*mindspore.dtype*, 可选) - 需求的输出 Tensor 的 dtype。如果是 None, *mindspore.float32* 会被使用。默认值: None。

返回:

Tensor, shape 和 dtype 由输入决定其元素为服从标准正态分布的数字。

异常:

- **ValueError** - 如果 *size* 包含负数。

支持平台:

Ascend

样例:

```
>>> from mindspore import mint
>>> print(mint.randn(2, 3).shape)
(2, 3)
```

mindspore.mint.randn_like

`mindspore.mint.randn_like(input, *, dtype=None)`

返回 shape 与输入相同, 类型为 *dtype* 的 Tensor。dtype 由输入决定, 其元素取值服从 [0, 1) 区间内的正态分布。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入的 Tensor。用来决定输出 Tensor 的 shape 和默认的 dtype。

关键字参数:

- **dtype** (*mindspore.dtype*, 可选) - 指定输出 Tensor 的 dtype, 必须是 float 类型。如果是 None, 则使用输入 Tensor 的 dtype。默认值: None。

返回:

Tensor, shape 和 dtype 由输入决定其元素为服从标准正态分布的数字。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import Tensor, mint
>>> a = Tensor([[2, 3, 4], [1, 2, 3]])
>>> print(mint.randn_like(a, dtype=ms.float32).shape)
(2, 3)
```

mindspore.mint.randperm

`mindspore.mint.randperm(n, *, generator=None, dtype=mstype.int64)`

生成从 0 到 $n-1$ 的整数随机排列。

警告:

- 这是一个实验性 API，后续可能修改或删除。

参数:

- **n** (`Union[Tensor, int]`) - 随机排列长度。int 或 shape 为 () 或 (1,)，数据类型为 int64 的 Tensor。数值必须大于 0。

关键字参数:

- **generator** (`mindspore.Generator`, 可选) - 伪随机数生成器。默认值: None，使用默认伪随机数生成器。
- **dtype** (`mindspore.dtype`, 可选) - 输出的 dtype。默认值: `mstype.int64`。

返回:

shape 为 $(n,)$ ，dtype 为 *dtype* 的 Tensor。

异常:

- **TypeError** - 如果 *dtype* 不是一个 *mstype* 类型。
- **ValueError** - 如果 n 是负数或 0。
- **ValueError** - 如果 n 是超过指定数据类型的最大范围。

支持平台:

Ascend

样例:

```
>>> from mindspore import mint
>>> from mindspore import dtype as mstype
>>> n = 4
>>> output = mint.randperm(n, dtype=mstype.int64)
>>> print(output.shape)
(4,)
```

mindspore.mint.zeros

`mindspore.mint.zeros(size, *, dtype=None)`

创建一个值全为 0 的 Tensor。第一个参数 *size* 指定 Tensor 的 shape，第二个参数 *dtype* 指定填充值的数据类型。

参数:

- **size** (`Union[tuple[int], list[int], int, Tensor]`) - 用来描述所创建的 Tensor 的 shape。需为正整数或者包含正整数的 tuple、list、Tensor。若为 Tensor，则必须是一个数据类型为 int32 或者 int64 的 0-D 或 1-D Tensor。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 用来描述所创建的 Tensor 的 dtype。如果为 None，则会使用 `mindspore.float32`。默认值: None。

返回:

Tensor, dtype 和 shape 由入参决定。

异常:

- **TypeError** - 如果 *size* 既不是一个 int, 也不是一个包含 int 的 tuple、list 或 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import mint
>>> output = mint.zeros((2, 2), dtype=mindspore.float32)
>>> print(output)
[[0. 0.]
 [0. 0.]]
```

mindspore.mint.zeros_like

`mindspore.mint.zeros_like(input, *, dtype=None)`

创建一个数值全为 0 的 Tensor, shape 和 *input* 相同, dtype 由 *dtype* 决定。

如果 *dtype* = None, 输出 Tensor 的数据类型会和 *input* 一致。

参数:

- **input** (Tensor) - 用来描述所创建的 Tensor 的 shape。

关键字参数:

- **dtype** (*mindspore.dtype*, 可选) - 用来描述所创建的 Tensor 的 *dtype*。如果为 None, 那么将会使用 *input* 的 dtype。默认值: None。

返回:

返回一个用 0 填充的 Tensor。

异常:

- **TypeError** - 如果 *dtype* 不是 MindSpore 的 dtype。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.arange(4).reshape(2, 2))
>>> output = mint.zeros_like(x, dtype=mindspore.float32)
>>> print(output)
[[0. 0.]
 [0. 0.]]
```

4.1.2 索引、切分、连接、突变运算

接口名	概述	支持平台
<code>mindspore.mint.cat</code>	在指定维度上拼接输入 Tensor。	Ascend
<code>mindspore.mint.chunk</code>	沿着指定轴 <i>dim</i> 将输入 Tensor 切分成 <i>chunks</i> 个 sub-tensor。	Ascend
<code>mindspore.mint.concat</code>	<code>mindspore.mint.cat()</code> 的别名。	Ascend
<code>mindspore.mint.count_nonzero</code>	在给定维度 <i>dim</i> 上, 计算 Tensor <i>input</i> 中非零元素的数量。	Ascend
<code>mindspore.mint.gather</code>	返回输入 Tensor 在指定 <i>index</i> 索引上对应元素组成的切片。	Ascend
<code>mindspore.mint.index_add</code>	根据 <i>index</i> 中的索引顺序, 将 <i>alpha</i> 乘以 <i>source</i> 的元素累加到 <i>input</i> 中。	Ascend
<code>mindspore.mint.index_select</code>	返回一个新的 Tensor, 该 Tensor 沿维度 <i>dim</i> 按 <i>index</i> 中给定的索引对 <i>input</i> 进行选择。	Ascend
<code>mindspore.mint.masked_select</code>	返回一个一维 tensor, 它根据掩码对输入 tensor 进行索引。	Ascend
<code>mindspore.mint.permute</code>	按照输入 <i>dims</i> 的维度顺序排列输入 Tensor。	Ascend
<code>mindspore.mint.reshape</code>	按指定 <i>shape</i> , 对输入 tensor 进行重排。	Ascend
<code>mindspore.mint.scatter</code>	根据指定索引将 <i>src</i> 中的值更新到 <i>input</i> 中返回输出。	Ascend
<code>mindspore.mint.scatter_add</code>	将 <i>src</i> 中所有的元素添加到 <i>input</i> 中 <i>index</i> 指定的索引处。	Ascend
<code>mindspore.mint.split</code>	根据指定的轴将输入 Tensor 切分成块。	Ascend
<code>mindspore.mint.narrow</code>	沿着指定的轴, 指定起始位置获取指定长度的 Tensor。	Ascend
<code>mindspore.mint.nonzero</code>	返回所有非零元素下标位置。	Ascend
<code>mindspore.mint.tile</code>	通过复制 <i>dims</i> 次输入 tensor 中的元素来创建新 tensor。	Ascend
<code>mindspore.mint.tril</code>	将指定对角线上方的元素设置为 0。	Ascend
<code>mindspore.mint.select</code>	在给定索引处沿选定维度对输入 Tensor 进行切片。	Ascend
<code>mindspore.mint.squeeze</code>	返回删除指定 <i>dim</i> 中大小为 1 的维度后的 Tensor。	Ascend
<code>mindspore.mint.stack</code>	在指定轴上对输入 Tensor 序列进行堆叠。	Ascend
<code>mindspore.mint.swapaxes</code>	<code>mindspore.mint.transpose()</code> 的别名。	Ascend
<code>mindspore.mint.transpose</code>	交换 Tensor 的两个维度。	Ascend
<code>mindspore.mint.triu</code>	将指定对角线下方的元素设置为 0。	Ascend
<code>mindspore.mint.unbind</code>	根据指定轴对 Tensor 进行分解。	Ascend
<code>mindspore.mint.unique_consecutive</code>	对输入 Tensor 中连续且重复的元素去重。	Ascend
<code>mindspore.mint.unsqueeze</code>	在输入 tensor 的指定维度上增加一维。	Ascend
<code>mindspore.mint.where</code>	返回一个 Tensor, Tensor 的元素从 <i>input</i> 或 <i>other</i> 中根据 <i>condition</i> 选择。	Ascend

mindspore.mint.cat

`mindspore.mint.cat(tensors, dim=0)`

在指定维度上拼接输入 Tensor。

输入的是一个 tuple 或 list。其元素秩相同，即 R 。将给定的维度设为 m ，并且 $0 \leq m < R$ 。输入元素的数量设为 N 。对于第 i 个数据， t_i 的 shape 为 $(x_1, x_2, \dots, x_{mi}, \dots, x_R)$ 。 x_{mi} 是第 t_i 个元素的第 m 个维度。则，输出 Tensor 的 shape 为：

$$(x_1, x_2, \dots, \sum_{i=1}^N x_{mi}, \dots, x_R)$$

参数：

- **tensors** (Union[tuple, list]) - 输入为 Tensor 组成的 tuple 或 list。假设在这个 tuple 或 list 中有两个 Tensor，即 $t1$ 和 $t2$ 。要在第 0 个维度方向上执行 *cat*，除第 0 维度外，其他维度的 shape 都应相等，即 $t1.shape[1] = t2.shape[1], t1.shape[2] = t2.shape[2], \dots, t1.shape[R-1] = t2.shape[R-1]$ ，其中 R 是 Tensor 的秩。
- **dim** (int, 可选) - 表示指定的维度，取值范围是 $[-R, R)$ 。默认值：0。

返回：

Tensor，shape 为 $(x_1, x_2, \dots, \sum_{i=1}^N x_{mi}, \dots, x_R)$ 。数据类型与 *tensors* 相同。

异常：

- **TypeError** - *dim* 不是 int。
- **ValueError** - *tensors* 是不同维度的 Tensor。
- **ValueError** - *dim* 的值不在区间 $[-R, R)$ 内。
- **ValueError** - 除了 *dim* 之外，*tensors* 的 shape 不相同。
- **ValueError** - *tensors* 为空 tuple 或 list。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> input_x1 = Tensor(np.array([[0, 1], [2, 1]]).astype(np.float32))
>>> input_x2 = Tensor(np.array([[0, 1], [2, 1]]).astype(np.float32))
>>> output = mint.cat((input_x1, input_x2))
>>> print(output)
[[0. 1.]
 [2. 1.]
 [0. 1.]
 [2. 1.]]
>>> output = mint.cat((input_x1, input_x2), 1)
>>> print(output)
[[0. 1. 0. 1.]
 [2. 1. 2. 1.]]
```

mindspore.mint.chunk

`mindspore.mint.chunk(input, chunks, dim=0)`

沿着指定轴 *dim* 将输入 Tensor 切分成 *chunks* 个 sub-tensor。

说明：此函数返回的数量可能小于通过 *chunks* 指定的数量。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 被切分的 Tensor。
- **chunks** (int) - 要切分的 sub-tensor 数量。
- **dim** (int, 可选) - 指定需要分割的维度。默认值：0。

返回：

tuple[Tensor]。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *dim* 不是 int 类型。
- **TypeError** - *chunks* 不是 int。
- **ValueError** - 参数 *dim* 超出 $[-input.ndim, input.ndim)$ 范围。
- **ValueError** - 参数 *chunks* 不是正数。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> input_x = np.arange(9).astype("float32")
>>> output = mindspore.mint.chunk(Tensor(input_x), 3)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  1.00000000e+00,  2.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 3.00000000e+00,  4.00000000e+00,  5.
↪00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 6.00000000e+00,  7.00000000e+00,  8.
↪00000000e+00]))
```

mindspore.mint.concat

`mindspore.mint.concat (tensors, dim=0)`
`mindspore.mint.cat()` 的别名。

警告：这是一个实验性 API，后续可能修改或删除。

支持平台：

Ascend

mindspore.mint.count_nonzero

`mindspore.mint.count_nonzero (input, dim=None)`
 在给定维度 `dim` 上，计算 Tensor `input` 中非零元素的数量。如果未指定维度，则计算 Tensor 中所有非零元素的数量。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 计算非零元素个数的输入。其 shape 为 (*)，其中 * 为任意维度。
- **dim** (Union[None, int, tuple(int), list(int)], 可选) - 计算非零元素数量的维度。默认值：None，表示计算所有非零元素的个数。如果 `dim` 为 None，则对 Tensor 中的所有元素求和。

返回：

Tensor，指定的轴上非零元素数量。

异常：

- **TypeError** - `input` 不是 Tensor 类型。
- **TypeError** - `dim` 类型不是 int、tuple(int)、list(int) 或 None。
- **ValueError** - `dim` 取值不在 `[-input.ndim, input.ndim)` 范围。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> import mindspore
>>> # case 1: each value specified.
>>> x = Tensor(np.array([[0, 1, 0], [1, 1, 0]]).astype(np.float32))
>>> nonzero_num = mint.count_nonzero(input=x, dim=[0, 1])
>>> print(nonzero_num)
3
>>> # case 2: all value is default.
>>> nonzero_num = mint.count_nonzero(input=x)
>>> print(nonzero_num)
3
>>> # case 3: dim value was specified 0.
```

(下页继续)

(续上页)

```
>>> nonzero_num = mint.count_nonzero(input=x, dim=[0,])
>>> print(nonzero_num)
[1 2 0]
>>> # case 4: dim value was specified 1.
>>> nonzero_num = mint.count_nonzero(input=x, dim=[1,])
>>> print(nonzero_num)
[1 2]
```

mindspore.mint.gather

`mindspore.mint.gather(input, dim, index)`

返回输入 Tensor 在指定 *index* 索引上对应元素组成的切片。

$$output[(i_0, i_1, \dots, i_{dim}, i_{dim+1}, \dots, i_n)] = input[(i_0, i_1, \dots, index[(i_0, i_1, \dots, i_{dim}, i_{dim+1}, \dots, i_n)], i_{dim+1}, \dots, i_n)]$$

警告： 在 Ascend 后端，以下场景将导致不可预测的行为：

- 正向执行流程中，当 *index* 的取值不在范围 $[-input.shape[dim], input.shape[dim])$ 内；
- 反向执行流程中，当 *index* 的取值不在范围 $[0, input.shape[dim])$ 内。

参数：

- **input** (Tensor) - 待索引切片取值的原始 Tensor。
- **dim** (int) - 指定要切片的维度索引。取值范围 $[-input.rank, input.rank)$ 。
- **index** (Tensor) - 指定原始 Tensor 中要切片的索引。数据类型必须是 int32 或 int64。需要同时满足以下条件：
 - $index.rank == input.rank$;
 - 对于 $axis \neq dim$, $index.shape[axis] \leq input.shape[axis]$;
 - *index* 的取值在有效区间 $[-input.shape[dim], input.shape[dim])$;

返回：

Tensor，数据类型与 *input* 保持一致，*shape* 与 *index* 保持一致。

异常：

- **ValueError** - *input* 的 *shape* 取值非法。
- **ValueError** - *dim* 取值不在有效范围 $[-input.rank, input.rank)$ 。
- **ValueError** - *index* 的值不在有效范围。
- **TypeError** - *index* 的数据类型非法。

支持平台：

Ascend

样例：


```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]), mindspore.float32)
>>> index = Tensor(np.array([[0, 0], [1, 1]]), mindspore.int32)
>>> output = mint.gather(input, 1, index)
>>> print(output)
[[-0.1 -0.1]
 [0.5   0.5]]
```

mindspore.mint.index_add

`mindspore.mint.index_add(input, dim, index, source, alpha=1)`

根据 *index* 中的索引顺序, 将 *alpha* 乘以 *source* 的元素累加到 *input* 中。例如, 如果 *dim* == 0, *index*[*i*] == *j*, 且 *alpha* = -1, 那么 *source* 的第 *i* 行将从 *input* 的第 *j* 行中被减去。*source* 的第 *dim* 维度必须与 *index* 的长度相同, 且所有其他维度必须与 *input* 匹配, 否则将引发错误。对于一个三维张量, 输出定义如下:

$$\begin{aligned} \text{input}[\text{index}[i], :, :] &+= \alpha * \text{source}[i, :, :] && \text{if dim} == 0 \\ \text{input}[:, \text{index}[i], :] &+= \alpha * \text{source}[:, i, :] && \text{if dim} == 1 \\ \text{input}[:, :, \text{index}[i]] &+= \alpha * \text{source}[:, :, i] && \text{if dim} == 2 \end{aligned}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 *Tensor*。
- **dim** (int) - 指定 *index* 属于哪个维度。
- **index** (Tensor) - 指定 *input* 和 *source* 在轴 *dim* 的指定下标位置相加, 数据类型为 *int32*。要求 *index* shape 的维度为一维, 并且 *index* shape 的大小与 *source* shape 在 *dim* 轴上的大小一致。*index* 中元素取值范围为 [0, *b*), 其中 *b* 的值为 *input* shape 在 *dim* 轴上的大小。
- **source** (Tensor) - 输入的要进行相加的 *Tensor*, 一定要与 *input* 有相同的数据类型, 与 *input* 在 *dim* 维度有相同的 shape。
- **alpha** (number, 可选) - *source* 的乘数。默认值: 1。

返回:

相加后的 *Tensor*。shape 和数据类型与输入 *input* 相同。

异常:

- **TypeError** - *index* 或者 *source* 的类型不是 *Tensor*。
- **ValueError** - *dim* 的值超出 *source* shape 的维度范围。
- **ValueError** - *index* shape 的维度和 *source* shape 的维度不一致。
- **ValueError** - *index* shape 的维度不是 1D 或者 *index* shape 的大小与 *source* shape 在 *dim* 轴上的大小不一致。
- **ValueError** - 除 *dim* 轴外, *input* shape 和 *source* shape 的大小不一致。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.float32)
>>> index = Tensor(np.array([0, 2]), mindspore.int32)
>>> y = Tensor(np.array([[0.5, 1.0], [1.0, 1.5], [2.0, 2.5]]), mindspore.float32)
>>> output = mint.index_add(x, 1, index, y, alpha=1)
>>> print(output)
[[ 1.5  2.   4. ]
 [ 5.   5.   7.5]
 [ 9.   8.  11.5]]
```

mindspore.mint.index_select

`mindspore.mint.index_select(input, dim, index)`

返回一个新的 Tensor，该 Tensor 沿维度 *dim* 按 *index* 中给定的索引对 *input* 进行选择。

返回的 Tensor 和输入 Tensor(*input*) 的维度数量相同，其第 *dim* 维度的大小和 *index* 的长度相同；其他维度和 *input* 相同。

说明：index 的值必须在 $[0, \text{input.shape}[dim])$ 范围内，超出该范围结果未定义。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (int) - 根据索引进行选择的维度。
- **index** (Tensor) - 包含索引的一维 Tensor。

返回：

Tensor，数据类型与输入 *input* 相同。

异常：

- **TypeError** - *input* 或 *index* 的类型不是 Tensor。
- **TypeError** - *dim* 的类型不是 int。
- **ValueError** - *dim* 值超出范围 $[-\text{input.ndim}, \text{input.ndim} - 1]$ 。
- **ValueError** - *index* 不是一维 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.arange(16).astype(np.float32).reshape(2, 2, 4))
>>> print(input)
[[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
```

(下页继续)

(续上页)

```
[12. 13. 14. 15.]]]
>>> index = Tensor([0,], mindspore.int32)
>>> y = mint.index_select(input, 1, index)
>>> print(y)
[[[ 0.  1.  2.  3.]]
 [[ 8.  9. 10. 11.]]]
```

mindspore.mint.masked_select

mindspore.mint.masked_select (*input, mask*)

返回一个一维 **tensor**，它根据掩码对输入 **tensor** 进行索引。

支持广播。

参数：

- **input** (**Tensor**) - 输入 **tensor**。
- **mask** (**Tensor[bool]**) - 输入掩码。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([1, 2, 3, 4], mindspore.int64)
>>> mask = mindspore.tensor([1, 0, 1, 0], mindspore.bool_)
>>> output = mindspore.mint.masked_select(x, mask)
>>> print(output)
[1 3]
```

mindspore.mint.permute

mindspore.mint.permute (*input, dims*)

按照输入 *dims* 的维度顺序排列输入 **Tensor**。

参数：

- **input** (**Tensor**) - 输入 **Tensor**。
- **dims** (**tuple(int)**) - 维度的顺序，**permute** 根据 *dims* 的顺序重新排列 *input*。

返回：

Tensor，具有和输入 **Tensor** 相同的维数，按照 *dims* 重新排列。

异常：

- **ValueError** - *dims* 为 **None**。
- **ValueError** - *dims* 的元素总量不等于 *input* 的维数。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input_x = Tensor(np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]]),
↳mindspore.float32)
>>> input_perm = (0, 2, 1)
>>> print(mint.permute(input_x, input_perm))
[[[ 1.  4.]
 [ 2.  5.]
 [ 3.  6.]]
 [[ 7. 10.]
 [ 8. 11.]
 [ 9. 12.]]]
```

mindspore.mint.reshape

`mindspore.mint.reshape(input, shape)`
按指定 *shape* , 对输入 tensor 进行重排。

说明: 参数 *shape* 中的-1 表示该维度值是从其他维度和输入的元素数量推断出来的。

参数:

- **input** (Tensor) - 输入 tensor。
- **shape** (Union[tuple[int], list[int], Tensor[int]]) - 新 shape。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]], mindspore.float32)
>>> # case1: Parameter `shape` does not contain -1.
>>> output = mindspore.mint.reshape(input, (3, 2))
>>> print(output)
[[-0.1  0.3]
 [ 3.6  0.4]
 [ 0.5 -3.2]]
>>> # case2: Parameter `shape` contains -1.
>>> output = mindspore.mint.reshape(input, (-1, 6))
>>> print(output)
[[-0.1  0.3  3.6  0.4  0.5 -3.2]]
```

mindspore.mint.scatter

`mindspore.mint.scatter(input, dim, index, src)`

根据指定索引将 *src* 中的值更新到 *input* 中返回输出。对于一个 3D 的 Tensor, 输出形式如下所示:

```

output[index[i][j][k]][j][k] = src[i][j][k]    # if dim == 0
output[i][index[i][j][k]][k] = src[i][j][k]    # if dim == 1
output[i][j][index[i][j][k]] = src[i][j][k]    # if dim == 2

```

说明: 如果 *src* 为 Tensor, 则仅当 *src* 的 shape 和 *index* 的 shape 相同时支持求反向梯度。

参数:

- **input** (Tensor) - 输入 Tensor。*input* 的秩必须至少为 1。
- **dim** (int) - 要进行更新操作的轴。取值范围是 $[-r, r)$, 其中 *r* 是 *input* 的秩。
- **index** (Tensor) - 输入 Tensor 的索引, 数据类型为 int32 或 int64 的正整数。其 rank 必须和 *input* 一致。取值范围是 $[-s, s)$, 这里的 *s* 是 *input* 在 *axis* 指定轴的 size。
- **src** (Tensor, float) - 指定对 *input* 进行更新操作的数据。可以为 Tensor, 此时其数据类型必须与输入 *input* 的数据类型相同。也可以是个 float 类型的标量。

返回:

Tensor, shape 和数据类型与输入 *input* 相同。

异常:

- **TypeError** - *index* 的数据类型既不是 int32, 也不是 int64。
- **ValueError** - *input*、*index* 和 *src* 中, 任意一者的秩小于 1。
- **ValueError** - *src* 的秩和 *input* 的秩不一样。
- **TypeError** - *input* 的数据类型和 *src* 的数据类型不一致。
- **RuntimeError** - *index* 中存在负数。

支持平台:

Ascend

样例:

```

>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[1, 2, 3, 4, 5]]), dtype=ms.float32)
>>> src = Tensor(np.array([[8, 8]]), dtype=ms.float32)
>>> index = Tensor(np.array([[2, 4]]), dtype=ms.int64)
>>> out = mint.scatter(input=input, dim=1, index=index, src=src)
>>> print(out)
[[1. 2. 8. 4. 8.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 0, 0], [2, 2, 2], [4, 4, 4]]), dtype=ms.int64)
>>> out = mint.scatter(input=input, dim=0, index=index, src=src)

```

(下页继续)

(续上页)

```
>>> print(out)
[[1. 2. 3. 0. 0.]
 [0. 0. 0. 0. 0.]
 [4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0.]
 [7. 8. 9. 0. 0.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 2, 4], [0, 2, 4], [0, 2, 4]]), dtype=ms.int64)
>>> out = mint.scatter(input=input, dim=1, index=index, src=src)
>>> print(out)
[[1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6.]
 [7. 0. 8. 0. 9.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

mindspore.mint.scatter_add

`mindspore.mint.scatter_add(input, dim, index, src)`

将 *src* 中所有的元素添加到 *input* 中 *index* 指定的索引处。其中 *dim* 控制 `scatter_add` 操作的轴。*input*、*index*、*src* 三者的 *rank* 都必须大于或等于 1。

下面看一个三维的例子：

```
input[index[i][j][k]][j][k] += src[i][j][k] # if dim == 0
input[i][index[i][j][k]][k] += src[i][j][k] # if dim == 1
input[i][j][index[i][j][k]] += src[i][j][k] # if dim == 2
```

参数：

- **input** (Tensor) - 输入 Tensor。rank 必须大于等于 1。
- **dim** (int) - *input* 执行 `scatter_add` 操作的轴。取值范围是 $[-r, r]$ ，其中 *r* 是 *input* 的 rank。
- **index** (Tensor) - *input* 要进行 `scatter_add` 操作的目标索引。数据类型为 `int32` 或 `int64`，rank 必须和 *input* 一致。除了 *dim* 指定的维度，*index* 的每一维的 size 都需要小于等于 *input* 对应维度的 size。
- **src** (Tensor) - 指定与 *input* 进行 `scatter_add` 操作的 Tensor，其数据类型与 *input* 类型相同，shape 中每一维的 size 必须大于等于 *index*。

返回：

Tensor，shape 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *index* 的数据类型不满足 `int32` 或 `int64`。
- **ValueError** - *input*、*index* 和 *src* 中，任意一者的 rank 小于 1。
- **ValueError** - *input*、*index* 和 *src* 的 rank 不一致。
- **ValueError** - 除了 *dim* 指定的维度，*index* 的任意维的 size 大于 *input* 对应维度的 size。
- **ValueError** - *src* 任意维度 size 小于 *index* 对应维度的 size。

支持平台：

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[1, 2, 3, 4, 5]]), dtype=ms.float32)
>>> src = Tensor(np.array([[8, 8]]), dtype=ms.float32)
>>> index = Tensor(np.array([[2, 4]]), dtype=ms.int64)
>>> out = mint.scatter_add(input=input, dim=1, index=index, src=src)
>>> print(out)
[[1. 2. 11. 4. 13.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 0, 0], [2, 2, 2], [4, 4, 4]]), dtype=ms.int64)
>>> out = mint.scatter_add(input=input, dim=0, index=index, src=src)
>>> print(out)
[[1. 2. 3. 0. 0.]
 [0. 0. 0. 0. 0.]
 [4. 5. 6. 0. 0.]
 [0. 0. 0. 0. 0.]
 [7. 8. 9. 0. 0.]]
>>> input = Tensor(np.zeros((5, 5)), dtype=ms.float32)
>>> src = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), dtype=ms.float32)
>>> index = Tensor(np.array([[0, 2, 4], [0, 2, 4], [0, 2, 4]]), dtype=ms.int64)
>>> out = mint.scatter_add(input=input, dim=1, index=index, src=src)
>>> print(out)
[[1. 0. 2. 0. 3.]
 [4. 0. 5. 0. 6.]
 [7. 0. 8. 0. 9.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

mindspore.mint.split

`mindspore.mint.split(tensor, split_size_or_sections, dim=0)`

根据指定的轴将输入 Tensor 切分成块。

参数:

- **tensor** (Tensor) - 要被切分的 Tensor。
- **split_size_or_sections** (Union[int, tuple(int), list(int)]) - 如果 `split_size_or_sections` 是 int 类型, tensor 将被均匀的切分成块, 每块的大小为 `split_size_or_sections`, 若 `tensor.shape[dim]` 不能被 `split_size_or_sections` 整除, 则最后一块大小将小于 `split_size_or_sections`; 如果 `split_size_or_sections` 是 list 类型, tensor 将沿 `dim` 轴被切分成 `len(split_size_or_sections)` 块, 大小为 `split_size_or_sections`。
- **dim** (int, 可选) - 指定分割轴。默认值: 0。

返回:

元素为 Tensor 的 Tuple。

异常:

- **TypeError** - `tensor` 不是 Tensor。
- **TypeError** - `dim` 不是 int 类型。

- **ValueError** - *dim* 不在 $[-\text{tensor.ndim}, \text{tensor.ndim})$ 范围中。
- **TypeError** - *split_size_or_sections* 中的每个元素都不是 `int` 类型。
- **TypeError** - *split_size_or_sections* 不是 `int`、`tuple(int)` 或 `list(int)`。
- **ValueError** - *split_size_or_sections* 的和不等于 `tensor.shape[dim]`。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import mint, Tensor
>>> input_x = np.arange(9).astype("float32")
>>> output = mint.split(Tensor(input_x), 3)
>>> print(output)
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 3.00000000e+00,  4.00000000e+00,  5.00000000e+00]),
 Tensor(shape=[3], dtype=Float32, value= [ 6.00000000e+00,  7.00000000e+00,  8.00000000e+00]))
```

mindspore.mint.narrow

`mindspore.mint.narrow(input, dim, start, length)`

沿着指定的轴，指定起始位置获取指定长度的 `Tensor`。

参数：

- **input** (`Tensor`) - 需要计算的 `Tensor`。
- **dim** (`int`) - 指定的轴。
- **start** (`Union[int, Tensor[int]]`) - 指定起始位置。
- **length** (`int`) - 指定长度。

返回：

`Tensor`。

异常：

- **ValueError** - *input* 的秩为 0。
- **ValueError** - *dim* 值超出范围 $[-\text{input.ndim}, \text{input.ndim})$ 。
- **ValueError** - *start* 值超出范围 $[-\text{input.shape}[dim], \text{input.shape}[dim]]$ 。
- **ValueError** - *length* 值超出范围 $[0, \text{input.shape}[dim]-\text{start}]$ 。

支持平台：

Ascend

样例：


```
>>> import mindspore
>>> from mindspore import mint
>>> from mindspore import Tensor
>>> x = Tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]], mindspore.int32)
>>> output = mint.narrow(x, 0, 0, 2)
>>> print(output)
[[ 1 2 3]
 [ 4 5 6]]
>>> output = mint.narrow(x, 1, 1, 2)
>>> print(output)
[[ 2 3]
 [ 5 6]
 [ 8 9]]
```

mindspore.mint.nonzero

`mindspore.mint.nonzero(input, *, as_tuple=False)`

返回所有非零元素下标位置。

参数:

- **input** (Tensor) - 输入 tensor。

说明:

- Ascend: 输入 tensor 的秩可以等于 0, jit level O2 模式除外。
- CPU/GPU: 输入 tensor 秩应大于等于 1。

关键字参数:

- **as_tuple** (bool, 可选) - 是否以 tuple 形式输出, 默认 False。

返回:

Tensor 或者由 tensor 组成的 tuple。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([[1, 0], [-5, 0]], mindspore.int32)
>>> output = mindspore.mint.nonzero(x)
>>> print(output)
[[0 0 0]
 [0 1 0]]
>>> x = mindspore.tensor([1, 0, 2, 0, 3], mindspore.int32)
>>> output = mindspore.mint.nonzero(x, as_tuple=False)
>>> print(output)
[[0]
 [2]
 [4]]
>>> x = mindspore.tensor([[1, 0], [-5, 0]], mindspore.int32)
>>> output = mindspore.mint.nonzero(x, as_tuple=True)
```

(下页继续)

(续上页)

```
>>> print(output)
(Tensor(shape=[2], dtype=Int64, value=[0, 0]),
 Tensor(shape=[2], dtype=Int64, value=[0, 1]),
 Tensor(shape=[2], dtype=Int64, value=[0, 0]))
>>> x = mindspore.tensor([1, 0, 2, 0, 3], mindspore.int32)
>>> output = mindspore.mint.nonzero(x, as_tuple=True)
>>> print(output)
(Tensor(shape=[3], dtype=Int64, value=[0, 2, 4]), )
```

mindspore.mint.tile

`mindspore.mint.tile(input, dims)`

通过复制 *dims* 次输入 *tensor* 中的元素来创建新 *tensor*。

返回 *tensor* 的第 *i* 维度有 *input.shape[i] * dims[i]* 个元素，并且 *input* 的值沿第 *i* 维度被复制 *dims[i]* 次。

说明:

- 在 Ascend 平台上，*dims* 参数的个数不大于 8，当前不支持超过 4 个维度同时做被复制的场景。
- 如果 *input.dim = d*，将其相应位置的 *shape* 相乘，返回的 *shape* 为 $(x_1 * y_1, x_2 * y_2, \dots, x_S * y_S)$ 。
- 如果 *input.dim < d*，在 *input* 的 *shape* 的前面填充 1，直到它们的长度一致。例如将 *input* 的 *shape* 设置为 $(1, \dots, x_1, x_2, \dots, x_S)$ ，然后可以将其相应位置的 *shape* 相乘，返回的 *shape* 为 $(1 * y_1, \dots, x_R * y_R, x_S * y_S)$ 。
- 如果 *input.dim > d*，在 *dims* 的前面填充 1，直到它们的长度一致。例如将 *dims* 设置为 $(1, \dots, y_1, y_2, \dots, y_S)$ ，然后可以将其相应位置的 *shape* 相乘，返回的 *shape* 为 $(x_1 * 1, \dots, x_R * y_R, x_S * y_S)$ 。

参数:

- input** (Tensor) - 输入 *tensor*。
- dims** (tuple[int]) - 指定每维度的复制次数。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2], [3, 4]])
>>> mindspore.mint.tile(input, (2, 3))
Tensor(shape=[4, 6], dtype=Int64, value=
[[1, 2, 1, 2, 1, 2],
 [3, 4, 3, 4, 3, 4],
 [1, 2, 1, 2, 1, 2],
 [3, 4, 3, 4, 3, 4]])
>>> mindspore.mint.tile(input, (2, 3, 2))
Tensor(shape=[2, 6, 4], dtype=Int64, value=
[[[1, 2, 1, 2],
 [3, 4, 3, 4],
```

(下页继续)

(续上页)

```
[1, 2, 1, 2],
[3, 4, 3, 4],
[1, 2, 1, 2],
[3, 4, 3, 4]],
[[[1, 2, 1, 2],
[3, 4, 3, 4],
[1, 2, 1, 2],
[3, 4, 3, 4],
[1, 2, 1, 2],
[3, 4, 3, 4]]])
```

mindspore.mint.tril

`mindspore.mint.tril(input, diagonal=0)`

将指定对角线上方的元素设置为 0。

参数：

- **input** (Tensor) - 输入 tensor，其秩至少为 2。
- **diagonal** (int, 可选) - 二维 tensor 的指定对角线。默认 0，表示主对角线。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[ 1,  2,  3,  4],
...                             [ 5,  6,  7,  8],
...                             [10, 11, 12, 13],
...                             [14, 15, 16, 17]])
>>> mindspore.mint.tril(input)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  0,  0,  0],
 [ 5,  6,  0,  0],
 [10, 11, 12,  0],
 [14, 15, 16, 17]])
>>> mindspore.mint.tril(input, 1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  2,  0,  0],
 [ 5,  6,  7,  0],
 [10, 11, 12, 13],
 [14, 15, 16, 17]])
>>> mindspore.mint.tril(input, -1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 0,  0,  0,  0],
 [ 5,  0,  0,  0],
 [10, 11,  0,  0],
 [14, 15, 16,  0]])
>>> input = mindspore.tensor([[[ 1,  2,  3],
...                             [ 5,  6,  7],
...                             [10, 11, 12]]],
```

(下页继续)

(续上页)

```

...             [[ 1,  2,  3],
...             [ 5,  6,  7],
...             [10, 11, 12]])
>>> mindspore.mint.tril(input)
Tensor(shape=[2, 3, 3], dtype=Int64, value=
[[[ 1,  0,  0],
  [ 5,  6,  0],
  [10, 11, 12]],
 [[ 1,  0,  0],
  [ 5,  6,  0],
  [10, 11, 12]])

```

mindspore.mint.select

`mindspore.mint.select(input, dim, index)`

在给定索引处沿选定维度对输入 Tensor 进行切片。

警告： 这是一个实验性 API，可能会更改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (int) - 进行切片的维度。
- **index** (int) - 要选择的索引。

返回：

Tensor。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台：

Ascend

样例：

```

>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([[2, 3, 4, 5], [3, 2, 4, 5]])
>>> y = mint.select(input, 0, 0)
>>> print(y)
[2 3 4 5]

```

mindspore.mint.squeeze

`mindspore.mint.squeeze(input, dim)`

返回删除指定 *dim* 中大小为 1 的维度后的 Tensor。

如果 *dim* = (), 则删除所有大小为 1 的维度。如果指定了 *dim*, 删除指定 *dim* 中大小为 1 的维度。例如, 如果不指定维度, 即 *dim* = (), 输入的 shape 为 (A, 1, B, C, 1, D), 则输出的 Tensor 的 shape 为 (A, B, C, D); 如果指定维度, 那么 *squeeze* 操作仅在指定维度中进行。如果输入的 shape 为 (A, 1, B), *axis* = 0 或 *axis* = 2 时不会改变输入的 Tensor, 但 *dim* = 1 时, 会使输入 Tensor 的 shape 变为 (A, B)。

说明:

- 请注意, 在动态图模式下, 输出 Tensor 将与输入 Tensor 共享数据, 并且没有 Tensor 数据复制过程。
- 维度索引从 0 开始, 并且必须在 $[-input.ndim, input.ndim)$ 范围内。
- 在 GE 模式下, 只支持对 input shape 中大小为 1 的维度进行压缩。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 用于计算 Squeeze 的输入 Tensor。其 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **dim** (Union[int, tuple(int)]) - 指定待删除 shape 的维度索引。其会删除给定 *dim* 参数中所有大小为 1 的维度。如果指定了维度索引, 其数据类型必须为 int32 或 int64。

返回:

Tensor, shape 为 $(x_1, x_2, \dots, x_S)$ 。

异常:

- **TypeError** - *input* 不是 tensor。
- **TypeError** - *dim* 不是 int、tuple。
- **TypeError** - *dim* 是 tuple, 但其元素不全是 int。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.ones(shape=[3, 2, 1]), mindspore.float32)
>>> output = mint.squeeze(input, 2)
>>> print(output)
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

mindspore.mint.stack

`mindspore.mint.stack(tensors, dim=0)`

在指定轴上对输入 Tensor 序列进行堆叠。

输入秩为 R 的 Tensor 序列, 则输出秩为 $(R+1)$ 的 Tensor。

给定输入 Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。若输入 Tensor 的长度为 N 。如果存在 $dim \geq 0$, 则输出 Tensor 的 shape 为 $(x_1, x_2, \dots, x_{dim}, N, x_{dim+1}, \dots, x_R)$ 。

参数:

- **tensors** (Union[tuple, list]) - 输入多个 Tensor 对象组成的 tuple 或 list, 每个 Tensor 具有相同 shape 和数据类型。
- **dim** (int, 可选) - 指定堆叠运算的轴。取值范围为 $[-(R+1), R+1)$ 。默认值: 0。

返回:

堆叠运算后的 Tensor, 数据类型和 *tensors* 的相同。

异常:

- **TypeError** - *tensors* 中元素的数据类型不相同。
- **ValueError** - *dim* 不在 $[-(R+1), R+1)$ 范围中, 或 *tensors* 中元素的 shape 不相同。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> data1 = Tensor(np.array([0, 1]).astype(np.float32))
>>> data2 = Tensor(np.array([2, 3]).astype(np.float32))
>>> output = mint.stack([data1, data2], 0)
>>> print(output)
[[0. 1.]
 [2. 3.]]
```

mindspore.mint.swapaxes

`mindspore.mint.swapaxes(input, axis0, axis1)`

`mindspore.mint.transpose()` 的别名。其中参数 *input* 对应参考接口的 *input*, 参数 *axis0* 和 *axis1* 分别对应参考接口的 *dim0* 和 *dim1*。

更多参考详见 `mindspore.mint.transpose()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import mint
>>> from mindspore import Tensor
>>> input = Tensor(np.ones((2,3,4), dtype=np.float32))
>>> output = mint.swapaxes(input, 0, 2)
>>> print(output.shape)
(4, 3, 2)
```

mindspore.mint.transpose

mindspore.mint.**transpose**(*input*, *dim0*, *dim1*)

交换 Tensor 的两个维度。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim0** (int) - 第一个维度。
- **dim1** (int) - 第二个维度。

返回：

转化后的 Tensor，与输入具有相同的数据类型。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *dim0* 或者 *dim1* 不是整数。
- **ValueError** - 如果 *dim0* 或者 *dim1* 不在 $[-ndim, ndim - 1]$ 范围内。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import mint
>>> from mindspore import Tensor
>>> input = Tensor(np.ones((2,3,4), dtype=np.float32))
>>> output = mint.transpose(input, 0, 2)
>>> print(output.shape)
(4, 3, 2)
```

mindspore.mint.triu

`mindspore.mint.triu(input, diagonal=0)`

将指定对角线下方的元素设置为 0。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 tensor。
- **diagonal** (int, 可选) - 二维 tensor 的指定对角线。默认 0，表示主对角线。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[ 1,  2,  3,  4],
...                             [ 5,  6,  7,  8],
...                             [10, 11, 12, 13],
...                             [14, 15, 16, 17]])
>>> mindspore.mint.triu(input)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  2,  3,  4],
 [ 0,  6,  7,  8],
 [ 0,  0, 12, 13],
 [ 0,  0,  0, 17]])
>>> mindspore.mint.triu(input, 1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 0,  2,  3,  4],
 [ 0,  0,  7,  8],
 [ 0,  0,  0, 13],
 [ 0,  0,  0,  0]])
>>> mindspore.mint.triu(input, -1)
Tensor(shape=[4, 4], dtype=Int64, value=
[[ 1,  2,  3,  4],
 [ 5,  6,  7,  8],
 [ 0, 11, 12, 13],
 [ 0,  0, 16, 17]])
>>> input = mindspore.tensor([[[ 1,  2,  3],
...                             [ 5,  6,  7],
...                             [10, 11, 12]],
...                             [[ 1,  2,  3],
...                             [ 5,  6,  7],
...                             [10, 11, 12]])
>>> mindspore.mint.triu(input)
Tensor(shape=[2, 3, 3], dtype=Int64, value=
[[[ 1,  2,  3],
   [ 0,  6,  7],
   [ 0,  0, 12]],
```

(下页继续)

(续上页)

```
[[ 1,  2,  3],
 [ 0,  6,  7],
 [ 0,  0, 12]]])
```

mindspore.mint.unbind

`mindspore.mint.unbind(input, dim=0)`

根据指定轴对 Tensor 进行分解。给定一个 shape 为 $(n_1, n_2, \dots, n_R)$ 的 Tensor，在指定 dim 上对其分解，结果返回多个 shape 为 $(n_1, n_2, \dots, n_{dim}, n_{dim+2}, \dots, n_R)$ 的 Tensor。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 用来分解的输入 Tensor。其 shape 为 $(n_1, n_2, \dots, n_R)$ 。其 rank 必须大于 0。
- **dim** (int, 可选) - 用以分解的维度。取值范围为 $[-R, R)$ 。默认值为 0。

返回：

Tensor 组成的 tuple。每个 Tensor 对象的 shape 相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *dim* 不是 int 类型。
- **ValueError** - 如果 *dim* 超出 $[-R, R)$ 范围。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]))
>>> output = mint.unbind(input, dim=0)
>>> print(output)
(Tensor(shape=[3], dtype=Int64, value=[1, 2, 3]), Tensor(shape=[3], dtype=Int64, value=[4, 5,
→ 6]),
Tensor(shape=[3], dtype=Int64, value=[7, 8, 9]))
```

mindspore.mint.unique_consecutive

`mindspore.mint.unique_consecutive(input, return_inverse=False, return_counts=False, dim=None)`

对输入 Tensor 中连续且重复的元素去重。

在 `return_inverse=True` 时, 返回一个 Tensor, 包含输入 Tensor 中的元素在输出 Tensor 中的索引。

在 `return_counts=True` 时, 返回一个 Tensor, 表示输出元素在输入中的个数。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 Tensor。
- **return_inverse** (bool, 可选) - 是否返回每个输入中元素映射到输出中位置的索引。默认值: `False`。
- **return_counts** (bool, 可选) - 是否返回每个去重元素在输入所在的连续序列的计数。默认值: `False`。
- **dim** (int, 可选) - 维度。如果为 `None`, 则对输入进行展平操作。如果指定维度, 则必须是 `int32` 或 `int64` 类型。默认值: `None`。

返回:

Tensor 或包含 Tensor 对象的元组 (`output`、`inverse_indices`、`counts`)。

- **output** (Tensor), `output` 为去重后的输出, 与 `input` 具有相同的数据类型。
- **inverse_indices** (Tensor, 可选), 如果 `return_inverse` 为 `True`, 则返回 Tensor `inverse_indices`。若 Tensor `inverse_indices` 的 `shape` 与 `input` 相同, 则表示每个输入中元素映射到输出中位置的索引。
- **counts** (Tensor, 可选), 如果 `return_counts` 为 `True`, 则返回 Tensor `counts`。若 Tensor `counts` 的 `shape` 与 `output` 相同, 或当给定 `dim` 值时为 `output.shape[dim]`, 则表示每个去重元素在输入中所在的连续序列的计数。

异常:

- **TypeError** - `input` 不是 Tensor。
- **TypeError** - `input` 的数据类型不支持。
- **TypeError** - `return_inverse` 不是 bool。
- **TypeError** - `return_counts` 不是 bool。
- **TypeError** - `dim` 不是 int。
- **ValueError** - `dim` 不在 `[-ndim, ndim - 1]` 范围内。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([1, 1, 2, 2, 3, 1, 1, 2]), mstype.int64)
>>> output, inverse_indices, counts = mint.unique_consecutive(x, True, True, None)
>>> print(output)
[1 2 3 1 2]
>>> print(inverse_indices)
```

(下页继续)

(续上页)

```
[0 0 1 1 2 3 3 4]
>>> print(counts)
[2 2 1 2 1]
```

mindspore.mint.unsqueeze

`mindspore.mint.unsqueeze(input, dim)`

在输入 `tensor` 的指定维度上增加一维。

参数：

- **input** (Tensor) - 输入 `tensor`。
- **dim** (int) - 指定维度。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, 3, 4])
>>> mindspore.mint.unsqueeze(input, 0)
Tensor(shape=[1, 4], dtype=Int64, value=
[[1, 2, 3, 4]])
>>> mindspore.mint.unsqueeze(input, 1)
Tensor(shape=[4, 1], dtype=Int64, value=
[[1],
 [2],
 [3],
 [4]])
```

mindspore.mint.where

`mindspore.mint.where(condition, input, other)`

返回一个 Tensor，Tensor 的元素从 `input` 或 `other` 中根据 `condition` 选择。

$$output_i = \begin{cases} input_i, & \text{if } condition_i \\ other_i, & \text{otherwise} \end{cases}$$

参数：

- **condition** (Tensor[bool]) - 如果是 `True`，选取 `input` 中的元素，否则选取 `other` 中的元素。
- **input** (Union[Tensor, Scalar]) - 在 `condition` 为 `True` 的索引处选择的值。
- **other** (Union[Tensor, Scalar]) - 当 `condition` 为 `False` 的索引处选择的值。

返回：

Tensor，其中的元素从 `input` 和 `other` 中选取。

异常：

- **TypeError** - 如果 *condition* 不是 Tensor。
- **TypeError** - 如果 *input* 和 *other* 都是常量。
- **ValueError** - *condition*、*input* 和 *other* 不能互相广播。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import tensor, mint
>>> from mindspore import dtype as mstype
>>> a = tensor(np.arange(4).reshape((2, 2)), mstype.float32)
>>> b = tensor(np.ones((2, 2)), mstype.float32)
>>> condition = a < 3
>>> output = mint.where(condition, a, b)
>>> print(output)
[[0. 1.]
 [2. 1.]]
```

`mindspore.mint.where(condition)`

与接口 `mindspore.ops.nonzero()` 在输入为 *condition* 且 *as_tuple* 为 True 时一致。

支持平台：

Ascend

4.2 随机采样

接口名	概述	支持平台
<code>mindspore.mint.multinomial</code>	根据输入生成一个多项式分布的 Tensor。	Ascend
<code>mindspore.mint.normal</code>	根据正态（高斯）随机数分布生成随机数。	Ascend
<code>mindspore.mint.rand_like</code>	返回 <i>shape</i> 与输入相同，类型为 <i>dtype</i> 的 Tensor， <i>dtype</i> 由输入决定，其元素取值服从 [0, 1) 区间内的均匀分布。	Ascend
<code>mindspore.mint.rand</code>	返回一个 Tensor， <i>shape</i> 和 <i>dtype</i> 由输入决定，其元素为服从均匀分布的 [0, 1) 区间的数字。	Ascend

4.2.1 mindspore.mint.multinomial

`mindspore.mint.multinomial(input, num_samples, replacement=False, *, generator=None)`

根据输入生成一个多项式分布的 Tensor。

多项式分布是一种概率分布，把二项分布公式推广至多种状态，就得到了多项式分布。在多项式分布中，每个事件都有一个固定的概率，这些概率的和为 1。

`mindspore.mint.multinomial()` 接口的作用是对输入 *input* 进行 *num_samples* 次抽样，输出的 Tensor 则为每一次抽样时输入 Tensor 的索引，其中 *input* 中的值为每次抽样取到对应索引的概率。

这里我们给一个相对极端的用例方便理解，我们给定一个输入概率值 Tensor，值为 `Tensor([90 / 100, 10 / 100, 0], mindspore.float32)`，代表我们一共可以对三个索引进行抽样，分别为索引 0，索引 1，索引 2，它们被抽中的概率分别为 90%，10%，0%，我们对其进行 *n* 次抽样，抽样的结果序列则为多项式分布的计算结果，计算结果长度与抽样次

数一致。

在样例代码 case 1 中, 我们对其进行两次不放回抽样 (*replacement* 为 `False`), 那我们的计算结果则大概率为 `[0, 1]`, 小概率为 `[1, 0]`, 由于每次抽样抽到索引 0 的概率为 90%, 因此抽到的结果序列中, 第一次大概率是抽到索引 0, 由于抽到索引 2 的概率为 0, 因此抽样两次结果不可能出现索引 2, 那第二次结果一定是索引 1, 因此结果序列为 `[0, 1]`。

在样例代码 case 2 中, 我们对其进行 10 次放回抽样 (*replacement* 为 `True`), 可以看到计算结果中大概有 90% 的抽样结果为抽到索引 0, 符合预期。

在样例代码 case 3 中, 我们将输入扩展为 2 维, 可以看到抽样结果在每一个维度中结果也符合我们抽样预期。

说明: 输入的行不要求和为 1 (当使用值作为权重的情况下), 但必须是非负的、有限的, 并且和不能为 0。在使用值作为权重的情况下, 可以理解为对输入沿最后一维进行了归一化操作, 以此保证概率和为 1。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入的概率值 Tensor, 必须是一维或二维, 数据类型为 float32。
- **num_samples** (int) - 采样的次数。
- **replacement** (bool, 可选) - 是否是可放回的采样, 默认值: `False`。

关键字参数:

- **generator** (generator, 可选) - MindSpore 随机种子。默认值: `None`。

返回:

Tensor, 数据类型为 int64。如果 *input* 是向量, 输出数据 shape 是 *num_samples* 的向量。如果 *input* 是 m 行矩阵, 输出数据 shape 是 m * *num_samples* 大小的矩阵。

异常:

- **TypeError** - 如果 *input* 不是数据类型不是 float16、float32、float64 或 bfloat16 的 Tensor。
- **TypeError** - 如果 *num_samples* 不是一个 int, 或元素为 int 的是 Scalar, 或 shape 为 (1,) 仅有一个 int 元素的 Tensor。
- **RuntimeError** - *num_samples* <= 0。
- **RuntimeError** - *replacement* 为 `False` 时, *num_samples* > *input* 最后一维的 shape。
- **RuntimeError** - *input* 最后一维的 shape 超过 2^{24} 。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> # case 1: The output is random, and the length of the output is the same as num_sample.
>>> # replacement is False.
>>> input1 = Tensor([90 / 100, 10 / 100, 0], mindspore.float32)
>>> input2 = Tensor([90, 10, 0], mindspore.float32)
```

(下页继续)

(续上页)

```

>>> # input1 and input2 have the same meaning.
>>> output1 = mint.multinomial(input1, 2)
>>> output2 = mint.multinomial(input2, 2)
>>> # print(output1)
>>> # [0 1]
>>> # print(output2)
>>> # [0 1]
>>> print(len(output1))
2
>>> print(len(output2))
2
>>> # case 2: The output is random, and the length of the output is the same as num_sample.
>>> # replacement is True.
>>> output3 = mint.multinomial(input1, 10, replacement=True)
>>> # print(output3)
>>> # [0 0 1 0 0 0 0 0 0 0]
>>> print(len(output3))
10
>>> # case 3: The output is random, and the length of the output is the same as num_sample.
>>> # replacement is True.
>>> # rank is 2
>>> input4 = Tensor([[90, 10, 0], [10, 90, 0]], mstype.float32)
>>> output4 = mint.multinomial(input4, 10, replacement=True)
>>> # print(output4)
>>> # [[0 0 0 0 0 0 0 0 1 0]
>>> #   [1 1 1 1 1 0 1 1 1 1]]

```

4.2.2 mindspore.mint.normal

`mindspore.mint.normal(mean, std, *, generator=None)`

根据正态（高斯）随机数分布生成随机数。

参数：

- **mean** (Union[float, Tensor]) - 每个元素的均值，shape 与 *std* 相同。
- **std** (Union[float, Tensor]) - 每个元素的标准差，shape 与 *mean* 相同。*std* 的值大于等于 0。

关键字参数：

- **generator** (generator, 可选) - MindSpore 随机种子。默认值：None。

返回：

Tensor，输出 tensor 的 shape 和 *mean* 的 shape 相同。

异常：

- **TypeError** - 如果 *mean* 或 *std* 不是 Union[float, Tensor]。

支持平台：

Ascend

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import mint

```

(下页继续)

(续上页)

```
>>> from mindspore import Tensor
>>> mean = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> std = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> output = mint.normal(mean, std)
>>> print(output.shape)
(3,)
```

`mindspore.mint.normal(mean, std=1.0)`

与上面接口类似。但所有生成的元素共享同一个标准差。

参数：

- **mean** (Tensor) - 每个元素的均值。
- **std** (float, 可选) - 每个元素的标准差。*std* 的值大于等于 0。默认值：1.0。

返回：

Tensor, 输出 tensor 的 shape 和 *mean* 的 shape 相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import mint
>>> from mindspore import Tensor
>>> mean = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> output = mint.normal(mean, 1.0)
>>> print(output.shape)
(3,)
```

`mindspore.mint.normal(mean, std, size)`

与上面接口类似。但所有生成的元素共享同一个均值和标准差。输出 Tensor 的大小由 *size* 指定。

参数：

- **mean** (float) - 每个元素的均值。
- **std** (float) - 每个元素的标准差。
- **size** (tuple) - 输出 shape。

返回：

Tensor, shape 为 *size*。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import mint
>>> from mindspore import Tensor
>>> output = mint.normal(1.0, 2.0, (2, 4))
```

(下页继续)

(续上页)

```
>>> print(output.shape)
(2, 4)
```

4.2.3 mindspore.mint.rand_like

`mindspore.mint.rand_like(input, *, dtype=None)`

返回 `shape` 与输入相同，类型为 `dtype` 的 `Tensor`，`dtype` 由输入决定，其元素取值服从 $[0, 1)$ 区间内的均匀分布。

参数：

- **input** (`Tensor`) - 输入的 `Tensor`，用来决定输出 `Tensor` 的 `shape` 和默认的 `dtype`。

关键字参数：

- **dtype** (`mindspore.dtype`，可选) - 指定的输出 `Tensor` 的 `dtype`，必须是 `float` 类型。如果是 `None`，`input` 的 `dtype` 会被使用。默认值：`None`。

返回：

`Tensor`，`shape` 和 `dtype` 由输入决定其元素为服从均匀分布的 $[0, 1)$ 区间的数字。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import Tensor, mint
>>> a = Tensor([[2, 3, 4], [1, 2, 3]])
>>> print(mint.rand_like(a, dtype=ms.float32).shape)
(2, 3)
```

4.2.4 mindspore.mint.rand

`mindspore.mint.rand(*size, generator=None, dtype=None)`

返回一个 `Tensor`，`shape` 和 `dtype` 由输入决定，其元素为服从均匀分布的 $[0, 1)$ 区间的数字。

参数：

- **size** (`Union[int, tuple(int), list(int)]`) - 输出的 `Tensor` 的 `shape`，例如， $(2, 3)$ 或 `2`。

关键字参数：

- **generator** (`mindspore.Generator`，可选) - 伪随机数生成器。默认值：`None`，使用默认伪随机数生成器。
- **dtype** (`mindspore.dtype`，可选) - 指定的输出 `Tensor` 的 `dtype`。如果是 `None`，`mindspore.float32` 会被使用。默认值：`None`。

返回：

`Tensor`，`shape` 和 `dtype` 由输入决定其元素为服从均匀分布的 $[0, 1)$ 区间的数字。

异常：

- **ValueError** - 如果 `size` 包含负数。

支持平台：

Ascend

样例:

```
>>> from mindspore import mint
>>> print(mint.rand(2, 3).shape)
(2, 3)
```

4.3 数学运算

4.3.1 逐元素运算

接口名	概述	支持平台
<code>mindspore.mint.abs</code>	逐元素计算输入 tensor 的绝对值。	Ascend
<code>mindspore.mint.add</code>	对 <i>other</i> 缩放 <i>scalar</i> 后与 <i>input</i> 相加。	Ascend
<code>mindspore.mint.addmv</code>	<i>mat</i> 和 <i>vec</i> 矩阵向量相乘，且将输入向量 <i>input</i> 加到最终结果中。	Ascend
<code>mindspore.mint.acos</code>	逐元素计算输入 Tensor 的反余弦。	Ascend
<code>mindspore.mint.acosh</code>	逐元素计算输入 Tensor 的反双曲余弦。	Ascend
<code>mindspore.mint.arccos</code>	<code>mindspore.mint.acos()</code> 的别名。	Ascend
<code>mindspore.mint.arccosh</code>	<code>mindspore.mint.acosh()</code> 的别名。	Ascend
<code>mindspore.mint.arcsin</code>	<code>mindspore.mint.asin()</code> 的别名。	Ascend
<code>mindspore.mint.arcsinh</code>	<code>mindspore.mint.asinh()</code> 的别名。	Ascend
<code>mindspore.mint.arctan</code>	<code>mindspore.mint.atan()</code> 的别名。	Ascend
<code>mindspore.mint.arctan2</code>	<code>mindspore.mint.atan2()</code> 的别名。	Ascend
<code>mindspore.mint.arctanh</code>	<code>mindspore.mint.atanh()</code> 的别名。	Ascend
<code>mindspore.mint.asin</code>	逐元素计算输入 Tensor 的反正弦。	Ascend
<code>mindspore.mint.asinh</code>	计算输入元素的反双曲正弦。	Ascend
<code>mindspore.mint.atan</code>	逐元素计算输入 Tensor 的反正切值。	Ascend
<code>mindspore.mint.atan2</code>	逐元素计算 <i>input/other</i> 的反正切值。	Ascend
<code>mindspore.mint.atanh</code>	逐元素计算输入 tensor 的反双曲正切值。	Ascend
<code>mindspore.mint.bitwise_and</code>	逐元素执行两个 Tensor 的与运算。	Ascend
<code>mindspore.mint.bitwise_or</code>	逐元素执行两个 Tensor 的或运算。	Ascend
<code>mindspore.mint.bitwise_xor</code>	逐元素执行两个 Tensor 的异或运算。	Ascend
<code>mindspore.mint.ceil</code>	逐元素对输入 tensor 做向上取整。	Ascend
<code>mindspore.mint.clamp</code>	将输入 Tensor 的值裁剪到指定的最小值和最大值之间。	Ascend
<code>mindspore.mint.cos</code>	逐元素计算输入 tensor 的余弦。	Ascend
<code>mindspore.mint.cosh</code>	逐元素计算输入 tensor 的双曲余弦。	Ascend
<code>mindspore.mint.cross</code>	按指定维度计算两个输入 tensor 的叉积。	Ascend
<code>mindspore.mint.diff</code>	计算沿给定维度的第 <i>n</i> 个正向差。	Ascend
<code>mindspore.mint.div</code>	逐元素计算 <i>input</i> 除以 <i>other</i> 的商。	Ascend
<code>mindspore.mint.divide</code>	<code>mindspore.mint.div()</code> 的别名。	Ascend
<code>mindspore.mint.erf</code>	逐元素计算输入 tensor 的高斯误差。	Ascend
<code>mindspore.mint.erfc</code>	逐元素计算输入 tensor 的互补误差。	Ascend
<code>mindspore.mint.erfinv</code>	逐元素计算输入 tensor 的逆误差。	Ascend
<code>mindspore.mint.exp</code>	逐元素计算输入 tensor 的指数。	Ascend
<code>mindspore.mint.exp2</code>	逐元素计算 Tensor <i>input</i> 以 2 为底的指数。	Ascend
<code>mindspore.mint.expm1</code>	逐元素计算输入 tensor 的指数，然后减 1。	Ascend
<code>mindspore.mint.fix</code>	<code>mindspore.mint.trunc()</code> 的别名。	Ascend

下页继续

表 4 - 续上页

<code>mindspore.mint.float_power</code>	使用双精度计算 <i>input</i> 中每个元素的 <i>exponent</i> 次幂, 并返回一个 <code>float64</code> 类型的 <code>Tensor</code> 。	Ascend
<code>mindspore.mint.floor</code>	逐元素对输入 <code>tensor</code> 做向下取整。	Ascend
<code>mindspore.mint.fmod</code>	计算除法运算 <i>input/other</i> 的浮点余数。	Ascend
<code>mindspore.mint.frac</code>	计算 <i>input</i> 中每个元素的小数部分。	Ascend
<code>mindspore.mint.lerp</code>	基于权重参数计算两个 <code>Tensor</code> 之间的线性插值。	Ascend
<code>mindspore.mint.log</code>	逐元素计算输入 <code>tensor</code> 的自然对数。	Ascend
<code>mindspore.mint.log1p</code>	逐元素计算 (<code>tensor + 1</code>) 的自然对数。	Ascend
<code>mindspore.mint.log2</code>	逐元素返回 <code>Tensor</code> 以 2 为底的对数。	Ascend
<code>mindspore.mint.log10</code>	逐元素返回 <code>Tensor</code> 以 10 为底的对数。	Ascend
<code>mindspore.mint.logaddexp</code>	计算输入的指数和对数的和。	Ascend
<code>mindspore.mint.logaddexp2</code>	计算以 2 为底的输入的指数和对数的和。	Ascend
<code>mindspore.mint.logical_and</code>	逐元素计算两个 <code>tensor</code> 的逻辑与运算。	Ascend
<code>mindspore.mint.logical_not</code>	逐元素计算输入 <code>tensor</code> 的逻辑非运算。	Ascend
<code>mindspore.mint.logical_or</code>	逐元素计算两个 <code>tensor</code> 的逻辑或运算。	Ascend
<code>mindspore.mint.logical_xor</code>	逐元素计算两个 <code>tensor</code> 的逻辑异或运算。	Ascend
<code>mindspore.mint.mul</code>	将 <i>other</i> 与 <i>input</i> 相乘。	Ascend
<code>mindspore.mint.mv</code>	实现矩阵 <i>input</i> 和向量 <i>vec</i> 相乘。	Ascend
<code>mindspore.mint.nansum</code>	计算 <i>input</i> 指定维度元素的和, 将非数字 (NaNs) 处理为零。	Ascend
<code>mindspore.mint.nan_to_num</code>	将 <i>input</i> 中的 <code>NaN</code> 、正无穷大和负无穷大值分别替换为 <i>nan</i> 、 <i>posinf</i> 和 <i>neginf</i> 指定的值。	Ascend
<code>mindspore.mint.neg</code>	计算输入 <i>input</i> 的相反数并返回。	Ascend
<code>mindspore.mint.negative</code>	<code>mindspore.mint.neg()</code> 的别名。	Ascend
<code>mindspore.mint.pow</code>	计算 <i>input</i> 中每个元素的 <i>exponent</i> 次幂。	Ascend
<code>mindspore.mint.polar</code>	将极坐标转化为笛卡尔坐标。	Ascend
<code>mindspore.mint.ravel</code>	沿着 0 轴方向, 将多维 <code>Tensor</code> 展开成一维。	Ascend
<code>mindspore.mint.reciprocal</code>	返回输入 <code>tensor</code> 每个元素的倒数。	Ascend
<code>mindspore.mint.remainder</code>	逐元素计算 <i>input</i> 除以 <i>other</i> 后的余数。	Ascend
<code>mindspore.mint.roll</code>	按维度移动 <code>tensor</code> 的元素。	Ascend
<code>mindspore.mint.round</code>	对输入数据四舍五入到最接近的整数。	Ascend
<code>mindspore.mint.rsqrt</code>	逐元素计算输入 <code>tensor</code> 的平方根倒数。	Ascend
<code>mindspore.mint.sigmoid</code>	逐元素计算 Sigmoid 激活函数。	Ascend
<code>mindspore.mint.sign</code>	按 <code>sign</code> 公式逐元素计算输入 <code>tensor</code> 。	Ascend
<code>mindspore.mint.sin</code>	逐元素计算输入 <code>tensor</code> 的正弦。	Ascend
<code>mindspore.mint.sinc</code>	计算输入的归一化正弦值。	Ascend
<code>mindspore.mint.sinh</code>	逐元素计算输入 <code>tensor</code> 的双曲正弦。	Ascend
<code>mindspore.mint.softmax</code>	<code>mindspore.mint.nn.functional.softmax()</code> 的别名。	Ascend
<code>mindspore.mint.sqrt</code>	逐元素返回当前 <code>Tensor</code> 的平方根。	Ascend
<code>mindspore.mint.square</code>	逐元素计算 <code>tensor</code> 的平方。	Ascend
<code>mindspore.mint.sub</code>	对 <i>other</i> 缩放 <i>scalar</i> 后与 <i>input</i> 相减。	Ascend
<code>mindspore.mint.t</code>	转置输入 <code>Tensor</code> 。	Ascend
<code>mindspore.mint.tan</code>	逐元素计算输入 <code>tensor</code> 的正切值。	Ascend
<code>mindspore.mint.tanh</code>	逐元素计算输入元素的双曲正切。	Ascend
<code>mindspore.mint.trunc</code>	返回一个 <code>tensor</code> , 包含输入元素的截断整数值。	Ascend
<code>mindspore.mint.xlogy</code>	计算第一个输入乘以第二个输入的对数。	Ascend

mindspore.mint.abs

`mindspore.mint.abs(input)`

逐元素计算输入 *tensor* 的绝对值。

$$out_i = |input_i|$$

参数:

- **input** (Tensor) - 输入 *tensor*。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> output = mindspore.mint.abs(mindspore.tensor([-1.0, 1.0, 0.0]))
>>> print(output)
[1. 1. 0.]
```

mindspore.mint.add

`mindspore.mint.add(input, other, *, alpha=1)`

对 *other* 缩放 *scalar* 后与 *input* 相加。

$$out_i = input_i + alpha \times other_i$$

说明:

- 当两个输入 *shape* 不同时, 它们必须能够广播到一个共同的 *shape*。
- 两个输入和 *alpha* 遵守隐式类型转换规则以使数据类型保持一致。

参数:

- **input** (Union[Tensor, number.Number, bool]) - 第一个输入是一个 `number.Number`、一个 `bool` 或一个数据类型为 `number` 或 `bool_` 的 Tensor。
- **other** (Union[Tensor, number.Number, bool]) - 第二个输入, 是一个 `number.Number`、一个 `bool` 或一个数据类型为 `number` 或 `bool_` 的 Tensor。

关键字参数:

- **alpha** (`number.Number`, 可选) - 应用于 *other* 的缩放因子, 默认值为 1。

返回:

Tensor, 其 *shape* 与输入 *input*、*other* 广播后的 *shape* 相同, 数据类型是两个输入和 *alpha* 中精度更高或位数更多的类型。

异常:

- **TypeError** - 如果 *input*、*other*、*alpha* 不是以下之一: `Tensor`、`number.Number`、`bool`。

- **TypeError** - 如果 *alpha* 是 float 类型, 但是 *input*、*other* 不是 float 类型。
- **TypeError** - 如果 *alpha* 是 bool 类型, 但是 *input*、*other* 不是 bool 类型。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> x = Tensor(1, mindspore.int32)
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> alpha = 0.5
>>> output = mint.add(x, y, alpha=alpha) # x.add(y, alpha=alpha)
>>> print(output)
[3. 3.5 4.]
>>> # the data type of x is int32, the data type of y is float32,
>>> # alpha is a float, and the output is the data format of higher precision float32.
>>> print(output.dtype)
Float32
```

mindspore.mint.addmv

`mindspore.mint.addmv(input, mat, vec, *, beta=1, alpha=1)`

mat 和 *vec* 矩阵向量相乘, 且将输入向量 *input* 加到最终结果中。

如果 *mat* 是一个大小为 (N, M) Tensor, *vec* 是一个大小为 M 的一维 Tensor, 那么 *input* 必须可广播到一个大小为 N 的一维 Tensor。这种情况下 *output* 是一个大小为 N 的一维 Tensor。

$$output = \beta input + \alpha(mat @ vec)$$

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 被加的向量。
- **mat** (Tensor) - 第一个需要相乘的 Tensor。
- **vec** (Tensor) - 第二个需要相乘的 Tensor。

关键字参数:

- **beta** (Union[float, int], 可选) - 输入的系数。默认值: 1。
- **alpha** (Union[float, int], 可选) - *mat@vec* 的系数。默认值: 1。

返回:

Tensor, shape 大小为 $(N,)$, 其数据类型与 *input* 相同。

异常:

- **TypeError** - *input*、*mat* 或 *vec* 不是 Tensor。
- **TypeError** - *mat* 和 *vec* 数据类型不一致。

- **ValueError** - *mat* 不是二维 Tensor。
- **ValueError** - *vec* 不是一维 Tensor。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([2., 3.]).astype(np.float32))
>>> mat = Tensor(np.array([[2., 5., 3.], [4., 2., 2.]]).astype(np.float32))
>>> vec = Tensor(np.array([3., 2., 4.]).astype(np.float32))
>>> output = mint.addmv(input, mat, vec)
>>> print(output)
[30. 27.]
```

mindspore.mint.acos

`mindspore.mint.acos(input)`

逐元素计算输入 Tensor 的反余弦。

$$out_i = \cos^{-1}(input_i)$$

参数：

- **input** (Tensor) - Tensor 的 shape 为 $(N,*)$ ，其中 $*$ 表示任意数量的附加维度。

返回：

Tensor, shape 与 *input* 相同。当输入数据类型为 bool、int8、uint8、int16、int32、int64 时，返回值数据类型为 float32。否则，返回值数据类型与输入数据类型相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0.74, 0.04, 0.30, 0.56]), mindspore.float32)
>>> output = mint.acos(input)
>>> print(output)
[0.7377037 1.5307857 1.2661037 0.9764114]
```

mindspore.mint.acosh

`mindspore.mint.acosh(input)`

逐元素计算输入 Tensor 的反双曲余弦。

$$out_i = \cosh^{-1}(input_i)$$

说明：给定一个输入 Tensor *input*，该函数计算每个元素的反双曲余弦。输入范围为 $[1, \infty]$ 。

参数：

- **input** (Tensor) - 需要计算反双曲余弦函数的输入 Tensor。

返回：

Tensor, shape 与 *input* 相同。当输入数据类型为 bool、int8、uint8、int16、int32、int64 时，返回值数据类型为 float32。否则，返回值数据类型与输入数据类型相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1.0, 1.5, 3.0, 100.0]), mindspore.float32)
>>> output = mint.acosh(input)
>>> print(output)
[0.          0.9624236 1.7627472 5.298292 ]
```

mindspore.mint.arccos

`mindspore.mint.arccos(input)`

`mindspore.mint.acos()` 的别名。

支持平台：

Ascend

mindspore.mint.arccosh

`mindspore.mint.arccosh(input)`

`mindspore.mint.acosh()` 的别名。

支持平台：

Ascend

mindspore.mint.arcsin

`mindspore.mint.arcsin(x)`
`mindspore.mint.asin()` 的别名。

支持平台:

Ascend

mindspore.mint.arcsinh

`mindspore.mint.arcsinh(input)`
`mindspore.mint.asinh()` 的别名。

支持平台:

Ascend

mindspore.mint.arctan

`mindspore.mint.arctan(input)`
`mindspore.mint.atan()` 的别名。

支持平台:

Ascend

mindspore.mint.arctan2

`mindspore.mint.arctan2(input, other)`
`mindspore.mint.atan2()` 的别名。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> x = Tensor(np.array([0, 1]), mindspore.float32)
>>> y = Tensor(np.array([1, 1]), mindspore.float32)
>>> output = mint.arctan2(x, y)
>>> print(output)
[0.          0.7853982]
```

mindspore.mint.arctanh

`mindspore.mint.arctanh(input)`
`mindspore.mint.atanh()` 的别名。

支持平台:

Ascend

mindspore.mint.asin

`mindspore.mint.asin(input)`
逐元素计算输入 Tensor 的反正弦。

$$out_i = \sin^{-1}(input_i)$$

参数:

- **input** (Tensor) - Tensor 的 shape 为 $(N,*)$ ，其中 $*$ 表示任意数量的附加维度。

返回:

Tensor, shape 与 *input* 相同。当输入数据类型为 bool、int8、uint8、int16、int32、int64 时，返回值数据类型为 float32。
否则，返回值数据类型与输入数据类型相同。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0.74, 0.04, 0.30, 0.56]), mindspore.float32)
>>> output = mint.asin(input)
>>> print(output)
[0.8330927  0.04001068  0.30469266  0.59438497 ]
```

mindspore.mint.asinh

`mindspore.mint.asinh(input)`
计算输入元素的反双曲正弦。

$$out_i = \sinh^{-1}(input_i)$$

参数:

- **input** (Tensor) - 需要计算反双曲正弦函数的输入。

返回:

Tensor, shape 与 *input* 相同。当输入数据类型为 bool、int8、uint8、int16、int32、int64 时，返回值数据类型为 float32。
否则，返回值数据类型与输入数据类型相同。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([-5.0, 1.5, 3.0, 100.0]), mindspore.float32)
>>> output = mint.asinh(input)
>>> print(output)
[-2.3124385  1.1947632  1.8184465  5.298342 ]
```

mindspore.mint.atan

`mindspore.mint.atan(input)`

逐元素计算输入 Tensor 的反正切值。

$$out_i = \tan^{-1}(input_i)$$

参数：

- **input** (Tensor) - Tensor 的 shape 为 $(N,*)$ ，其中 $*$ 表示任意数量的附加维度。

返回：

Tensor, shape 与 *input* 相同。当输入数据类型为 bool、int8、uint8、int16、int32、int64 时，返回值数据类型为 float32。否则，返回值数据类型与输入数据类型相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1.0, 0.0]), mindspore.float32)
>>> output = mint.atan(input)
>>> print(output)
[0.7853982 0. ]
```

mindspore.mint.atan2

`mindspore.mint.atan2(input, other)`

逐元素计算 `input/other` 的反正切值。

返回 $\theta \in [-\pi, \pi]$, 使得 $input = r * \sin(\theta), other = r * \cos(\theta)$, 其中 $r = \sqrt{input^2 + other^2}$ 。

说明:

- 参数 `input` 和 `other` 遵循隐式类型转换规则, 使数据类型保持一致。如果两参数数据类型不一致, 则低精度类型会被转换成较高精度类型。

参数:

- **input** (Tensor, Number.number) - 输入 Tensor 或常数。
- **other** (Tensor, Number.number) - 输入 Tensor 或常数, shape 与 `input` 相同, 或能与 `input` 的 shape 广播。

返回:

Tensor, 与广播后的输入 shape 相同。当输入类型为 bool、int8、uint8、int16、int32、int64 时, 返回值类型为 float32。否则, 返回值类型与输入类型相同。

异常:

- **TypeError** - `input` 或 `other` 不是 Tensor 或常数。
- **RuntimeError** - `input` 与 `other` 之间的数据类型转换不被支持。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0, 1]), mindspore.float32)
>>> other = Tensor(np.array([1, 1]), mindspore.float32)
>>> output = mint.atan2(input, other)
>>> print(output)
[0.          0.7853982]
```

mindspore.mint.atanh

`mindspore.mint.atanh(input)`

逐元素计算输入 tensor 的反双曲正切值。

$$out_i = \tanh^{-1}(input_i)$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> output = mindspore.mint.atanh(mindspore.tensor([0, -0.5]))
>>> print(output)
[ 0.          -0.54930615]
```

mindspore.mint.bitwise_and

`mindspore.mint.bitwise_and(input, other)`

逐元素执行两个 Tensor 的与运算。

说明: 参数 *input* 和 *other* 遵循隐式类型转换规则, 使数据类型保持一致。如果两参数数据类型不一致, 则低精度类型会被转换成较高精度类型。

$$out_i = input_i \mid other_i$$

参数:

- **input** (Tensor) - 输入 Tensor。
- **other** (Tensor, Number.number) - 输入 Tensor 或常量, shape 与 *input* 相同, 或能与 *input* 的 shape 广播。

返回:

Tensor, 与广播后的输入 shape 相同, 和 *input* 数据类型相同。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> other = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> output = mint.bitwise_and(input, other)
>>> print(output)
[ 0  0  1 -1  1  0  1]
```

`mindspore.mint.bitwise_or`

`mindspore.mint.bitwise_or(input, other)`

逐元素执行两个 Tensor 的或运算。

$$out_i = input_i \wedge other_i$$

说明：参数 *input* 和 *other* 遵循隐式类型转换规则，使数据类型保持一致。如果两参数数据类型不一致，则低精度类型会被转换成较高精度类型。

参数：

- **input** (Tensor) - 输入 Tensor。
- **other** (Tensor, Number.number) - 输入 Tensor 或常量，shape 与 *input* 相同，或能与 *input* 的 shape 广播。

返回：

Tensor，与广播后的输入 shape 相同，和 *input* 数据类型相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> other = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> output = mint.bitwise_or(input, other)
>>> print(output)
[ 0  1  1 -1 -1  3  3]
```

`mindspore.mint.bitwise_xor`

`mindspore.mint.bitwise_xor(input, other)`

逐元素执行两个 Tensor 的异或运算。

$$out_i = input_i \oplus other_i$$

说明：参数 *input* 和 *other* 遵循隐式类型转换规则，使数据类型保持一致。如果两参数数据类型不一致，则低精度类型会被转换成较高精度类型。

参数：

- **input** (Tensor) - 输入 Tensor。
- **other** (Tensor, Number.number) - 输入 Tensor 或常量，shape 与 *input* 相同，或能与 *input* 的 shape 广播。

返回：

Tensor，与广播后的输入 shape 相同，和 *input* 数据类型相同。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> other = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> output = mint.bitwise_xor(input, other)
>>> print(output)
[ 0  1  0  0 -2  3  2]
```

mindspore.mint.ceil

`mindspore.mint.ceil(input)`

逐元素对输入 tensor 做向上取整。

$$out_i = \lceil input_i \rceil$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1.1, 2.5, -1.5])
>>> output = mindspore.mint.ceil(input)
>>> print(output)
[ 2.  3. -1.]
```

mindspore.mint.clamp

`mindspore.mint.clamp(input, min=None, max=None)`

将输入 Tensor 的值裁剪到指定的最小值和最大值之间。

限制 *input* 的范围, 其最小值为 *min*, 最大值为 *max*。

$$out_i = \begin{cases} max & \text{if } input_i \geq max \\ input_i & \text{if } min < input_i < max \\ min & \text{if } input_i \leq min \end{cases}$$

说明:

- *min* 和 *max* 不能同时为 None;

- 当 *min* 为 *None*, *max* 不为 *None* 时, *Tensor* 中大于 *max* 的元素会变为 *max*;
- 当 *min* 不为 *None*, *max* 为 *None* 时, *Tensor* 中小于 *min* 的元素会变为 *min*;
- 当 *min* 大于 *max* 时, *Tensor* 中所有元素的值会被置为 *max*;
- *input*, *min* 和 *max* 的数据类型需支持隐式类型转换, 且不能为 *bool* 类型。

参数:

- **input** (*Tensor*) - *clamp* 的输入, 类型为 *Tensor*。支持任意维度的 *Tensor*。
- **min** (*Union(Tensor, float, int)*, 可选) - 指定最小值。默认值为 *None*。
- **max** (*Union(Tensor, float, int)*, 可选) - 指定最大值。默认值为 *None*。

返回:

Tensor, 表示裁剪后的 *Tensor*。其 *shape* 和数据类型和 *input* 相同。

异常:

- **ValueError** - 如果 *min* 和 *max* 都为 *None*。
- **TypeError** - 如果 *input* 的数据类型不是 *Tensor*。
- **TypeError** - 如果 *min* 的数据类型不为 *None*、*Tensor*、*float* 或 *int*。
- **TypeError** - 如果 *max* 的数据类型不为 *None*、*Tensor*、*float* 或 *int*。

支持平台:

Ascend

样例:

```
>>> # case 1: the data type of input is Tensor
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> min_value = Tensor(5, mindspore.float32)
>>> max_value = Tensor(20, mindspore.float32)
>>> input = Tensor(np.array([[1., 25., 5., 7.], [4., 11., 6., 21.]]), mindspore.float32)
>>> output = mint.clamp(input, min_value, max_value)
>>> print(output)
[[ 5. 20.  5.  7.]
 [ 5. 11.  6. 20.]]
>>> # case 2: the data type of input is number
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> min_value = 5
>>> max_value = 20
>>> input = Tensor(np.array([[1., 25., 5., 7.], [4., 11., 6., 21.]]), mindspore.float32)
>>> output = mint.clamp(input, min_value, max_value)
>>> print(output)
[[ 5. 20.  5.  7.]
 [ 5. 11.  6. 20.]]
```

mindspore.mint.cos

`mindspore.mint.cos(input)`

逐元素计算输入 `tensor` 的余弦。

$$out_i = \cos(x_i)$$

警告：如果使用 `float64`，可能会存在精度丢失的问题。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> # The datatype of output will be float32 when datatype of input is in [bool, int8, uint8,
↳ int16, int32, int64] (Datatype only supported on Ascend).
>>> input = mindspore.tensor([0, 1, 2], mindspore.int32)
>>> mindspore.mint.cos(input)
Tensor(shape=[3], dtype=Float32, value= [ 1.00000000e+00,  5.40302306e-01, -4.16146837e-01])
>>>
>>> # Otherwise output has the same dtype as the `input`.
>>> input = mindspore.tensor([0.74, 0.04, 0.30, 0.56], mindspore.float64)
>>> mindspore.mint.cos(input)
Tensor(shape=[4], dtype=Float64, value= [ 7.38468559e-01,  9.99200107e-01,  9.55336489e-01,
↳ 8.47255111e-01])
```

mindspore.mint.cosh

`mindspore.mint.cosh(input)`

逐元素计算输入 `tensor` 的双曲余弦。

$$out_i = \cosh(input_i)$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([0.74, 0.04, 0.30, 0.56])
>>> output = mindspore.mint.cosh(input)
>>> print(output)
[1.2865248 1.0008001 1.0453385 1.1609408]
```

mindspore.mint.cross

`mindspore.mint.cross(input, other, dim=None)`

按指定维度计算两个输入 `tensor` 的叉积。

说明: `input` 和 `other` 必须有相同的 `shape`, 且指定的 `dim` 上 `size` 必须为 3。如果不指定 `dim`, 则使用第一个 `size` 为 3 的维度。

参数:

- **input** (Tensor) - 第一个输入 `tensor`。
- **other** (Tensor) - 第二个输入 `tensor`。
- **dim** (int, 可选) - 指定维度。默认 `None`。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3], [1, 2, 3], [1, 2, 3]])
>>> other = mindspore.tensor([[4, 5, 6], [4, 5, 6], [4, 5, 6]])
>>> mindspore.mint.cross(input, other)
Tensor(shape=[3, 3], dtype=Int64, value=
[[0, 0, 0],
 [0, 0, 0],
 [0, 0, 0]])
>>> mindspore.mint.cross(input, other, dim=1)
Tensor(shape=[3, 3], dtype=Int64, value=
[[-3, 6, -3],
 [-3, 6, -3],
 [-3, 6, -3]])
```


mindspore.mint.diff

`mindspore.mint.diff(input, n=1, dim=-1, prepend=None, append=None)`

计算沿给定维度的第 n 个正向差。

第一个正向差通过 $out[i] = input[i + 1] - input[i]$ 计算得到，对于第 n 个正相差，可以通过递归使用 *diff* 得到。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 用于计算正向差的张量。
- **n** (int, 可选) - 递归计算正向差的次数。默认值为 1。
- **dim** (int, 可选) - 指定用于计算正相差的维度，默认为最后一个维度。默认值为 -1。
- **prepend** (Tensor, 可选) - 在计算正向差前，在 *input* 的 *dim* 维度上添加相应的值。*prepend* 的维度需要与 *input* 保持一致，并且除了指定 *dim*，其余维度上，*prepend* 的大小需要与 *input* 保持一致。默认值为 None。
- **append** (Tensor, 可选) - 在计算正向差前，在 *input* 的 *dim* 维度上追加相应的值。*append* 的维度需要与 *input* 保持一致，并且除了指定 *dim*，其余维度上，*append* 的大小需要与 *input* 保持一致。默认值为 None。

返回：

- Tensor。计算得到的第 n 个正向差。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *n* 不是标量或标量 Tensor。
- **TypeError** - 如果 *dim* 不是标量或标量 Tensor。
- **TypeError** - 如果 *input* 的类型为 complex64、complex128、float64、int16。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor, mint
>>> x = Tensor([1, 3, -1, 0, 4])
>>> out = mint.diff(x)
>>> print(out.asnumpy())
[ 2 -4  1  4]
```

mindspore.mint.div

`mindspore.mint.div(input, other, *, rounding_mode=None)`

逐元素计算 *input* 除以 *other* 的商。

$$out_i = input_i / other_i$$

说明：

- 当两个输入具有不同的 `shape` 时，它们的 `shape` 必须要能广播为一个共同的 `shape`。
- 两个输入不能同时为 `bool` 类型。[`True`, `Tensor(True, bool_)`, `Tensor(np.array([True]), bool_)`] 等都为 `bool` 类型。
- 两个输入遵循隐式类型转换规则，使数据类型保持一致。

参数：

- **input** (`Union[Tensor, Number, bool]`) - 被除数。
- **other** (`Union[Tensor, Number, bool]`) - 除数。

关键字参数：

- **rounding_mode** (`str`, 可选) - 应用于结果的舍入类型。三种类型被定义为 `None`、`"floor"` 和 `"trunc"`。默认值：`None`。
 - **None**：默认行为。相当于 Python 中的 *true division* 或 NumPy 中的 *true_divide*。
 - **"floor"**：将除法的结果向下舍入。相当于 Python 中的 *floor division* 或 NumPy 中的 *floor_divide*。
 - **"trunc"**：将除法的结果舍入到零。相当于 C 语言风格的整数除法。

返回：

`Tensor`，输出的 `shape` 与广播后的 `shape` 相同，数据类型取两个输入中精度较高或数字较高的。

异常：

- **TypeError** - 如果 `input` 和 `other` 不是以下之一：`Tensor`、`Number`、`bool`。
- **ValueError** - 如果 `rounding_mode` 不是以下之一：`None`、`"floor"`、`"trunc"`。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> y = Tensor(np.array([4.0, 5.0, 6.0]), mindspore.float32)
>>> output = mint.div(x, y)
>>> print(output)
[0.25 0.4 0.5]
```

`mindspore.mint.divide`

`mindspore.mint.divide` (`input, other, *, rounding_mode=None`)
`mindspore.mint.div()` 的别名。

支持平台：

Ascend

mindspore.mint.erf

`mindspore.mint.erf(input)`

逐元素计算输入 tensor 的高斯误差。

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # The datatype of output will be float32 when datatype of input is in [int64,
↳bool](Datatype only supported on Ascend).
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.int64)
>>> mindspore.mint.erf(input)
Tensor(shape=[5], dtype=Float32, value= [-8.42700793e-01,  0.00000000e+00,  8.42700793e-01,  9.95322265e-01,  9.9977910e-01])
>>>
>>> # Otherwise output has the same dtype as the input.
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.float64)
>>> mindspore.mint.erf(input)
Tensor(shape=[5], dtype=Float64, value= [-8.42700793e-01,  0.00000000e+00,  8.42700793e-01,  9.95322265e-01,  9.9977910e-01])
```

mindspore.mint.erfc

`mindspore.mint.erfc(input)`

逐元素计算输入 tensor 的互补误差。

$$\text{erfc}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # The datatype of output will be float32 when datatype of input is in [int64,
↳bool](Datatype only supported on Ascend).
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.int64)
>>> mindspore.mint.erfc(input)
Tensor(shape=[5], dtype=Float32, value= [ 1.84270079e+00,  1.00000000e+00,  1.57299207e-01,
↳4.67773498e-03,  2.20904970e-05])
>>>
>>> # Otherwise output has the same dtype as the input.
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.float64)
>>> mindspore.mint.erfc(input)
Tensor(shape=[5], dtype=Float64, value= [ 1.84270079e+00,  1.00000000e+00,  1.57299207e-01,
↳4.67773498e-03,  2.20904970e-05])
```

mindspore.mint.erfinv

`mindspore.mint.erfinv(input)`

逐元素计算输入 `tensor` 的逆误差。逆误差函数在 $(-1, 1)$ 范围内定义为：

$$\text{erfinv}(\text{erf}(x)) = x$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> # When the `input` is int8, int16, int32, int64, uint8 or bool, the return value type is
↳float32.
>>> input = mindspore.tensor([0, 0.5, -0.9], mindspore.int64)
>>> mindspore.mint.erfinv(input)
Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00])
>>> # Otherwise, the return value type is the same as the input type.
>>> input = mindspore.tensor([0, 0.5, -0.9], mindspore.float32)
>>> mindspore.mint.erfinv(input)
Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  4.76936132e-01, -1.16308689e+00])
```

mindspore.mint.exp

`mindspore.mint.exp(input)`

逐元素计算输入 `tensor` 的指数。

$$\text{out}_i = e^{x_i}$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([0.0, 1.0, 3.0], mindspore.float32)
>>> output = mindspore.mint.exp(input)
>>> print(output)
[ 1.          2.7182817 20.085537]
```

mindspore.mint.exp2

`mindspore.mint.exp2(input)`

逐元素计算 Tensor *input* 以 2 为底的指数。

$$out_i = 2^{input_i}$$

参数：

- **input** (Tensor) - 输入 Tensor。

返回：

Tensor, shape 与 *input* 相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([0.0, 1.0, 2.0, 4.0]), mindspore.float32)
>>> output = mint.exp2(x)
>>> print(output)
[ 1.  2.  4. 16.]
```

mindspore.mint.expm1

`mindspore.mint.expm1(input)`

逐元素计算输入 tensor 的指数，然后减 1。

$$out_i = e^{x_i} - 1$$

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([0.0, 1.0, 3.0], mindspore.float32)
>>> output = mindspore.mint.expml(input)
>>> print(output)
[ 0.          1.7182817 19.085537 ]
```

mindspore.mint.fix

`mindspore.mint.fix(input)`

`mindspore.mint.trunc()` 的别名。

更多参考详见 `mindspore.mint.trunc()`。

支持平台：

Ascend

mindspore.mint.float_power

`mindspore.mint.float_power(input, exponent)`

使用双精度计算 `input` 中每个元素的 `exponent` 次幂，并返回一个 `float64` 类型的 Tensor。

$$out_i = input_i^{exponent_i}$$

警告：这是一个实验性 API，后续可能修改或删除。

说明：和 `ops.pow` 不同，该函数总是使用双精度进行计算，而 `ops.pow` 的计算精度则取决于类型提升结果。目前该函数不支持复数计算。由于在 Ascend 设备上 `float64` 类型计算性能比不上其他数据类型，因此建议该函数有且仅有在追求双精度且对性能要求不高的场景使用，否则 `ops.pow` 会是更好的选项。

参数：

- **input** (Union[Tensor, Number]) - 第一个输入，为 Tensor 或数值型数据类型。
- **exponent** (Union[Tensor, Number]) - 第二个输入，如果第一个输入是 Tensor，第二个输入可以是 Number 或 Tensor。否则，必须是 Tensor。

返回：

Tensor，输出的 shape 与广播后的 shape 相同，返回的数据类型为 `mindspore.float64`。

异常：

- **TypeError** - `input` 和 `exponent` 都不是 Tensor。

- **TypeError** - *input* 或 *exponent* 既不是 Tensor 也不是数值，或者其数据中包含了复数。
- **ValueError** - *input* 和 *exponent* 具有不同的 shape 并且无法互相广播。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor, mint
>>> input = Tensor([1, 2, 3])
>>> mint.float_power(input, 2)
Tensor(shape=[3], dtype=Float64, value= [ 1.00000000e+00,  4.00000000e+00,  9.00000000e+00])
>>>
>>> exp = Tensor([2, -3, -4])
>>> mint.float_power(input, exp)
Tensor(shape=[3], dtype=Float64, value= [ 1.00000000e+00,  1.25000000e-01,  1.23456790e-02])
```

mindspore.mint.floor

`mindspore.mint.floor(input)`

逐元素对输入 tensor 做向下取整。

$$out_i = \lfloor input_i \rfloor$$

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1.1, 2.5, -1.5])
>>> output = mindspore.mint.floor(input)
>>> print(output)
[ 1.  2. -2.]
```

mindspore.mint.fmod

`mindspore.mint.fmod(input, other)`

计算除法运算 *input/other* 的浮点余数。

$$out = input - n * other$$

其中 *n* 是 *input/other* 结果中的整数部分。返回值的符号和 *input* 相同，在数值上小于 *other*。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 被除数。
- **other** (Union[Tensor, Number]) - 除数。

返回：

Tensor，输出的 shape 与广播后的 shape 相同，数据类型取两个输入中精度较高或数字较高的。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([-4., -3.5, 0, 3.5, 4]), mindspore.float32)
>>> output = mint.fmod(input, 2.5)
>>> print(output)
[-1.5 -1.    0.    1.    1.5]
```

mindspore.mint.frac

`mindspore.mint.frac(input)`

计算 *input* 中每个元素的小数部分。

$$out_i = input_i - \lfloor |input_i| \rfloor * sgn(input_i)$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。

返回：

Tensor，其类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：


```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor([2, 4.2, -2.5], mindspore.float16)
>>> output = mint.frac(x)
>>> print(output)
[ 0.          0.1992 -0.5    ]
```

mindspore.mint.lerp

`mindspore.mint.lerp(input, end, weight)`

基于权重参数计算两个 Tensor 之间的线性插值。

如果权重参数 `weight` 是一个 Tensor, 则 `input` 与 `end` 与 `weight` 广播后应有相同 shape。如果权重参数 `weight` 是一个浮点数, 则 `input` 与 `end` 广播后 shape 应相同。如果权重参数 `weight` 是一个浮点数并且平台为 Ascend, 则 `input` 与 `end` 应为 float32。

警告: 这是一个实验性 API, 后续可能修改或删除。

$$output_i = input_i + weight_i * (end_i - input_i)$$

参数:

- **input** (Tensor) - 进行线性插值的 Tensor 开始点, 其数据类型必须为 float16 或者 float32。
- **end** (Tensor) - 进行线性插值的 Tensor 结束点, 其数据类型必须与 `input` 一致。
- **weight** (Union[float, Tensor]) - 线性插值公式的权重参数。为 Scalar 时, 其数据类型为 float; 为 Tensor 时, 其数据类型为 float16 或者 float32。

返回:

Tensor, 其数据类型和维度必须和输入中的 `input` 保持一致。

异常:

- **TypeError** - 如果 `input` 或者 `end` 不是 Tensor。
- **TypeError** - 如果 `weight` 不是 float 类型 Scalar 或者 Tensor。
- **TypeError** - 如果 `input` 或者 `end` 的数据类型不是 float16 或者 float32。
- **TypeError** - 如果 `weight` 为 Tensor 且 `weight` 不是 float16 或者 float32。
- **TypeError** - 如果 `input` 和 `end` 的数据类型不一致。
- **TypeError** - 如果 `weight` 为 Tensor 且 `input`、`end` 和 `weight` 数据类型不一致。
- **ValueError** - 如果 `input` 与 `end` 的 shape 无法广播至一致。
- **ValueError** - 如果 `weight` 为 Tensor 且 `weight` 与 `input` 的 shape 无法广播至一致。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> start = Tensor(np.array([1., 2., 3., 4.]), mindspore.float32)
>>> end = Tensor(np.array([10., 10., 10., 10.]), mindspore.float32)
>>> output = mint.lerp(start, end, 0.5)
>>> print(output)
[5.5 6. 6.5 7. ]
```

mindspore.mint.log

`mindspore.mint.log(input)`

逐元素计算输入 `tensor` 的自然对数。

$$y_i = \log_e(x_i)$$

警告：如果输入值在 (0, 0.01] 或 [0.95, 1.05] 范围内，则输出精度可能会存在误差。

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.mint.log(x)
>>> print(output)
[0.          0.6931472  1.3862944]
```

mindspore.mint.log1p

`mindspore.mint.log1p(input)`

逐元素计算 $(\text{tensor} + 1)$ 的自然对数。

$$out_i = \log_e(input_i + 1)$$

参数：

- **input** (Tensor) - 输入 `tensor`。

返回：

Tensor

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.mint.log1p(x)
>>> print(output)
[0.6931472 1.0986123 1.609438 ]
```

mindspore.mint.log2

`mindspore.mint.log2(input)`

逐元素返回 Tensor 以 2 为底的对数。

$$y_i = \log_2(x_i)$$

警告:

- 如果输入值在 (0, 0.01] 或 [0.95, 1.05] 范围内, 则输出精度可能会存在误差。

参数:

- **input** (Tensor) - 任意维度的输入 Tensor。其值必须大于 0。

返回:

Tensor, 具有与 *input* 相同的 shape。如果 *input.dtype* 为整数或布尔类型, 输出数据类型为 float32。否则输出数据类型与 *input.dtype* 相同。

异常:

- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([3.0, 5.0, 7.0]), mindspore.float32)
>>> output = mint.log2(x)
>>> print(output)
[1.5849625 2.321928 2.807355 ]
```

mindspore.mint.log10

`mindspore.mint.log10(input)`

逐元素返回 Tensor 以 10 为底的对数。

$$y_i = \log_{10}(x_i)$$

警告:

- 这是一个实验性 API，后续可能修改或删除。
- 如果输入值在 (0, 0.01] 或 [0.95, 1.05] 范围内，则输出精度可能会存在误差。

参数:

- **input** (Tensor) - 任意维度的输入 Tensor。其值必须大于 0。

返回:

Tensor，具有与 *input* 相同的 shape，dtype 根据 *input.dtype* 变化。

- 如果 *input.dtype* 为 [float16、float32、float64、bfloat16]，输出数据类型 dtype 与输入 *input.dtype* 相同。
- 如果 *input.dtype* 为整数或布尔类型，输出数据类型 dtype 为 float32。

异常:

- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([3.0, 5.0, 7.0]), mindspore.float32)
>>> output = mint.log10(x)
>>> print(output)
[0.47712136 0.69897    0.845098 ]
```

mindspore.mint.logaddexp

`mindspore.mint.logaddexp(input, other)`

计算输入的指数和的对数。该函数可以很好地解决统计学中的一个问题，即计算得到的事件概率可能非常小，以至于超出正常浮点数的范围。

$$out_i = \log(\exp(input_i) + \exp(other_i))$$

警告: 这是一个实验性 API，后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 Tensor。其数据类型必须是 float。
- **other** (Tensor) - 输入 Tensor。其数据类型必须是 float。如果 *input* 的 shape 不等于 *other* 的 shape，它们必须被广播成相同 shape。

返回：

Tensor，数据类型与输入 *input* 和 *other* 相同。

异常：

- **TypeError** - *input* 或 *other* 不是 Tensor。
- **TypeError** - *input* 或 *other* 的数据类型不是 float。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x1 = Tensor(np.array([1, 2, 3]).astype(np.float16))
>>> x2 = Tensor(np.array(2).astype(np.float16))
>>> output = mint.logaddexp(x1, x2)
>>> print(output)
[2.312 2.693 3.312]
```

mindspore.mint.logaddexp2

`mindspore.mint.logaddexp2(input, other)`

计算以 2 为底的输入的指数和的对数。

$$out_i = \log_2(2^{input_i} + 2^{other_i})$$

参数：

- **input** (Tensor) - 输入 Tensor，其数据类型必须是 float。
- **other** (Tensor) - 输入 Tensor，其数据类型必须是 float。如果 *input* 的 shape 不等于 *other* 的 shape，它们必须被广播成相同 shape(输出的形状)。

返回：

Tensor，数据类型与输入 *input* 和 *other* 相同。

异常：

- **TypeError** - *input* 或 *other* 不是 Tensor。
- **TypeError** - *input* 或 *other* 的数据类型不是 float。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x1 = Tensor(np.array([1, 2, 3]).astype(np.float16))
>>> x2 = Tensor(np.array(2).astype(np.float16))
>>> output = mint.logaddexp2(x1, x2)
>>> print(output)
[2.586 3. 3.586]
```

mindspore.mint.logical_and

mindspore.mint.logical_and(input, other)

逐元素计算两个 tensor 的逻辑与运算。

$$out_i = input_i \wedge other_i$$

说明:

- 支持广播。
- 支持隐式类型转换。

参数:

- **input** (Union[Tensor, bool]) - 第一个输入。
- **other** (Union[Tensor, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> y = mindspore.tensor([True, True, False], mindspore.bool_)
>>> output = mindspore.mint.logical_and(x, y)
>>> print(output)
[ True False False]
>>> x = mindspore.tensor(1, mindspore.bool_)
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.mint.logical_and(x, y)
>>> print(output)
False
>>> x = True
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.mint.logical_and(x, y)
>>> print(output)
False
>>> x = True
>>> y = mindspore.tensor([True, False], mindspore.bool_)
```

(下页继续)

(续上页)

```
>>> output = mindspore.mint.logical_and(x, y)
>>> print(output)
[True False]
```

mindspore.mint.logical_not

`mindspore.mint.logical_not(input)`
逐元素计算输入 `tensor` 的逻辑非运算。

$$out_i = \neg input_i$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> output = mindspore.mint.logical_not(x)
>>> print(output)
[False True False]
```

mindspore.mint.logical_or

`mindspore.mint.logical_or(input, other)`
逐元素计算两个 `tensor` 的逻辑或运算。

$$out_i = input_i \vee other_i$$

说明:

- 支持广播。
- 支持隐式类型转换。

参数:

- **input** (Union[Tensor, bool]) - 第一个输入。
- **other** (Union[Tensor, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> y = mindspore.tensor([True, True, False], mindspore.bool_)
>>> output = mindspore.mint.logical_or(x, y)
>>> print(output)
[ True True True]
>>> x = mindspore.tensor(1, mindspore.bool_)
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.mint.logical_or(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.mint.logical_or(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor([True, False], mindspore.bool_)
>>> output = mindspore.mint.logical_or(x, y)
>>> print(output)
[True True]
```

mindspore.mint.logical_xor

`mindspore.mint.logical_xor` (*input, other*)

逐元素计算两个 `tensor` 的逻辑异或运算。

$$out_i = input_i \oplus other_i$$

说明:

- 支持广播。
 - 支持隐式类型转换。
-

参数:

- **input** (Tensor) - 第一个输入 `tensor`。
- **other** (Tensor) - 第二个输入 `tensor`。

返回:

Tensor

支持平台:

Ascend

样例:


```

>>> import mindspore
>>> x = mindspore.tensor([True, False, True], mindspore.bool_)
>>> y = mindspore.tensor([True, True, False], mindspore.bool_)
>>> output = mindspore.mint.logical_xor(x, y)
>>> print(output)
[False True True]
>>> x = mindspore.tensor(1, mindspore.bool_)
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.mint.logical_xor(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor(0, mindspore.bool_)
>>> output = mindspore.mint.logical_xor(x, y)
>>> print(output)
True
>>> x = True
>>> y = mindspore.tensor([True, False], mindspore.bool_)
>>> output = mindspore.mint.logical_xor(x, y)
>>> print(output)
[False True]

```

mindspore.mint.mul

`mindspore.mint.mul(input, other)`

将 *other* 与 *input* 相乘。

$$out_i = input_i \times other_i$$

说明:

- 当两个输入 *shape* 不同时，它们必须能够广播到一个共同的 *shape*。
- 两个输入遵守隐式类型转换规则以使数据类型保持一致。

参数:

- **input** (`Union[Tensor, number.Number, bool]`) - 第一个输入。是一个 `number.Number`、一个 `bool` 或一个数据类型为 `number` 或 `bool_` 的 `Tensor`。
- **other** (`Union[Tensor, number.Number, bool]`) - 第二个输入。是一个 `number.Number`、一个 `bool` 或一个数据类型为 `number` 或 `bool_` 的 `Tensor`。

返回:

`Tensor`, *shape* 与广播后的 *shape* 相同，数据类型为两个输入中精度较高的类型。

异常:

- **TypeError** - 如果 *input*、*other* 不是以下之一: `Tensor`、`number.Number`、`bool`。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.array([2, 6, 9]).astype(np.int32))
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> output = mint.mul(x, y)
>>> print(output)
[8. 30. 54.]
>>> # the data type of x is int32, the data type of y is float32,
>>> # and the output is the data format of higher precision float32.
>>> print(output.dtype)
Float32
```

mindspore.mint.mv

`mindspore.mint.mv(input, vec)`

实现矩阵 *input* 和向量 *vec* 相乘。如果 *input* 是 shape 为 (N, M) 的 Tensor, *vec* 是 shape 为 $(M,)$ 的 Tensor, 则输出 shape 为 $(N,)$ 的一维 Tensor。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入矩阵。其 shape 为 (N, M) ，且 rank 必须为二维。
- **vec** (Tensor) - 输入向量。其 shape 为 $(M,)$ ，且 rank 必须为一维。

返回：

Tensor, shape 为 $(N,)$ 。

异常：

- **TypeError** - 如果 *input* 或者 *vec* 不是 Tensor。
- **TypeError** - 如果 *input* 或 *vec* 的 dtype 均不是 float16、float32。
- **TypeError** - 如果 *input* 和 *vec* 的 dtype 不同。
- **ValueError** - 如果 *input* 不是二维张量，或者 *vec* 不是一维张量。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[3., 4.], [1., 6.], [1., 3.]]).astype(np.float32))
>>> vec = Tensor(np.array([1., 2.]).astype(np.float32))
>>> output = mint.mv(input, vec)
>>> print(output)
[11. 13. 7.]
```

mindspore.mint.nansum

`mindspore.mint.nansum(input, dim=None, keepdim=False, *, dtype=None)`

计算 *input* 指定维度元素的和，将非数字 (NaNs) 处理为零。

警告:

- 仅支持 Atlas A2 训练系列产品。
- 这是一个实验性 API，后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 Tensor。
- **dim** (Union[int, tuple(int)], 可选) - 求和的维度。取值范围 $[-\text{rank}(\text{input}), \text{rank}(\text{input})]$ 。默认值: `None`，对 Tensor 中的所有元素求和。
- **keepdim** (bool, 可选) - 输出 Tensor 是否保持维度。默认值: `False`，不保留维度。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 输出 Tensor 的类型。默认值: `None`。

返回:

Tensor，输入 *input* 指定维度的元素和，将非数字 (NaNs) 处理为零。

- 如果 *dim* 为 `None`，且 *keepdim* 为 `False`，则输出一个零维 Tensor，表示输入 Tensor 中所有元素的和。
- 如果 *dim* 为 `int`，值为 2，并且 *keepdim* 为 `False`，则输出的 shape 为: $(\text{input}_1, \text{input}_3, \dots, \text{input}_R)$ 。
- 如果 *dim* 为 `tuple(int)` 或 `list(int)`，值为 (2, 3)，并且 *keepdim* 为 `False`，则输出的 shape 为 $(\text{input}_1, \text{input}_4, \dots, \text{input}_R)$ 。

异常:

- **TypeError** - *input* 不是一个 Tensor。
- **TypeError** - *keepdim* 不是 bool 类型。
- **TypeError** - *input* 的数据类型或 *dtype* 是 complex 类型。
- **ValueError** - *dim* 不在 $[-\text{rank}(\text{input}), \text{rank}(\text{input})]$ 。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[float("nan"), 2, 3], [1, 2, float("nan")]]), mindspore.float32)
>>> output1 = mint.nansum(x, dim=0, keepdim=False, dtype=mindspore.float32)
>>> output2 = mint.nansum(x, dim=0, keepdim=True, dtype=mindspore.float32)
>>> print(output1)
[1. 4. 3.]
>>> print(output2)
[[1. 4. 3.]]
```

mindspore.mint.nan_to_num

`mindspore.mint.nan_to_num(input, nan=None, posinf=None, neginf=None)`

将 `input` 中的 `NaN`、正无穷大和负无穷大值分别替换为 `nan`、`posinf` 和 `neginf` 指定的值。

警告：对于 Ascend，仅支持 Atlas A2 训练系列产品。这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 tensor。
- **nan** (number, 可选) - `NaN` 的替换值，默认 `None`。
- **posinf** (number, 可选) - `posinf` 的替换值。如果为 `None`，则为 `input` 类型支持的上限，默认 `None`。
- **neginf** (number, 可选) - `neginf` 的替换值。如果为 `None`，则为 `input` 类型支持的下限，默认 `None`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([float('nan'), float('inf'), -float('inf'), 5.0], mindspore.
↪float32)
>>> output = mindspore.mint.nan_to_num(input)
>>> print(output)
[ 0.0000000e+00  3.4028235e+38 -3.4028235e+38  5.0000000e+00]
>>> output = mindspore.mint.nan_to_num(input, 1.0)
>>> print(output)
[ 1.0000000e+00  3.4028235e+38 -3.4028235e+38  5.0000000e+00]
>>> output = mindspore.mint.nan_to_num(input, 1.0, 2.0)
>>> print(output)
[ 1.0000000e+00  2.0000000e+00 -3.4028235e+38  5.0000000e+00]
>>> output = mindspore.mint.nan_to_num(input, 1.0, 2.0, 3.0)
>>> print(output)
[1.  2.  3.  5.]
```

mindspore.mint.neg

`mindspore.mint.neg(input)`

计算输入 `input` 的相反数并返回。

$$out_i = -input_i$$

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1, 2, -1, 2, 0, -3.5], mindspore.float32)
>>> output = mindspore.mint.neg(input)
>>> print(output)
[-1.  -2.   1.  -2.   0.   3.5]
```

mindspore.mint.negative

`mindspore.mint.negative(input)`
`mindspore.mint.neg()` 的别名。

支持平台:

Ascend

mindspore.mint.pow

`mindspore.mint.pow(input, exponent)`
 计算 `input` 中每个元素的 `exponent` 次幂。

当 `exponent` 是 Tensor 时, `input` 和 `exponent` 的 shape 必须是可广播的。

$$out_i = input_i^{exponent_i}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Union[Tensor, Number]) - 第一个输入, 是一个 Number 值或数据类型为 `number` 或 `bool_` 的 Tensor。
- **exponent** (Union[Tensor, Number]) - 第二个输入, 是一个 Number 值或数据类型为 `number` 或 `bool_` 的 Tensor。

返回:

Tensor, shape 与广播后的 shape 相同, 数据类型为两个输入中精度较高的类型。

异常:

- **TypeError** - `input` 和 `exponent` 同时为 `bool` 类型。
- **TypeError** - `input` 是一个 Tensor 且为 `int` 或 `bool` 类型, 而 `exponent` 是一个 `Scalar` 且值为负整型时。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1.0, 2.0, 4.0]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> exponent = 3.0
>>> output = mint.pow(input, exponent)
>>> print(output)
[ 1.  8. 64.]
>>>
>>> input = Tensor(np.array([1.0, 2.0, 4.0]), mindspore.float32)
>>> exponent = Tensor(np.array([2.0, 4.0, 3.0]), mindspore.float32)
>>> output = mint.pow(input, exponent)
>>> print(output)
[ 1. 16. 64.]
```

mindspore.mint.polar

`mindspore.mint.polar(abs, angle)`

将极坐标转化为笛卡尔坐标。

返回一个复数 `tensor`，其元素是由输入极坐标构造的笛卡尔坐标。其中极坐标由极径 `abs` 和极角 `angle` 给定。

$$y_i = abs_i * \cos(angle_i) + abs_i * \sin(angle_i) * j$$

参数：

- **abs** (Tensor, float) - 极径。
- **angle** (Tensor, float) - 极角。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> abs = mindspore.tensor([1, 2], mindspore.float32)
>>> angle = mindspore.tensor([np.pi / 2, 5 * np.pi / 4], mindspore.float32)
>>> output = mindspore.mint.polar(abs, angle)
>>> print(output)
[ -4.3711388e-08+1.j          -1.4142137e+00-1.4142134j]
```

mindspore.mint.ravel

`mindspore.mint.ravel(input)`

沿着 0 轴方向，将多维 Tensor 展开成一维。

参数：

- **input** (Tensor) - 要展开的 Tensor。

返回：

一维 Tensor，含有与输入相同的元素。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[0, 1], [2, 1]]).astype(np.float32))
>>> output = mint.ravel(x)
>>> print(output)
[0. 1. 2. 1.]
>>> print(output.shape)
(4,)
```

mindspore.mint.reciprocal

`mindspore.mint.reciprocal(input)`

返回输入 **tensor** 每个元素的倒数。

$$out_i = \frac{1}{x_i}$$

参数：

- **input** (Tensor) - 输入 **tensor**。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.mint.reciprocal(input)
>>> print(output)
[1.    0.5  0.25]
```

mindspore.mint.remainder

`mindspore.mint.remainder(input, other)`

逐元素计算 *input* 除以 *other* 后的余数。结果与除数同号且绝对值小于除数的绝对值。

支持广播和隐式数据类型提升。

```
remainder(input, other) == input - input.div(other, rounding_mode="floor") * other
```

说明：输入不支持复数类型。至少一个输入为 **tensor**，且不能都为布尔型 **tensor**。

参数：

- **input** (Union[Tensor, numbers.Number, bool]) - 除数为数值型、bool 或数据类型为 `number` 或 `bool_` 的 Tensor。
- **other** (Union[Tensor, numbers.Number, bool]) - 被除数为数值型、bool 或数据类型为数值型或 bool 的 Tensor。当除数是 Tensor 时，则被除数是数值型、bool 或数据类型为数值型或 bool 的 Tensor。当除数是 Scalar 时，则被除数必须是数据类型为数值型或 bool 的 Tensor。

返回：

Tensor，经过隐式类型提升和广播。

异常：

- **TypeError** - 如果 *input* 和 *other* 不是以下类型之一：(tensor, tensor), (tensor, number), (tensor, bool), (number, tensor) 或 (bool, tensor)。
- **ValueError** - 如果 *input* 和 *other* 不能被广播。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([-4.0, 5.0, 6.0]).astype(np.float32))
>>> y = Tensor(np.array([3.0, 2.0, 3.0]).astype(np.float64))
>>> output = mint.remainder(x, y)
>>> print(output)
[2.  1.  0.]
```

mindspore.mint.roll

`mindspore.mint.roll(input, shifts, dims=None)`

按维度移动 tensor 的元素。

参数：

- **input** (Tensor) - 输入 tensor。
- **shifts** (Union[list(int), tuple(int), int]) - 元素移动量。
- **dims** (Union[list(int), tuple(int), int], 可选) - 指定移动维度，默认 None，表示将输入 tensor 展平后再进行计算，然后将计算结果 reshape 为输入的 shape。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([0, 1, 2, 3, 4], mindspore.float32)
>>> # case1: Parameter `shifts` is positive
>>> output = mindspore.mint.roll(input, shifts=2, dims=0)
>>> print(output)
[3.  4.  0.  1.  2.]
>>> # case2: Parameter `shifts` is negative
```

(下页继续)

(续上页)

```
>>> output = mindspore.mint.roll(input, shifts=-2, dims=0)
>>> print(output)
[2. 3. 4. 0. 1.]
```

mindspore.mint.round

`mindspore.mint.round(input, *, decimals=0)`

对输入数据四舍五入到最近的整数。

$$out_i \approx input_i$$

说明: Ascend 平台支持的输入数据类型包括 `bfloat16` (Atlas 训练系列产品不支持)、`float16`、`float32`、`float64`、`int32`、`int64`。

参数:

- **input** (Tensor) - 输入 tensor。

关键字参数:

- **decimals** (int, 可选) - 要舍入到的小数位数。如果为负数, 则指定小数点左侧的位数。默认 0。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> output = mindspore.mint.round(mindspore.tensor([4.7, -2.3, 9.1, -7.7]))
>>> print(output)
[ 5. -2.  9. -8.]
>>> # Values equidistant from two integers are rounded towards the
>>> # the nearest even value (zero is treated as even)
>>> output = mindspore.mint.round(mindspore.tensor([-0.5, 0.5, 1.5, 2.5]))
>>> print(output)
[0. 0.  2.  2.]
>>> # A positive decimals argument rounds to the to that decimal place
>>> output = mindspore.mint.round(mindspore.tensor([0.1234567]), decimals=3)
>>> print(output)
[0.123]
>>> # A negative decimals argument rounds to the left of the decimal
>>> output = mindspore.mint.round(mindspore.tensor([1200.1234567]), decimals=-3)
>>> print(output)
[1000.]
```

mindspore.mint.rsqrt

`mindspore.mint.rsqrt(input)`

逐元素计算输入 tensor 的平方根倒数。

$$out_i = \frac{1}{\sqrt{input_i}}$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([-0.0370, 0.2970, 1.5420, -0.9105])
>>> output = mindspore.mint.rsqrt(input)
>>> print(output)
[      nan 1.8349396 0.8053002      nan]
```

mindspore.mint.sigmoid

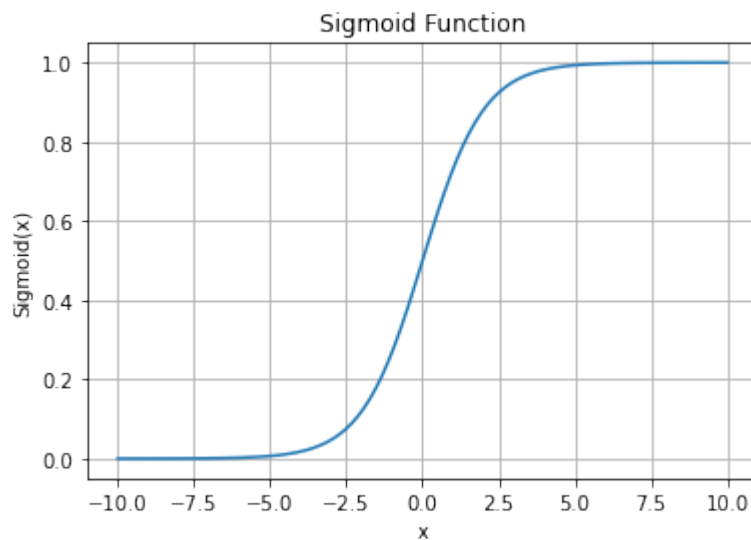
`mindspore.mint.sigmoid(input)`

逐元素计算 Sigmoid 激活函数。Sigmoid 函数定义为:

$$\text{sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)}$$

其中 x_i 是 x 的一个元素。

Sigmoid 函数图:



参数:

- **input** (Tensor) - *input* 即为上述公式中的 x 。任意维度的 Tensor, 数据类型为 float16、float32、float64、complex64 或 complex128。

返回:

Tensor, 数据类型和 shape 与 *input* 的相同。

异常:

- **TypeError** - *input* 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = mint.sigmoid(input)
>>> print(output)
[0.7310586  0.880797  0.95257413 0.98201376 0.9933072 ]
```

mindspore.mint.sign

mindspore.mint.**sign**(*input*)

按 sign 公式逐元素计算输入 tensor。

$$\text{out}_i = \begin{cases} -1 & \text{input}_i < 0 \\ 0 & \text{input}_i = 0 \\ 1 & \text{input}_i > 0 \end{cases}$$

说明: 在输入为 NaN 且数据类型为 float64 时, 该算子计算结果为 NaN。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[ -1,  0,  2,  4,  6], [ 2,  3,  5, -6,  0]])
>>> output = mindspore.mint.sign(input)
>>> print(output)
```

(下页继续)

(续上页)

```

[[-1  0  1  1  1]
 [ 1  1  1 -1  0]]
>>> mindspore.set_device(device_target="CPU")
>>> x = mindspore.tensor([[-1, 0, float('inf'), 4, float('nan')], [2, 3, float('-inf'), -6, 0]])
>>> output = mindspore.mint.sign(x)
>>> print(output)
[[-1.  0.  1.  1.  0.]
 [ 1.  1. -1. -1.  0.]]

```

mindspore.mint.sin

`mindspore.mint.sin(input)`

逐元素计算输入 `tensor` 的正弦。

$$output_i = \sin(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend

样例:

```

>>> import mindspore
>>> input = mindspore.tensor([0.62, 0.28, 0.43, 0.62], mindspore.float32)
>>> output = mindspore.mint.sin(input)
>>> print(output)
[0.58103514 0.27635565 0.4168708 0.58103514]

```

mindspore.mint.sinc

`mindspore.mint.sinc(input)`

计算输入的归一化正弦值。

$$out_i = \begin{cases} \frac{\sin(\pi input_i)}{\pi input_i} & input_i \neq 0 \\ 1 & input_i = 0 \end{cases}$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.62, 0.28, 0.43, 0.62], mindspore.float32)
>>> output = mindspore.mint.sinc(input)
>>> print(output)
[0.47735003 0.8759357 0.7224278 0.47735003]
```

mindspore.mint.sinh

`mindspore.mint.sinh(input)`

逐元素计算输入 `tensor` 的双曲正弦。

$$output_i = \sinh(input_i)$$

参数:

- **input** (Tensor) - 输入 `tensor`。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.62, 0.28, 0.43, 0.62], mindspore.float32)
>>> output = mindspore.mint.sinh(input)
>>> print(output)
[0.6604918 0.28367308 0.44337422 0.6604918 ]
```

mindspore.mint.softmax

`mindspore.mint.softmax(input, dim, *, dtype=None)`

`mindspore.mint.nn.functional.softmax()` 的别名。

支持平台:

Ascend

mindspore.mint.sqrt

`mindspore.mint.sqrt(input)`

逐元素返回当前 `Tensor` 的平方根。

$$out_i = \sqrt{input_i}$$

参数:

- **input** (Tensor) - 输入 `Tensor`，数据类型为 `number.Number`。

返回:

Tensor，具有与 `input` 相同的 `shape`。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1.0, 4.0, 9.0]), mindspore.float32)
>>> output = mint.sqrt(input)
>>> print(output)
[1. 2. 3.]
```

mindspore.mint.square

`mindspore.mint.square(input)`

逐元素计算 **tensor** 的平方。

$$y_i = input_i^2$$

参数:

- **input** (Tensor) - 输入 **tensor**。

返回:

Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> output = mindspore.mint.square(input)
>>> print(output)
[1. 4. 9.]
```

mindspore.mint.sub

`mindspore.mint.sub(input, other, *, alpha=1)`

对 *other* 缩放 *scalar* 后与 *input* 相减。

$$out_i = input_i - alpha \times other_i$$

说明:

- 当两个输入 **shape** 不同时, 它们必须能够广播到一个共同的 **shape**。
 - 两个输入和 *alpha* 遵守隐式类型转换规则以使数据类型保持一致。
-

参数：

- **input** (Union[Tensor, number.Number, bool]) - 第一个输入是一个 number.Number、一个 bool 或一个数据类型为 number 或 bool_ 的 Tensor。
- **other** (Union[Tensor, number.Number, bool]) - 第二个输入，是一个 number.Number、一个 bool 或一个数据类型为 number 或 bool_ 的 Tensor。

关键字参数：

- **alpha** (number.Number, 可选) - 应用于 *other* 的缩放因子，默认值为 1。

返回：

Tensor，其 shape 与输入 *input*、*other* 广播后的 shape 相同，数据类型是两个输入和 alpha 中精度更高或位数更多的类型。

异常：

- **TypeError** - 如果 *input*、*other* 不是以下之一：Tensor、number.Number、bool。
- **TypeError** - 如果 *alpha* 是 float 类型，但是 *input*、*other* 不是 float 类型。
- **TypeError** - 如果 *alpha* 是 bool 类型，但是 *input*、*other* 不是 bool 类型。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> y = Tensor(1, mindspore.int32)
>>> alpha = 0.5
>>> output = mint.sub(x, y, alpha=alpha)
>>> print(output)
[3.5 4.5 5.5]
>>> # the data type of x is float32, the data type of y is int32,
>>> # alpha is a float, and the output is the data format of higher precision float32.
>>> print(output.dtype)
Float32
```

mindspore.mint.t

mindspore.mint.t(*input*)

转置输入 Tensor。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。

返回：

Tensor，转置 2 维 Tensor，一维 Tensor 按原样返回。

异常:

- **ValueError** - 输入 Tensor 维度大于 2。
- **ValueError** - 输入空 Tensor。
- **TypeError** - 输入非 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[1, 2, 3], [4, 5, 6]]), mindspore.float32)
>>> output = mint.t(input)
>>> print(output)
[[ 1.  4.]
 [ 2.  5.]
 [ 3.  6.]]
```

mindspore.mint.tan

`mindspore.mint.tan(input)`

逐元素计算输入 tensor 的正切值。

$$out_i = \tan(input_i)$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> output = mindspore.mint.tan(mindspore.tensor([-1.0, 0.0, 1.0]))
>>> print(output)
[-1.5574077  0.  1.5574077]
```


mindspore.mint.tanh

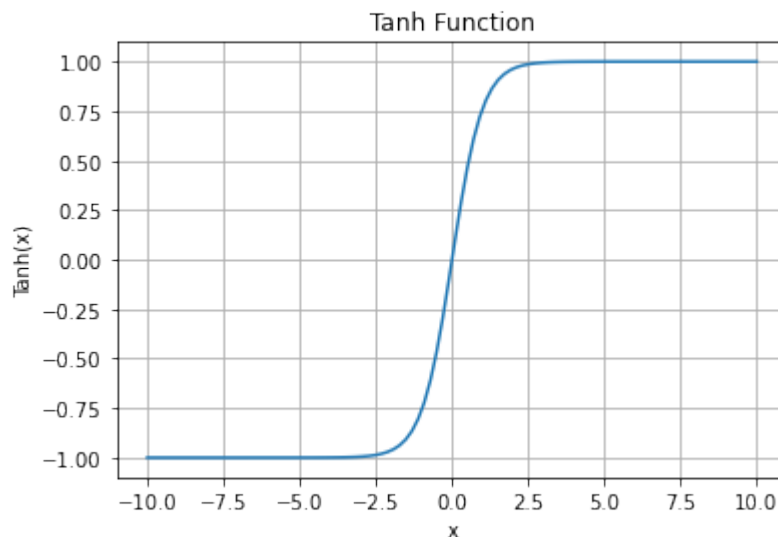
`mindspore.mint.tanh(input)`

逐元素计算输入元素的双曲正切。Tanh 函数定义为：

$$\tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 Tensor 的元素。

Tanh 函数图：



参数：

- **input** (Tensor) - Tanh 的输入。

返回：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = mint.tanh(input)
>>> print(output)
[0.7615941 0.9640276 0.9950547 0.9993293 0.9999092]
```

mindspore.mint.trunc

`mindspore.mint.trunc(input)`

返回一个 `tensor`，包含输入元素的截断整数值。

参数：

- **input** (`Tensor`) - 输入 `tensor`。

返回：

`Tensor`

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> output = mindspore.mint.trunc(mindspore.tensor([3.4742, 0.5466, -0.8008, -3.9079]))
>>> print(output)
[3. 0. 0. -3.]
```

mindspore.mint.xlogy

`mindspore.mint.xlogy(input, other)`

计算第一个输入乘以第二个输入的对数。当 `input` 为零时，则返回零。

$$out_i = input_i * \log(other_i)$$

`input` 和 `other` 的输入遵循隐式类型转换规则，使数据类型一致。输入必须是两个 `Tensor` 或一个 `Tensor` 和一个 `Scalar`。当输入是两个 `Tensor` 时，它们的 `shape` 可以广播。

参数：

- **input** (`Union[Tensor, numbers.Number, bool]`) - 第一个输入为数值型、`bool` 或数据类型为 `number` 或 `bool_` 的 `Tensor`。
- **other** (`Union[Tensor, numbers.Number, bool]`) - 第二个输入为数值型、`bool` 或数据类型为数值型或 `bool` 的 `Tensor`。当第一个输入是 `Tensor` 时，则第二个输入是数值型、`bool` 或数据类型为数值型或 `bool` 的 `Tensor`。当第一个输入是 `Scalar` 时，则第二个输入必须是数据类型为数值型或 `bool` 的 `Tensor`。

返回：

`Tensor`，`shape` 与广播后的 `shape` 相同，数据类型为两个输入中精度较高或数值较高的类型。

异常：

- **TypeError** - 如果 `input` 和 `other` 不是数值型、`bool` 或 `Tensor`。
- **ValueError** - 如果 `input` 不能广播到与 `other` 的 `shape` 一致。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([-5, 0, 4]), mindspore.float32)
>>> other = Tensor(np.array([2, 2, 2]), mindspore.float32)
>>> output = mint.xlogy(input, other)
>>> print(output)
[-3.465736  0.          2.7725887]
```

4.3.2 Reduction 运算

接口名	概述	支持平台
<code>mindspore.mint.amax</code>	计算输入 <code>input</code> 中指定 <code>dim</code> 维度上所有元素的最大值, 并根据 <code>keepdim</code> 参数决定是否保留该维度。	Ascend
<code>mindspore.mint.amin</code>	计算输入 <code>input</code> 中指定 <code>dim</code> 维度上所有元素的最小值, 并根据 <code>keepdim</code> 参数决定是否保留该维度。	Ascend
<code>mindspore.mint.argmax</code>	返回输入 <code>Tensor</code> 最大值索引。	Ascend
<code>mindspore.mint.argmin</code>	返回输入 <code>Tensor</code> 在指定轴上的最小值索引。	Ascend
<code>mindspore.mint.argsort</code>	按指定顺序对输入 <code>Tensor</code> 沿给定维度进行排序, 并返回排序后的索引。	Ascend
<code>mindspore.mint.all</code>	检查是否所有元素均为 <code>True</code> 。	Ascend
<code>mindspore.mint.any</code>	检查指定维度上是否含有 <code>True</code> 。	Ascend
<code>mindspore.mint.cumprod</code>	返回 <code>tensor</code> 在指定维度上累积的元素乘积。	Ascend
<code>mindspore.mint.histc</code>	计算 <code>Tensor</code> 的直方图。	Ascend
<code>mindspore.mint.logsumexp</code>	计算输入 <code>input</code> 中指定 <code>dim</code> 维度上所有元素指数和对数的对数 (计算过程经过数值稳定处理), 并根据 <code>keepdim</code> 参数决定是否保留该维度。	Ascend
<code>mindspore.mint.max</code>	返回输入 <code>Tensor</code> 的最大值。	Ascend
<code>mindspore.mint.mean</code>	移除输入所有维度, 返回 <code>input</code> 中所有元素的平均值。	Ascend
<code>mindspore.mint.median</code>	输出指定维度 <code>dim</code> 上的中值与其对应的索引。	Ascend
<code>mindspore.mint.min</code>	返回输入 <code>Tensor</code> 的最小值。	Ascend
<code>mindspore.mint.norm</code>	返回给定 <code>Tensor</code> 的矩阵范数或向量范数。	Ascend
<code>mindspore.mint.prod</code>	计算 <code>Tensor</code> 所有元素的乘积。	Ascend
<code>mindspore.mint.sum</code>	计算 <code>Tensor</code> 所有元素的和。	Ascend
<code>mindspore.mint.std</code>	计算指定维度 <code>dim</code> 上的标准差。	Ascend
<code>mindspore.mint.std_mean</code>	默认情况下, 返回 <code>Tensor</code> 各维度上的标准差和均值。	Ascend
<code>mindspore.mint.unique</code>	对输入 <code>Tensor</code> 中元素去重。	Ascend
<code>mindspore.mint.var</code>	计算指定维度 <code>dim</code> 上的方差。	Ascend
<code>mindspore.mint.var_mean</code>	默认情况下, 返回 <code>Tensor</code> 各维度上的方差和均值。	Ascend

mindspore.mint.amax

`mindspore.mint.amax(input, dim=(), keepdim=False)`

计算输入 *input* 中指定 *dim* 维度上所有元素的最大值，并根据 *keepdim* 参数决定是否保留该维度。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (Union[int, tuple(int), list(int)], 可选) - 需要规约的维度，数值在 $[-len(input.shape), len(input.shape) - 1]$ 之间，输入为 `()` 时规约所有维度，默认值：`()`。
- **keepdim** (bool, 可选) - 输出张量是否保留维度 *dim*，默认值：`False`。

返回：

Tensor，数据类型与 *input* 一致，shape 根据输入 *dim* 和 *keepdim* 的数值而变化。

- 如果 *dim* 为 `()`，并且 *keepdim* 为 `False`，则输出为一个零维 Tensor，表示输入 *input* 中所有元素最大值。
- 如果 *dim* 为 `1`，并且 *keepdim* 为 `False`，则输出 shape 为 $(input.shape[0], input.shape[2], \dots, input.shape[n])$ 。
- 如果 *dim* 为 $(1, 2)$ ，并且 *keepdim* 为 `False`，则输出 shape 为 $(input.shape[0], input.shape[3], \dots, input.shape[n])$ 。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *dim* 不是 int 或 tuple(int) 或 list(int)。
- **TypeError** - *keepdim* 不是 bool 类型。
- **ValueError** - *dim* 中任意元素的数值不在 $[-len(input.shape), len(input.shape) - 1]$ 之间。
- **RuntimeError** - *dim* 中任意元素重复。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> output = mint.amax(x, 1, keepdim=True)
>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by the maximum value of all elements in the dimension.
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
... [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
... [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
... mindspore.float32)
>>> output = mint.amax(x)
>>> print(output)
9.0
```

(下页继续)

(续上页)

```

>>> print(output.shape)
()
>>> # case 2: Reduces a dimension along axis 0.
>>> output = mint.amax(x, 0, True)
>>> print(output)
[[[7. 7. 7. 7. 7. 7.]
  [8. 8. 8. 8. 8. 8.]
  [9. 9. 9. 9. 9. 9.]]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = mint.amax(x, 1, True)
>>> print(output)
[[[3. 3. 3. 3. 3. 3.]
  [6. 6. 6. 6. 6. 6.]
  [9. 9. 9. 9. 9. 9.]]]
>>> # case 4: Reduces a dimension along axis 2.
>>> output = mint.amax(x, 2, True)
>>> print(output)
[[[1.]
  [2.]
  [3.]]
 [4.]
 [5.]
 [6.]]
 [7.]
 [8.]
 [9.]]]

```

mindspore.mint.amin

`mindspore.mint.amin(input, dim=(), keepdim=False)`

计算输入 *input* 中指定 *dim* 维度上所有元素的最小值，并根据 *keepdim* 参数决定是否保留该维度。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (Union[int, tuple(int), list(int)], 可选) - 需要规约的维度，数值在 $[-len(input.shape), len(input.shape) - 1]$ 之间，输入为 `()` 时规约所有维度，默认值：`()`。
- **keepdim** (bool, 可选) - 输出张量是否保留维度 *dim*，默认值：`False`。

返回：

Tensor，数据类型与 *input* 一致，shape 根据输入 *dim* 和 *keepdim* 的数值而变化。

- 如果 *dim* 为 `()`，并且 *keepdim* 为 `False`，则输出为一个零维 Tensor，表示输入 *input* 中所有元素最小值。
- 如果 *dim* 为 `1`，并且 *keepdim* 为 `False`，则输出 shape 为 $(input.shape[0], input.shape[2], \dots, input.shape[n])$ 。
- 如果 *dim* 为 $(1, 2)$ ，并且 *keepdim* 为 `False`，则输出 shape 为 $(input.shape[0], input.shape[3], \dots, input.shape[n])$ 。

异常：

- **TypeError** - *input* 不是 Tensor。

- **TypeError** - *dim* 不是 int 或 tuple(int) 或 list(int)。
- **TypeError** - *keepdim* 不是 bool 类型。
- **ValueError** - *dim* 中任意元素的数值不在 $[-len(input.shape), len(input.shape) - 1]$ 之间。
- **RuntimeError** - *dim* 中任意元素重复。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> output = mint.amin(x, 1, keepdim=True)
>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by the minimum value of all elements in the dimension.
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
... [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
... [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
... mindspore.float32)
>>> output = mint.amin(x)
>>> print(output)
1.0
>>> print(output.shape)
()
>>> # case 2: Reduces a dimension along axis 0.
>>> output = mint.amin(x, 0, True)
>>> print(output)
[[[1. 1. 1. 1. 1. 1.]
  [2. 2. 2. 2. 2. 2.]
  [3. 3. 3. 3. 3. 3.]]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = mint.amin(x, 1, True)
>>> print(output)
[[[1. 1. 1. 1. 1. 1.]
  [4. 4. 4. 4. 4. 4.]
  [7. 7. 7. 7. 7. 7.]]]
>>> # case 4: Reduces a dimension along axis 2.
>>> output = mint.amin(x, 2, True)
>>> print(output)
[[[1.]
  [2.]
  [3.]]
 [[4.]
  [5.]
  [6.]]
 [[7.]
  [8.]
  [9.]]]
```

mindspore.mint.argmax

`mindspore.mint.argmax(input)`

返回输入 Tensor 最大值索引。

参数：

- **input** (Tensor) - 输入 Tensor。

返回：

Tensor。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.array([[1, 20, 5], [67, 8, 9], [130, 24, 15]]).astype(np.float32))
>>> output = mint.argmax(x)
>>> print(output)
6
```

`mindspore.mint.argmax(input, dim, keepdim=False)`

返回输入 Tensor 在指定轴上的最大值索引。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (int) - 指定计算轴。
- **keepdim** (bool, 可选) - 输出 Tensor 是否保留指定轴。默认值：False。

返回：

Tensor，输出为指定轴上输入 Tensor 最大值的索引。

异常：

- **TypeError** - 如果 `keepdim` 的类型不是 bool 值。
- **ValueError** - 如果 `dim` 的设定值超出了范围。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.array([[1, 20, 5], [67, 8, 9], [130, 24, 15]]).astype(np.float32))
>>> output = mint.argmax(x, dim=-1)
>>> print(output)
[1 0 0]
```

mindspore.mint.argmaxin

`mindspore.mint.argmaxin(input, dim=None, keepdim=False)`

返回输入 Tensor 在指定轴上的最小值索引。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (Union[int, None], 可选) - 指定计算轴。如果是 None，将会返回扁平化 Tensor 在指定轴上的最小值索引。默认值：None。
- **keepdim** (bool, 可选) - 输出 Tensor 是否保留指定轴。如果 dim 是 None，忽略该选项。默认值：False。

返回：

Tensor，输出为指定轴上输入 Tensor 最小值的索引。

异常：

- **TypeError** - 如果 keepdim 的类型不是 bool 值。
- **ValueError** - 如果 dim 的设定值超出了范围。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.array([[1, 20, 5], [67, 8, 9], [130, 24, 15]]).astype(np.float32))
>>> output = mint.argmaxin(x, dim=-1)
>>> print(output)
[0 1 2]
```

mindspore.mint.argsort

`mindspore.mint.argsort(input, dim=-1, descending=False, stable=False)`

按指定顺序对输入 Tensor 沿给定维度进行排序，并返回排序后的索引。

警告：这是一个实验性 API，后续可能修改或删除

参数：

- **input** (Tensor) - 待排序的输入 Tensor。
- **dim** (int, 可选) - 指定排序的轴。默认值：-1，表示指定最后一维。当前 Ascend 后端只支持对最后一维进行排序。
- **descending** (bool, 可选) - 输出顺序。如果 descending 为 True，按照元素值降序排序，否则升序排序。默认值：False。
- **stable** (bool, 可选) - 是否使用稳定排序算法。默认值：False。

返回：

Tensor，排序后输入 Tensor 的索引。数据类型为 int64。

异常:

- **ValueError** - 如果 *dim* 的设置值超出了范围。
- **TypeError** - 如果 *dim* 的数据类型不是 `int32`。
- **TypeError** - 如果 *descending* 的数据类型不是 `bool`。
- **TypeError** - 如果 *stable* 的数据类型不是 `bool`。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> import mindspore.mint as mint
>>> x = Tensor(np.array([[8, 2, 1], [5, 9, 3], [4, 6, 7]]), mindspore.float16)
>>> sort = mint.argsort(x)
>>> print(sort)
[[2 1 0]
 [2 0 1]
 [0 1 2]]
```

mindspore.mint.all

`mindspore.mint.all(input)`

检查是否所有元素均为 `True`。

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[True, False], [True, True]])
>>> output = mindspore.mint.all(input)
>>> print(output)
False
```

`mindspore.mint.all(input, dim, keepdim=False)`

检查指定维度上是否所有元素均为 `True`。

参数:

- **input** (Tensor) - 输入 tensor。
- **dim** (Union[int, tuple(int), list(int), Tensor]) - 要减少的维度。如果为 `None`，减少所有维度。
- **keepdim** (bool, 可选) - 输出 tensor 是否保留维度。默认 `False`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: Reduces a dimension along dim 1, with keepdim False.
>>> mindspore.mint.all(input, dim=1)
Tensor(shape=[2], dtype=Bool, value= [False,  True])
>>>
>>> # case 2: Reduces a dimension along dim (0, 1), with keepdim False.
>>> mindspore.mint.all(input, dim=(0,1))
Tensor(shape=[], dtype=Bool, value= False)
>>>
>>> # case 3: Reduces a dimension along dim [0, 1], with keepdim True.
>>> mindspore.mint.all(input, dim=[0,1], keepdim=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[False]])
```

mindspore.mint.any

`mindspore.mint.any(input, dim=None, keepdim=False)`

检查指定维度上是否含有 `True`。

参数：

- **input** (Tensor) - 输入 tensor。
- **dim** (Union[int, tuple(int), list(int), Tensor], 可选) - 要减少的维度。如果为 `None`，减少所有维度。默认 `None`。
- **keepdim** (bool, 可选) - 输出 tensor 是否保留维度。默认 `False`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([[True, False], [True, True]])
>>>
>>> # case 1: By default, mindspore.mint.any tests along all the axes.
>>> mindspore.mint.any(input)
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 2: Reduces a dimension along dim 1, with keepdim False.
>>> mindspore.mint.any(input, dim=1)
Tensor(shape=[2], dtype=Bool, value= [ True,  True])
>>>
>>> # case 3: Reduces a dimension along dim (0, 1), with keepdim False.
```

(下页继续)

(续上页)

```
>>> mindspore.mint.any(input, dim=(0,1))
Tensor(shape=[], dtype=Bool, value= True)
>>>
>>> # case 4: Reduces a dimension along dim [0, 1], with keepdim True.
>>> mindspore.mint.any(input, dim=[0,1], keepdim=True)
Tensor(shape=[1, 1], dtype=Bool, value=
[[ True]])
```

mindspore.mint.cumprod

`mindspore.mint.cumprod(input, dim, dtype=None)`

返回 `tensor` 在指定维度上累积的元素乘积。

$$y_i = x_1 * x_2 * x_3 * \dots * x_i$$

参数:

- **input** (Tensor) - 输入 `tensor`。
- **dim** (int) - 指定计算维度。
- **dtype** (`mindspore.dtype`, 可选) - 返回的数据类型。默认 `None`。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([[1, 2, 3],
...                             [4, 5, 6]])
>>> mindspore.mint.cumprod(input, dim=0)
Tensor(shape=[2, 3], dtype=Int64, value=
[[ 1,  2,  3],
 [ 4, 10, 18]])
>>> mindspore.mint.cumprod(input, dim=1)
Tensor(shape=[2, 3], dtype=Int64, value=
[[ 1,  2,  6],
 [ 4, 20, 120]])
```

mindspore.mint.histc

`mindspore.mint.histc(input, bins=100, min=0, max=0)`

计算 `Tensor` 的直方图。

元素被分类到 `min` 和 `max` 之间的等宽箱中。如果 `min` 和 `max` 均为 0，则使用数据的最小值和最大值。

低于最小值和高于最大值的元素将被忽略。

警告: 这是一个实验性 API，后续可能修改或删除。如果 `input` 是 `int64`，有效取值在 `int32` 范围内，超出范围可能产生精度误差。

参数：

- **input** (Tensor) - 输入 Tensor。
- **bins** (int, 可选) - 直方图箱的数量，可选。默认值：100。若指定，则必须为正数。
- **min** (int, float, 可选) - 范围下限（含），可选。默认值：0。
- **max** (int, float, 可选) - 范围上限（含），可选。默认值：0。

返回：

1-D Tensor, shape 为 (bins,)，数据类型与 *input* 相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *input* 的数据类型不支持。
- **TypeError** - 如果属性 *min* 或 *max* 不是 float 类型或 int 类型。
- **TypeError** - 如果属性 *bins* 不是整数。
- **ValueError** - 如果属性 *min* > *max*。
- **ValueError** - 如果属性 *bins* <= 0。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor, mint
>>> x = Tensor([1., 2, 1])
>>> y = mint.histc(x, bins=4, min=0, max=3)
>>> print(y)
[0 2 1 0]
```

mindspore.mint.logsumexp

`mindspore.mint.logsumexp(input, dim, keepdim=False)`

计算输入 *input* 中指定 *dim* 维度上所有元素指数和的对数（计算过程经过数值稳定处理），并根据 *keepdim* 参数决定是否保留该维度。

$$\text{logsumexp}(\text{input}) = \log\left(\sum(e^{\text{input}-\text{input}_{\max}})\right) + \text{input}_{\max}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (Union[int, tuple(int), list(int)], 可选) - 需要规约的维度（数值在 $[0, \text{len}(\text{input.shape}) - 1]$ 之间），输入为 () 时规约所有维度。
- **keepdim** (bool, 可选) - 输出张量是否保留维度 *dim*，默认值：False。

返回：

Tensor, 数据类型 `dtype` 根据 `input.dtype` 变化, `shape` 根据输入 `dim` 和 `keepdim` 的数值而变化。

- 如果 `input.dtype` 为 `[float16, float32, bfloat16]`, 输出数据类型 `dtype` 与输入 `input.dtype` 相同。
- 如果 `input.dtype` 为整数或布尔类型, 输出数据类型 `dtype` 为 `float32`。
- 如果 `dim` 为 `()`, 并且 `keepdim` 为 `False`, 则输出为一个零维 Tensor, 表示输入 `input` 中所有元素指数和对数的对数。
- 如果 `dim` 为 `1`, 并且 `keepdim` 为 `False`, 则输出 `shape` 为 `(input.shape[0], input.shape[2], ..., input.shape[n])`。
- 如果 `dim` 为 `(1, 2)`, 并且 `keepdim` 为 `False`, 则输出 `shape` 为 `(input.shape[0], input.shape[3], ..., input.shape[n])`。

异常:

- **TypeError** - `input` 不是 Tensor。
- **TypeError** - `input` 的数据类型不是 `bool`、`int8`、`int16`、`int32`、`int64`、`uint8`、`float16`、`float32`、`bfloat16` 之一。
- **TypeError** - `dim` 不是 `int` 或 `tuple(int)` 或 `list(int)`。
- **TypeError** - `keepdim` 不是 `bool` 类型。
- **ValueError** - `dim` 中任意元素的数值不在 `[0, len(input.shape) - 1]` 之间。
- **RuntimeError** - `dim` 中任意元素重复。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> output = mint.logsumexp(x, 1, keepdim=True)
>>> print(output.shape)
(3, 1, 5, 6)
```

mindspore.mint.max

`mindspore.mint.max(input)`

返回输入 Tensor 的最大值。

参数:

- **input** (Tensor) - 输入的 Tensor。

返回:

Tensor, 值为输入 Tensor 的最大值, 类型与 `input` 相同。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output = mint.max(x)
```

(下页继续)

(续上页)

```
>>> print(output)
0.7
```

```
mindspore.mint.max(input, dim, keepdim=False)
```

在给定轴上计算输入 Tensor 的最大值，并返回最大值和索引值。

参数：

- **input** (Tensor) - 输入任意维度的 Tensor，设置输入 Tensor 的 shape 为 $(input_1, input_2, \dots, input_N)$ ，不支持 complex 类型。
- **dim** (int) - 指定计算维度。
- **keepdim** (bool, 可选) - 表示是否减少维度，如果为 True，输出将与输入保持相同的维度；如果为 False，输出将减少维度。默认值：False。

返回：

tuple(Tensor)，返回两个元素类型为 Tensor 的 tuple，包含输入 Tensor 沿指定维度 *dim* 的最大值和相应的索引。

- **values** (Tensor) - 输入 Tensor 沿给定维度的最大值，数据类型和 *input* 相同，shape 和 *index* 相同。
- **index** (Tensor) - 输入 Tensor 的沿给定维度的最大值索引，数据类型为 *int64*。如果 *keepdim* 为 True，输出 Tensor 的维度是 $(input_1, input_2, \dots, input_{dim-1}, 1, input_{dim+1}, \dots, input_N)$ 。否则输出维度为 $(input_1, input_2, \dots, input_{dim-1}, input_{dim+1}, \dots, input_N)$ 。

异常：

- **TypeError** - 如果 *input* 不是 tensor。
- **TypeError** - 如果 *keepdim* 不是 bool 类型。
- **TypeError** - 如果 *dim* 不是 int 类型。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output, index = mint.max(x, 0, keepdim=True)
>>> print(output, index)
[0.7] [3]
```

```
mindspore.mint.max(input, other)
```

详情请参考 [mindspore.mint.maximum\(\)](#)。

支持平台：

Ascend

mindspore.mint.mean

`mindspore.mint.mean(input, *, dtype=None)`

移除输入所有维度, 返回 *input* 中所有元素的平均值。

参数:

- **input** (Tensor[Number]) - 输入 Tensor, 其数据类型为数值型。shape: (N,*), 其中 * 表示任意数量的附加维度。

关键字参数:

- **dtype** (*mindspore.dtype*, 可选) - 期望输出 Tensor 的类型。默认值: None。

返回:

Tensor。

异常:

- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[[[2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2]],
... [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
... [[6, 6, 6, 6, 6, 6], [8, 8, 8, 8, 8, 8], [10, 10, 10, 10, 10, 10]]]),
... mindspore.float32)
>>> output = mint.mean(x)
>>> print(output)
5.0
>>> print(output.shape)
()
```

`mindspore.mint.mean(input, dim, keepdim=False, *, dtype=None)`

默认情况下, 移除输入所有维度, 返回 *input* 中所有元素的平均值。也可仅缩小指定维度 *dim* 大小至 1。*keepdim* 控制输出和输入的维度是否相同。

说明: Tensor 类型的 *dim* 仅用作兼容旧版本, 不推荐使用。

参数:

- **input** (Tensor[Number]) - 输入 Tensor, 其数据类型为数值型。shape: (N,*), 其中 * 表示任意数量的附加维度。
- **dim** (Union[int, tuple(int), list(int), Tensor]) - 要减少的维度。只允许常量值。假设 *input* 的秩为 *r*, 取值范围 [-r,r]。
- **keepdim** (bool) - 如果为 True, 则保留缩小的维度, 大小为 1。否则移除维度。默认值: False。

关键字参数:

- **dtype** (*mindspore.dtype*, 可选) - 期望输出 Tensor 的类型。默认值: None。

返回：

Tensor。

- 如果 *dim* 为 int，取值为 1，并且 *keepdim* 为 False，则输出的 shape 为 $(input_0, input_2, \dots, input_R)$ 。
- 如果 *dim* 为 tuple(int) 或 list(int)，取值为 (1, 2)，并且 *keepdim* 为 False，则输出 Tensor 的 shape 为 $(input_0, input_3, \dots, input_R)$ 。
- 如果 *dim* 为一维 Tensor，例如取值为 [1, 2]，并且 *keepdim* 为 False，则输出 Tensor 的 shape 为 $(input_0, input_3, \dots, input_R)$ 。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *dim* 不是以下数据类型之一：int、tuple、list 或 Tensor。
- **TypeError** - *keepdim* 不是 bool 类型。
- **ValueError** - *dim* 超出范围。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[[2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2]],
... [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
... [[6, 6, 6, 6, 6, 6], [8, 8, 8, 8, 8, 8], [10, 10, 10, 10, 10, 10]]]),
... mindspore.float32)
>>> output = mint.mean(x, 0, True)
>>> print(output)
[[4. 4. 4. 4. 4. 4.]
 [5. 5. 5. 5. 5. 5.]
 [6. 6. 6. 6. 6. 6.]])
```

mindspore.mint.median

`mindspore.mint.median(input, dim=None, keepdim=False)`

输出指定维度 *dim* 上的中值与其对应的索引。如果 *dim* 为 None，则计算 Tensor 中所有元素的中值。

参数：

- **input** (Tensor) - 任意维度的 Tensor，支持的数据类型为：uint8、int16、int32、int64、float16 或 float32。
- **dim** (int, 可选) - 指定计算的轴。默认值：None。
- **keepdim** (bool, 可选) - 是否保留 *dim* 指定的维度。默认值：False。

返回：

- **y** (Tensor) - 输出中值，数据类型与 *input* 相同。
 - 如果 *dim* 为 None，*y* 只有一个元素。
 - 如果 *keepdim* 为 True，*y* 的 shape 除了在 *dim* 维度上为 1 外与 *input* 一致。
 - 其他情况下，*y* 比 *input* 缺少 *dim* 指定的维度。

- **indices** (Tensor) - 中值的索引。shape 与 y 一致，数据类型为 int64。

异常：

- **TypeError** - input 不是以下数据类型之一：uint8、int16、int32、int64、float16、float32。
- **TypeError** - input 不是 Tensor。
- **TypeError** - dim 不是 int。
- **TypeError** - keepdim 不是 bool 值。
- **ValueError** - dim 不在 [-x.dim, x.dim-1] 范围内。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[0.57, 0.11, 0.21], [0.38, 0.50, 0.57], [0.36, 0.16, 0.44]]).
    ↳ astype(np.float32))
>>> y = mint.median(x, dim=0, keepdim=False)
>>> print(y)
(Tensor(shape=[3], dtype=Float32, value= [ 3.79999995e-01,  1.59999996e-01,  4.39999998e-
    ↳ 01]),
Tensor(shape=[3], dtype=Int64, value= [1, 2, 2]))
```

mindspore.mint.min

`mindspore.mint.min(input)`

返回输入 Tensor 的最小值。

参数：

- **input** (Tensor) - 输入的 Tensor。

返回：

Tensor，值为输入 Tensor 的最小值，类型与 *input* 相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output = mint.min(x)
>>> print(output)
0.0
```

`mindspore.mint.min(input, dim, keepdim=False)`

在给定轴上计算输入 Tensor 的最小值，并返回最小值和索引值。

参数：

- **input** (Tensor) - 输入任意维度的 Tensor，设置输入 Tensor 的 shape 为 $(input_1, input_2, \dots, input_N)$ ，不支持 complex 类型。
- **dim** (int) - 指定计算维度。
- **keepdim** (bool, 可选) - 表示是否减少维度，如果为 True，输出将与输入保持相同的维度；如果为 False，输出将减少维度。默认值：False。

返回：

tuple(Tensor)，返回两个元素类型为 Tensor 的 tuple，包含输入 Tensor 沿指定维度 *dim* 的最小值和相应的索引。

- **values** (Tensor) - 输入 Tensor 沿给定维度的最小值，shape 和 *index* 相同，数据类型和 *self* 相同。
- **index** (Tensor) - 输入 Tensor 的沿给定维度的最小值索引，数据类型为 *int64*。如果 *keepdim* 为 True，输出 Tensor 的维度是 $(input_1, input_2, \dots, input_{dim-1}, 1, input_{dim+1}, \dots, input_N)$ 。否则输出维度为 $(input_1, input_2, \dots, input_{dim-1}, input_{dim+1}, \dots, input_N)$ 。

异常：

- **TypeError** - 如果 *input* 不是 tensor。
- **TypeError** - 如果 *keepdim* 不是 bool 类型。
- **TypeError** - 如果 *dim* 不是 int 类型。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> output, index = mint.min(x, 0, keepdim=True)
>>> print(output, index)
[0.0] [0]
```

`mindspore.mint.min(input, other)`

详情请参考 `mindspore.mint.minimum()`。

支持平台：

Ascend

mindspore.mint.norm

`mindspore.mint.norm(input, p='fro', dim=None, keepdim=False, *, dtype=None)`

返回给定 Tensor 的矩阵范数或向量范数。

p 为范数的计算模式。支持下列范数模式。

<i>p</i>	矩阵范数	向量范数
'fro'	Frobenius 范数	不支持
'nuc'	Nuclear 范数	不支持
其余 int 或 float 值	不支持	$sum(abs(x)^p)^{(1/p)}$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - shape 为 (*) 或者 $(*, m, n)$ 的 Tensor，其中 * 是零个或多个 batch 维度。
- **p** (Union[bool, int, float, inf, -inf, 'fro', 'nuc'], 可选) - 范数的计算模式。行为参考上表。默认值：'fro'。
- **dim** (Union[int, List(int), Tuple(int)], 可选) - 计算向量范数或矩阵范数的维度。默认值：None。
- **keepdim** (bool, 可选) - 输出 Tensor 是否保留原有的维度。默认值：False。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 如果设置此参数，则会在执行之前将 *input* 转换为指定的类型，返回的 Tensor 类型也将为指定类型 *dtype*。默认值：None。

返回：

Tensor，在指定维度 *dim* 上进行范数计算的结果。

异常：

- **TypeError** - *input* 不是一个 Tensor。
- **ValueError** - *dim* 超出范围。
- **TypeError** - *dim* 既不是 int 也不是由 int 组成的 tuple 或 list。
- **ValueError** - *dim* 的两个元素在标准化过后取值相同。
- **ValueError** - *dim* 的任意元素超出索引。

说明：动态 shape、动态 rank 和可变输入不支持在 图模式 (*mode=mindspore.GRAPH_MODE*) 下执行。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import mint, ops
>>> data_range = ops.arange(-13, 13, dtype=ms.float32)
>>> x = data_range[data_range != 0]
>>> y = x.reshape(5, 5)
>>> print(mint.norm(x, 2.0))
38.327538
```

mindspore.mint.prod

`mindspore.mint.prod(input, *, dtype=None)`

计算 Tensor 所有元素的乘积。

参数：

- **input** (Tensor[Number]) - 输入 Tensor，其数据类型为数值型。shape: (N,*), 其中 * 表示任意数量的附加维度。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 期望输出 Tensor 的类型。默认值: None。

返回：

Tensor。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
               ↪mindspore.float32)
>>> output = mint.prod(x)
>>> print(output)
2.2833798e+33
>>> print(output.shape)
()
```

`mindspore.mint.prod(input, dim, keepdim=False, *, dtype=None)`

默认情况下，使用指定维度的所有元素的乘积代替该维度的其他元素，以移除该维度。也可仅缩小该维度大小至 1。
keepdim 控制输出和输入的维度是否相同。

参数：

- **input** (Tensor[Number]) - 输入 Tensor，其数据类型为数值型。shape: (N,*), 其中 * 表示任意数量的附加维度。
- **dim** (int) - 要减少的维度。只允许常量值。假设 *input* 的秩为 *r*，取值范围 [-*r*,*r*)。
- **keepdim** (bool) - 如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值: False。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 期望输出 Tensor 的类型。默认值: None。

返回：

Tensor。

- 如果 *dim* 为 int，取值为 1，并且 *keepdim* 为 False，则输出的 shape 为 (*input*₀, *input*₂, ..., *input*_R)。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *dim* 不是 int。
- **TypeError** - *keepdim* 不是 bool 类型。
- **ValueError** - *dim* 超出范围。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]])),
mindspore.float32)
>>> output = mint.prod(x, 0, True)
>>> print(output)
[[[ 28.  28.  28.  28.  28.  28.]
  [ 80.  80.  80.  80.  80.  80.]
  [162. 162. 162. 162. 162. 162.]]]
```

mindspore.mint.sum

`mindspore.mint.sum(input, *, dtype=None)`

计算 Tensor 所有元素的和。

参数：

- **input** (Tensor) - 输入 Tensor。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 期望输出 Tensor 的类型。默认值：None。

返回：

Tensor, *input* 所有元素的和。

异常：

- **TypeError** - *input* 不是 Tensor 类型。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]])),
mindspore.float32)
```

(下页继续)

(续上页)

```
>>> out = mint.sum(x)
>>> print(out)
270.0
```

`mindspore.mint.sum(input, dim, keepdim=False, *, dtype=None)`

计算 Tensor 指定维度元素的和。

说明: Tensor 类型的 `dim` 仅用作兼容旧版本, 不推荐使用。

参数:

- **input** (Tensor) - 输入 Tensor。
- **dim** (Union[int, tuple(int), list(int), Tensor]) - 求和的维度。如果 `dim` 为 int 组成的 tuple 或 list, 将对 tuple 中的所有维度求和, 取值范围必须在 `[-input.ndim, input.ndim)`。
- **keepdim** (bool) - 是否保留输出 Tensor 的维度, 如果为 True, 保持对应的维度且长度为 1。如果为 False, 不保持维度。默认值: False。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 期望输出 Tensor 的类型。默认值: None。

返回:

Tensor, `input` 指定维度的和。

异常:

- **TypeError** - `input` 不是 Tensor 类型。
- **TypeError** - `dim` 类型不是 int, tuple(int), list(int) 或 Tensor。
- **ValueError** - `dim` 取值不在 `[-input.ndim, input.ndim)` 范围。
- **TypeError** - `keepdim` 不是 bool 类型。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
               mstype.float32)
>>> out = mint.sum(x)
>>> print(out)
270.0
>>> out = mint.sum(x, dim=2)
>>> print(out)
[[ 6. 12. 18.]
 [24. 30. 36.]
```

(下页继续)

(续上页)

```
[42. 48. 54.]]
>>> out = mint.sum(x, dim=2, keepdim=True)
>>> print(out)
[[ [ 6.]
   [12.]
   [18.]]
 [24.]
 [30.]
 [36.]]
 [[42.]
  [48.]
  [54.]]]
```

mindspore.mint.std

`mindspore.mint.std(input, dim=None, *, correction=1, keepdim=False)`

计算指定维度 `dim` 上的标准差。`dim` 可以是单个维度、维度列表，也可以是 `None`，表示移除所有维度。

标准差 (σ) 计算如下：

$$\sigma = \sqrt{\frac{1}{N - \delta N} \sum_{j=1}^{N-1} (self_{ij} - \bar{x}_i)^2}$$

其中 x 表示用来计算标准差的样本集, $\bar{x}$ 表示样本的均值, N 表示样本的数量, δN 则为 `correction` 的值。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 用来求标准差的 Tensor。
- **dim** (None, int, tuple(int), 可选) - 用来进行规约计算的维度。默认值为 `None`，所有维度都进行规约计算。

关键字参数：

- **correction** (int, 可选) - 样本大小和样本自由度之间的差异。默认为 Bessel 校正，默认值为 1。
- **keepdim** (bool, 可选) - 是否保留输出 Tensor 的维度。如果为 `True`，则保留缩小的维度，其大小为 1，否则移除维度。默认值为 `False`。

返回：

Tensor，标准差。假设输入 `input` 的 shape 为 $(x_0, x_1, \dots, x_R)$ ：

- 如果 `dim` 为 `()`，且 `keepdim` 为 `False`，则返回一个零维 Tensor，表示输入 Tensor `input` 中所有元素的标准差。
- 如果 `dim` 为 `int`，如取值为 1，且 `keepdim` 为 `False`，则返回 Tensor 的 shape 为 $(x_0, x_2, \dots, x_R)$ 。
- 如果 `dim` 为 `tuple(int)` 或者 `list(int)`，如取值为 `(1, 2)`，且 `keepdim` 为 `False`，则返回 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。

异常：

- **TypeError** - 如果 `input` 不是 Tensor。
- **TypeError** - 如果 `input` 的数据类型不是 `bfloat16`，`float16` 或 `float32`。
- **TypeError** - 如果 `dim` 不是 `None`，`int` 或 `tuple` 类型。

- **TypeError** - 如果 *correction* 不是 int 类型。
- **TypeError** - 如果 *keepdim* 不是 bool 类型。
- **ValueError** - 如果 *dim* 不在 $[-input.ndim, input.ndim)$ 范围内。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import mint, Tensor
>>> input = Tensor(np.array([[1, 2, 3], [-1, 1, 4]]).astype(np.float32))
>>> output = mint.std(input, dim=1, correction=1, keepdim=False)
>>> print(output)
[1.      2.5166113]
```

mindspore.mint.std_mean

`mindspore.mint.std_mean(input, dim=None, *, correction=1, keepdim=False)`

默认情况下，返回 Tensor 各维度上的标准差和均值。如果 *dim* 是维度列表，则计算对应维度的标准差和均值。

标准差 (σ) 计算如下：

$$\sigma = \sqrt{\frac{1}{N - \delta N} \sum_{j=0}^{N-1} (sel f_{ij} - \bar{x}_i)^2}$$

其中 x 表示用来计算标准差的样本集， $\bar{x}$ 表示样本的均值， N 表示样本的数量， δN 为 *correction* 的值。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。支持的数据类型：float16、float32。
- **dim** (Union[int, tuple(int), list(int)], 可选) - 指定计算标准差和平均值的维度。默认值：None。

关键字参数：

- **correction** (int, 可选) - 样本大小和样本自由度之间的差异。默认采用贝塞尔校正。默认值：1。
- **keepdim** (bool, 可选) - 是否保留输出 Tensor 的维度。如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值：False。

返回：

包含标准差和均值的 tuple。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *dim* 不是以下数据类型之一：int、tuple、list 或 Tensor。
- **TypeError** - *keepdim* 不是 bool 类型。
- **ValueError** - *dim* 超出范围。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> input = ms.Tensor([[1, 2, 3, 4], [-1, 1, 4, -10]], ms.float32)
>>> output_std, output_mean = ms.mint.std_mean(input, 1, correction=2, keepdim=True)
>>> print(output_std)
[[1.5811388]
 [7.3824115]]
>>> print(output_mean)
[[ 2.5]
 [-1.5]]
```

mindspore.mint.unique

`mindspore.mint.unique(input, sorted=True, return_inverse=False, return_counts=False, dim=None)`

对输入 Tensor 中元素去重。

在 `return_inverse=True` 时, 会返回一个索引 Tensor, 包含输入 Tensor 中的元素在输出 Tensor 中的索引;

在 `return_counts=True` 时, 会返回一个 Tensor, 表示输出元素在输入中的个数。

参数:

- **input** (Tensor) - 输入 Tensor。
- **sorted** (bool, 可选) - 输出是否需要进行升序排序。默认值: True。
- **return_inverse** (bool, 可选) - 是否输出 `input` 在 `output` 上对应的 index。默认值: False。
- **return_counts** (bool, 可选) - 是否输出 `output` 中元素的数量。默认值: False。
- **dim** (int, 可选) - 做去重操作的维度, 当设置为 None 的时候, 对展开的输入做去重操作, 否则, 将给定维度的 Tensor 视为一个元素去做去重操作。默认值: None。

返回:

输出为一个 Tensor, 或者以下一个或几个 Tensor 的集合: (`output`, `inverse_indices`, `counts`)

- **output** (Tensor) - 与 `input` 数据类型相同的 Tensor, 包含 `input` 中去重后的元素。
- **inverse_indices** (Tensor) - 当 `return_inverse=True` 时返回, 表示输入 Tensor 中的元素在输出 Tensor 中的索引。当 `dim=None` 时, `shape` 和 `input` 一样; 当 `dim` 有值的时候, `shape` 是 `input.shape[dim]`。
- **counts** (Tensor) - 当 `return_counts=True` 时返回, 表示输出 Tensor 中元素在输入 Tensor 中的数量。当 `dim=None` 时, `shape` 和 `output` 一样; 当 `dim` 有值的时候, `shape` 是 `output.shape[dim]`。

异常:

- **TypeError** - `input` 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> from mindspore import mint
>>> x = Tensor(np.array([1, 2, 5, 2]), mindspore.int32)
>>> output = mint.unique(x, return_inverse=True)
>>> print(output)
(Tensor(shape=[3], dtype=Int32, value= [1, 2, 5]), Tensor(shape=[4], dtype=Int64, value= [0, 1, 2, 1]))
>>> y = output[0]
>>> print(y)
[1 2 5]
>>> idx = output[1]
>>> print(idx)
[0 1 2 1]
```

mindspore.mint.var

`mindspore.mint.var(input, dim=None, *, correction=1, keepdim=False)`

计算指定维度 `dim` 上的方差。`dim` 可以是单个维度、维度列表，也可以是 `None`，表示移除所有维度。

方差 (δ^2) 计算如下：

$$\delta^2 = \frac{1}{\max(0, N - \delta N)} \sum_{i=0}^{N-1} (x_i - \bar{x})^2$$

其中 x 表示用来计算方差的样本集， $\bar{x}$ 表示样本的均值， N 表示样本的数量， δN 则为 `correction` 的值。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 用来求方差的 Tensor。
- **dim** (None, int, tuple(int), 可选) - 用来进行规约计算的维度。默认值为 `None`，所有维度都进行规约计算。

关键字参数：

- **correction** (int, 可选) - 样本大小和样本自由度之间的差异。默认为 Bessel 校正，默认值为 1。
- **keepdim** (bool, 可选) - 是否保留输出 Tensor 的维度。如果为 `True`，则保留缩小的维度，其大小为 1，否则移除维度。默认值为 `False`。

返回：

Tensor，方差。假设输入 `input` 的 shape 为 $(x_0, x_1, \dots, x_R)$ ：

- 如果 `dim` 为 `()`，且 `keepdim` 为 `False`，则返回一个零维 Tensor，表示输入 Tensor `input` 中所有元素的方差。
- 如果 `dim` 为 `int`，如取值为 1，且 `keepdim` 为 `False`，则返回 Tensor 的 shape 为 $(x_0, x_2, \dots, x_R)$ 。
- 如果 `dim` 为 `tuple(int)` 或者 `list(int)`，如取值为 `(1, 2)`，且 `keepdim` 为 `False`，则返回 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。

异常：

- **TypeError** - 如果 `input` 不是 Tensor。
- **TypeError** - 如果 `input` 的数据类型不是 `bfloat16`，`float16` 或 `float32`。

- **TypeError** - 如果 *dim* 不是 None, int, list 或 tuple 类型。
- **TypeError** - 如果 *correction* 不是 int 类型。
- **TypeError** - 如果 *keepdim* 不是 bool 类型。
- **ValueError** - 如果 *dim* 不在 $[-input.ndim, input.ndim)$ 范围内。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([[8, 2, 1], [5, 9, 3], [4, 6, 7]], mindspore.float32)
>>> output = mint.var(input, dim=0, correction=1, keepdim=True)
>>> print(output)
[[ 4.333333, 12.333333, 9.333333]]
```

mindspore.mint.var_mean

`mindspore.mint.var_mean(input, dim=None, *, correction=1, keepdim=False)`

默认情况下, 返回 Tensor 各维度上的方差和均值。如果 *dim* 是维度列表, 则计算对应维度的方差和均值。

方差 (σ^2) 计算如下:

$$\sigma^2 = \frac{1}{N - \delta N} \sum_{j=0}^{N-1} (self_{ij} - \bar{x}_i)^2$$

其中 x 表示用来计算方差的样本集, $\bar{x}$ 表示样本的均值, N 表示样本的数量, δN 为 *correction* 的值。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 Tensor。支持的数据类型: float16、float32。
- **dim** (Union[int, tuple(int), list(int)], 可选) - 指定计算方差和平均值的维度。默认值: None。

关键字参数:

- **correction** (int, 可选) - 样本大小和样本自由度之间的差异。默认采用贝塞尔校正。默认值: 1。
- **keepdim** (bool, 可选) - 是否保留输出 Tensor 的维度。如果为 True, 则保留缩小的维度, 大小为 1。否则移除维度。默认值: False。

返回:

包含方差和均值的 tuple。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *dim* 不是以下数据类型之一: int、tuple、list 或 Tensor。
- **TypeError** - *keepdim* 不是 bool 类型。
- **ValueError** - *dim* 超出范围。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> input = ms.Tensor([[1, 2, 3, 4], [-1, 1, 4, -10]], ms.float32)
>>> output_var, output_mean = ms.mint.var_mean(input, 1, correction=2, keepdim=True)
>>> print(output_var)
[[ 2.5]
 [54.5]]
>>> print(output_mean)
[[ 2.5]
 [-1.5]]
```

4.3.3 比较运算

接口名	概述	支持平台
<code>mindspore.mint.allclose</code>	返回一个布尔型标量，表示 <i>input</i> 的每个元素均与 <i>other</i> 的对应元素在给定容忍度内“接近”。	Ascend
<code>mindspore.mint.argsort</code>	按指定顺序对输入 Tensor 沿给定维度进行排序，并返回排序后的索引。	Ascend
<code>mindspore.mint.eq</code>	逐元素计算两个输入是否相等。	Ascend
<code>mindspore.mint.equal</code>	比较两个输入是否相等。	Ascend
<code>mindspore.mint.greater</code>	逐元素计算 <i>input</i> > <i>other</i> 的值。	Ascend
<code>mindspore.mint.greater_equal</code>	按元素比较输入参数 <i>input</i> >= <i>other</i> 的值，输出结果为 bool 值。	Ascend
<code>mindspore.mint.gt</code>	逐元素计算 <i>input</i> > <i>other</i> 的值。	Ascend
<code>mindspore.mint.isclose</code>	返回一个布尔型 tensor ，表示两个 tensor 在容忍度内是否逐元素相等。	Ascend
<code>mindspore.mint.isfinite</code>	判断输入数据每个位置上的元素是否是有限数。	Ascend
<code>mindspore.mint.isinf</code>	返回一个布尔型 tensor ，指示哪些元素是正负无穷数。	Ascend
<code>mindspore.mint.isneginf</code>	确定输入 Tensor 每个位置上的元素是否为负无穷。	Ascend
<code>mindspore.mint.le</code>	逐元素计算 <i>input</i> <= <i>other</i> 的值。	Ascend
<code>mindspore.mint.less</code>	逐元素计算 <i>input</i> < <i>other</i> 的值。	Ascend
<code>mindspore.mint.less_equal</code>	逐元素计算 <i>input</i> <= <i>other</i> 的值。	Ascend
<code>mindspore.mint.lt</code>	<code>mindspore.mint.less()</code> 的别名。	Ascend
<code>mindspore.mint.maximum</code>	逐元素计算两个输入 tensor 中最大值。	Ascend
<code>mindspore.mint.minimum</code>	逐元素计算两个输入 tensor 的最小值。	Ascend
<code>mindspore.mint.ne</code>	逐元素计算两个输入是否不相等。	Ascend
<code>mindspore.mint.not_equal</code>	<code>mindspore.mint.ne()</code> 的别名。	Ascend
<code>mindspore.mint.topk</code>	沿给定维度查找 <i>k</i> 个最大或最小元素和对应的索引。	Ascend
<code>mindspore.mint.sort</code>	按指定顺序对输入 Tensor 的指定维上的元素进行排序。	Ascend

mindspore.mint.allclose

`mindspore.mint.allclose(input, other, rtol=1e-05, atol=1e-08, equal_nan=False)`

返回一个布尔型标量，表示 *input* 的每个元素均与 *other* 的对应元素在给定容忍度内“接近”。其中“接近”的数学公式为：

$$|input - other|atol + rtol \times |other|$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 对比的第一个输入，支持的类型有 float16、float32、float64、int8、int16、int32、int64、uint8，Ascend 平台额外支持 bfloat16 和 bool 类型。
- **other** (Tensor) - 对比的第二个输入，数据类型必须与 *input* 相同。
- **rtol** (Union[float, int, bool], 可选) - 相对容忍度。默认值：1e-05。
- **atol** (Union[float, int, bool], 可选) - 绝对容忍度。默认值：1e-08。
- **equal_nan** (bool, 可选) - 若为 True，则两个 NaN 被视为相同。默认值：False。

返回：

布尔型标量。

异常：

- **TypeError** - *input* 和 *other* 中的任何一个不是 Tensor。
- **TypeError** - *input* 和 *other* 的数据类型不在支持的类型列表中。
- **TypeError** - *atol* 和 *rtol* 中的任何一个不是 float、int 或 bool。
- **TypeError** - *equal_nan* 不是 bool。
- **TypeError** - *input* 和 *other* 的数据类型不同。
- **ValueError** - *input* 和 *other* 无法广播。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1.3, 2.1, 3.2, 4.1, 5.1]), mindspore.float16)
>>> other = Tensor(np.array([1.3, 3.3, 2.3, 3.1, 5.1]), mindspore.float16)
>>> output = mint.allclose(input, other)
>>> print(output)
False
```

mindspore.mint.eq

`mindspore.mint.eq(input, other)`
逐元素计算两个输入是否相等。

$$out_i = \begin{cases} \text{True, if } input_i = other_i \\ \text{False, if } input_i \neq other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 输入必须是两个 Tensor，或是一个 Tensor 和一个 Scalar。
- 两个输入的 shape 支持广播。

参数:

- **input** (Union[Tensor, Number]) - 第一个输入。
- **other** (Union[Tensor, Number]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> x = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.eq(x, 2.0)
>>> print(output)
[False True False]
>>> # case 2: The shape of two inputs are the same
>>> x = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> y = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.mint.eq(x, y)
>>> print(output)
[ True True False]
```

mindspore.mint.equal

`mindspore.mint.equal(input, other)`
比较两个输入是否相等。

说明: *input* 和 *other* 遵循隐式类型转换规则，使数据类型保持一致。

警告: 这是一个实验性 API，后续可能修改或删除。

参数:

- **input** (Tensor) - 第一个输入。
- **other** (Tensor) - 第二个输入。

返回:

bool。

异常:

- **TypeError** - 如果 *input* 或 *other* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> x = Tensor([1, 2, 3], mindspore.int32)
>>> y = Tensor([1, 2, 4], mindspore.int32)
>>> output = mint.equal(x, y)
>>> print(output)
False
```

mindspore.mint.greater

`mindspore.mint.greater` (*input*, *other*)

逐元素计算 $input > other$ 的值。

参数:

- **input** (Union[Tensor, Number]) - 第一个输入。
- **other** (Union[Tensor, Number]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.greater(input, 2.0)
>>> print(output)
[False False True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.mint.greater(input, other)
>>> print(output)
[ False False False]
```

mindspore.mint.greater_equal

`mindspore.mint.greater_equal(input, other)`

按元素比较输入参数 $input \geq other$ 的值，输出结果为 bool 值。

$$out_i = \begin{cases} \text{True, if } input_i > other_i \\ \text{False, if } input_i \leq other_i \end{cases}$$

说明:

- 输入 *input* 和 *other* 遵循隐式类型转换规则，使数据类型保持一致。
- 输入必须是两个 Tensor，或一个 Tensor 和一个 Scalar。
- 当输入是两个 Tensor 时，它们的数据类型不能同时是 bool，并保证其 shape 可以广播。
- 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
- 支持广播。
- 若输入的 Tensor 可以广播，则会把低维度通过复制该维度的值的方式扩展到另一个输入中对应的高维度。

参数:

- **input** (Union[Tensor, Number]) - 第一个输入，是一个 Number 值或数据类型为 `number` 或 `bool_` 的 Tensor。
- **other** (Union[Tensor, Number]) - 第二个输入，当第一个输入是 Tensor 时，第二个输入应该是一个 Number 值，或数据类型为 `number` 或 `bool_` 的 Tensor。当第一个输入是 Scalar 时，第二个输入必须是数据类型为 `number` 或 `bool_` 的 Tensor。

返回:

Tensor, shape 与广播后的 shape 相同，数据类型为 bool。

异常:

- **TypeError** - *input* 和 *other* 都不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1, 2, 3]), mindspore.int32)
>>> other = Tensor(np.array([1, 1, 4]), mindspore.int32)
>>> output = mint.greater_equal(input, other)
>>> print(output)
[True True False]
>>> y = 2.1
>>> output = mint.greater_equal(input, y)
>>> print(output)
[False False True]
```


mindspore.mint.gt

`mindspore.mint.gt(input, other)`
 逐元素计算 $input > other$ 的值。

$$out_i = \begin{cases} \text{True, if } input_i > other_i \\ \text{False, if } input_i \leq other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 输入必须是两个 Tensor，或一个 Tensor 和一个 Scalar。
- 当输入是两个 Tensor 时，它们的数据类型不能同时是 bool，并保证其 shape 可以广播。
- 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
- 支持广播。
- 若输入的 Tensor 可以广播，则会把低维度通过复制该维度的值的方式扩展到另一个输入中对应的高维度。

参数:

- **input** (Union[Tensor, number.Number, bool]) - 第一个输入。
- **other** (Union[Tensor, number.Number, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.gt(input, 2.0)
>>> print(output)
[False False True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.mint.gt(input, other)
>>> print(output)
[ False False False]
```

mindspore.mint.isclose

`mindspore.mint.isclose(input, other, rtol=1e-05, atol=1e-08, equal_nan=False)`

返回一个布尔型 `tensor`，表示两个 `tensor` 在容忍度内是否逐元素相等。数学公式为：

$$|input - other|atol + rtol \times |other|$$

如果 `inf` 值具有相同的符号，则认为它们相等；如果 `equal_nan` 为 `True`，`NaN` 值被认为相等。

参数：

- **input** (`Tensor`) - 第一个输入 `tensor`。
- **other** (`Tensor`) - 第二个输入 `tensor`，数据类型和 `shape` 必须与 `input` 相同。
- **rtol** (`Union[float, int, bool]`, 可选) - 相对容忍度。
- **atol** (`Union[float, int, bool]`, 可选) - 绝对容忍度。
- **equal_nan** (`bool`, 可选) - 若为 `True`，则两个 `NaN` 被视为相等。

返回：

`Tensor`

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> mindspore.mint.isclose(mindspore.tensor([2e6, float("inf"), float("-inf"), float("inf"),
↪float("nan")]),
...                        mindspore.tensor([2e7, float("inf"), float("-inf"), float("-inf"),
↪float("nan")]))
Tensor(shape=[5], dtype=Bool, value= [False,  True,  True, False, False])
>>>
>>> mindspore.mint.isclose(mindspore.tensor([1e6, 2e6, 3e6]),
...                        mindspore.tensor([1.00008e6, 2.00008e7, 3.00008e8]), rtol=1e3)
Tensor(shape=[3], dtype=Bool, value= [ True,  True,  True])
>>>
>>> mindspore.mint.isclose(mindspore.tensor([1e6, 2e6, 3e6]),
...                        mindspore.tensor([1.00001e6, 2.00002e6, 3.00009e6]), atol=1e3)
Tensor(shape=[3], dtype=Bool, value= [ True,  True,  True])
>>> mindspore.mint.isclose(mindspore.tensor([float("nan"), 1, 2]),
...                        mindspore.tensor([float("nan"), 1, 2]), equal_nan=True)
Tensor(shape=[3], dtype=Bool, value= [ True,  True,  True])
```

mindspore.mint.isfinite

`mindspore.mint.isfinite(input)`

判断输入数据每个位置上的元素是否是有限数。如果某位置的元素不是 `NaN`，`-INF`，`INF`，则该位置的元素被认为是有限数。

$$out_i = \begin{cases} \text{if } input_i = \text{Finite}, & \text{True} \\ \text{if } input_i \neq \text{Finite}, & \text{False} \end{cases}$$

参数：

- **input** (Tensor) - IsFinite 的输入。

返回：

Tensor，输出的 shape 与输入相同，数据类型为 bool。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([np.log(-1), 1, np.log(0)]), mindspore.float32)
>>> output = mint.isfinite(x)
>>> print(output)
[False True False]
>>> x = Tensor(2.1, mindspore.float64)
>>> output = mint.isfinite(x)
>>> print(output)
True
```

mindspore.mint.isinf

`mindspore.mint.isinf(input)`

返回一个布尔型 tensor，指示哪些元素是正负无穷数。

警告：

- 这是一个实验性 API，后续可能修改或删除。
- 该 API 目前只支持在 Atlas A2 训练系列产品上使用。

参数：

- **input** (Tensor) - 输入 tensor。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor([-1, 3, float("inf"), float("-inf"), float("nan")])
>>> mindspore.mint.isinf(input)
Tensor(shape=[5], dtype=Bool, value= [False, False,  True,  True, False])
```

mindspore.mint.isneginf

`mindspore.mint.isneginf(input)`

确定输入 Tensor 每个位置上的元素是否为负无穷。

警告:

- 该 API 目前只支持在 Atlas A2 训练系列产品上使用。

参数:

- **input** (Tensor) - 输入 Tensor。

返回:

Tensor, 与输入具有相同形状, 其中元素在对应输入为负无穷大时为 True, 否则为 False。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> from mindspore import mint, Tensor
>>> from mindspore import dtype as mstype
>>> output = mint.isneginf(Tensor([[-float("inf"), float("inf")], [1, -float("inf")]],
↪mstype.float32))
>>> print(output)
[[ True False]
 [False  True]]
```

mindspore.mint.le

`mindspore.mint.le(input, other)`

逐元素计算 $input \leq other$ 的值。

$$out_i = \begin{cases} \text{True, if } input_i \leq other_i \\ \text{False, if } input_i > other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 输入必须是两个 Tensor, 或一个 Tensor 和一个 Scalar。
- 当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常数。

参数:

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.le(input, 2.0)
>>> print(output)
[True True False]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.mint.le(input, other)
>>> print(output)
[ True True True]
```

mindspore.mint.less

`mindspore.mint.less(input, other)`

逐元素计算 $input < other$ 的值。

$$out_i = \begin{cases} \text{True, if } input_i < other_i \\ \text{False, if } input_i \geq other_i \end{cases}$$

说明：支持隐式类型转换。

参数：

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.less(input, 2.0)
>>> print(output)
[True False False]
>>> # case 2: The shape of two inputs are the same
```

(下页继续)

(续上页)

```
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.mint.less(input, other)
>>> print(output)
[ False  False  True]
```

mindspore.mint.less_equal

`mindspore.mint.less_equal(input, other)`

逐元素计算 $input \leq other$ 的值。

$$out_i = \begin{cases} \text{True, if } input_i \leq other_i \\ \text{False, if } input_i > other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常数。

参数:

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.less_equal(input, 2.0)
>>> print(output)
[True  True False]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.mint.less_equal(input, other)
>>> print(output)
[ True  True  True]
```

mindspore.mint.lt

`mindspore.mint.lt(input, other)`
`mindspore.mint.less()` 的别名。

支持平台:

Ascend

mindspore.mint.maximum

`mindspore.mint.maximum(input, other)`
 逐元素计算两个输入 tensor 中最大值。

$$output_i = \max(input_i, other_i)$$

说明:

- 输入 `input` 和 `other` 遵循隐式类型转换规则，使数据类型保持一致。
- 当输入是两个 Tensor 时，它们的数据类型不能同时是 bool，并保证其 shape 可以广播。
- 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
- 支持广播。
- 如果一个元素和 NaN 比较，则返回 NaN。

警告: 如果所有输入都为标量 int 类型，在 Graph 模式下，输出为 int32 类型的 Tensor，在 PyNative 模式下，输出为 int64 类型的 Tensor。

参数:

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # case 1 : same data type
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.float32)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float32)
>>> mindspore.mint.maximum(input, other)
Tensor(shape=[3], dtype=Float32, value= [ 4.00000000e+00,  5.00000000e+00,  6.00000000e+00])
>>>
>>> # case 2 : the data type is the one with higher precision or higher digits among the two
↳ inputs.
```

(下页继续)

(续上页)

```
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.int64)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float64)
>>> mindspore.mint.maximum(input, other)
Tensor(shape=[3], dtype=Float64, value= [ 4.00000000e+00,  5.00000000e+00,  6.00000000e+00])
```

mindspore.mint.minimum

`mindspore.mint.minimum(input, other)`

逐元素计算两个输入 `tensor` 的最小值。

$$output_i = \min(input_i, other_i)$$

说明:

- 输入 `input` 和 `other` 遵循隐式类型转换规则, 使数据类型保持一致。
- 当输入是两个 `Tensor` 时, 它们的数据类型不能同时是 `bool`。
- 当输入是一个 `Tensor` 和一个 `Scalar` 时, `Scalar` 只能是一个常数。
- 支持广播操作。
- 当一个元素与 `NaN` 比较时, 将返回 `NaN`。

参数:

- **input** (`Union[Tensor, Number, bool]`) - 第一个输入。
- **other** (`Union[Tensor, Number, bool]`) - 第二个输入。

返回:

`Tensor`

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # case 1 : same data type
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.float32)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float32)
>>> mindspore.mint.minimum(input, other)
Tensor(shape=[3], dtype=Float32, value= [ 1.00000000e+00,  2.00000000e+00,  3.00000000e+00])
>>>
>>> # case 2 : the data type is the one with higher precision or higher digits among the two
↳ inputs.
>>> input = mindspore.tensor([1.0, 5.0, 3.0], mindspore.int64)
>>> other = mindspore.tensor([4.0, 2.0, 6.0], mindspore.float64)
>>> mindspore.mint.minimum(input, other)
Tensor(shape=[3], dtype=Float64, value= [ 1.00000000e+00,  2.00000000e+00,  3.00000000e+00])
```


`mindspore.mint.ne`

`mindspore.mint.ne(input, other)`
逐元素计算两个输入是否不相等。

$$out_i = \begin{cases} \text{True, if } input_i \neq other_i \\ \text{False, if } input_i = other_i \end{cases}$$

说明:

- 支持隐式类型转换。
- 当输入是两个 Tensor 时，它们的 shape 可以广播。
- 当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常数。
- 支持广播。

参数:

- **input** (Union[Tensor, Number, bool]) - 第一个输入。
- **other** (Union[Tensor, Number, bool]) - 第二个输入。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> # case 1: The shape of two inputs are different
>>> input = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.ne(input, 2.0)
>>> print(output)
[True False  True]
>>> # case 2: The shape of two inputs are the same
>>> input = mindspore.tensor([1, 2, 3], mindspore.int32)
>>> other = mindspore.tensor([1, 2, 4], mindspore.int32)
>>> output = mindspore.mint.ne(input, other)
>>> print(output)
[ False False  True]
```

`mindspore.mint.not_equal`

`mindspore.mint.not_equal(input, other)`
`mindspore.mint.ne()` 的别名。

支持平台:

Ascend

mindspore.mint.topk

`mindspore.mint.topk(input, k, dim=-1, largest=True, sorted=True)`

沿给定维度查找 k 个最大或最小元素和对应的索引。

警告:

- 如果 `sorted` 设置为 `False`, 由于在不同平台上存在内存排布以及遍历方式不同等问题, `sorted` 设置为 `False` 时计算结果的显示顺序可能会出现不一致的情况。

如果 `input` 是一维 Tensor, 则查找 Tensor 中 k 个最大或最小元素, 并将其值和索引输出为 Tensor。 `values[k]` 是 `input` 中 k 个最大元素, 其索引是 `indices[k]`。

对于多维矩阵, 计算给定维度中最大或最小的 k 个元素, 因此:

$$values.shape = indices.shape$$

如果两个比较的元素相同, 则优先返回索引值较小的元素。

参数:

- **input** (Tensor) - 需计算的输入。
- **k** (int) - 指定计算最大或最小元素的数量。
- **dim** (int, 可选) - 需要排序的维度。默认值: `-1`。
- **largest** (bool, 可选) - 如果为 `False`, 则会返回前 k 个最小值。默认值: `True`。
- **sorted** (bool, 可选) - 如果为 `True`, 则获取的元素将根据设置的 `largest` 对值进行升序或降序排序。如果为 `False`, 则不对获取的元素进行排序。默认值: `True`。

返回:

由 `values` 和 `indices` 组成的 tuple。

- **values** (Tensor) - 给定维度的每个切片中的 k 最大元素或最小元素。
- **indices** (Tensor) - k 最大元素的对应索引。

异常:

- **TypeError** - 如果 `sorted` 不是 bool。
- **TypeError** - 如果 `input` 不是 Tensor。
- **TypeError** - 如果 `k` 不是 int。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> x = ms.Tensor([[0.5368, 0.2447, 0.4302, 0.9673],
...               [0.4388, 0.6525, 0.4685, 0.1868],
...               [0.3563, 0.5152, 0.9675, 0.8230]], dtype=ms.float32)
>>> output = mint.topk(x, 2, dim=1)
>>> print(output)
(Tensor(shape=[3, 2], dtype=Float32, value=
```

(下页继续)

(续上页)

```
[[ 9.67299998e-01,  5.36800027e-01],
 [ 6.52499974e-01,  4.68499988e-01],
 [ 9.67499971e-01,  8.23000014e-01]])], Tensor(shape=[3, 2], dtype=Int64, value=
[[3, 0],
 [1, 2],
 [2, 3]]))
>>> output2 = mint.topk(x, 2, dim=1, largest=False)
>>> print(output2)
(Tensor(shape=[3, 2], dtype=Float32, value=
[[ 2.44700000e-01,  4.30200011e-01],
 [ 1.86800003e-01,  4.38800007e-01],
 [ 3.56299996e-01,  5.15200019e-01]])], Tensor(shape=[3, 2], dtype=Int64, value=
[[1, 2],
 [3, 0],
 [0, 1]]))
```

mindspore.mint.sort

`mindspore.mint.sort(input, *, dim=-1, descending=False, stable=False)`

按指定顺序对输入 Tensor 的指定维上的元素进行排序。

警告：目前能良好支持的数据类型有：float16、uint8、int8、int16、int32、int64。如果使用 float32，可能产生精度误差。

参数：

- **input** (Tensor) - 进行排序的 Tensor，shape 为 $(N,*)$ ，其中 * 表示任意数量的额外维度。

关键字参数：

- **dim** (int, 可选) - 指定排序的轴。默认值：-1，表示指定最后一维。
- **descending** (bool, 可选) - 按降序还是升序。如果为 True，则元素按降序排列，否则按升序排列。默认值：False。
- **stable** (bool, 可选) - 按稳定排序还是非稳定排序。如果为 True，则是稳定排序，否则为非稳定排序。默认值：False。

返回：

- **y1** (Tensor) - 排序后的值，其 shape 和数据类型与输入一致。
- **y2** (Tensor) - 返回值在原输入 Tensor 里对应的索引，数据类型为 int64。

异常：

- **TypeError** - dim 不是 int 类型。
- **TypeError** - descending 不是 bool 类型。
- **TypeError** - input 不是 float16、float32、uint8、int8、int16、int32、int64 或 bfloat16。
- **TypeError** - stable 不是 bool 类型。
- **ValueError** - dim 不在 $[-\text{len}(\text{input.shape}), \text{len}(\text{input.shape})]$ 范围内。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[8, 2, 1], [5, 9, 3], [4, 6, 7]]), mindspore.float16)
>>> output = mint.sort(x)
>>> # The output below is based on the Ascend platform.
>>> print(output)
(Tensor(shape=[3, 3], dtype=Float16, value=
[[ 1.0000e+00,  2.0000e+00,  8.0000e+00],
[ 3.0000e+00,  5.0000e+00,  9.0000e+00],
[ 4.0000e+00,  6.0000e+00,  7.0000e+00]]), Tensor(shape=[3, 3], dtype=Int64, value=
[[2, 1, 0],
[2, 0, 1],
[0, 1, 2]]))
```

4.3.4 BLAS 和 LAPACK 运算

接口名	概述	支持平台
<code>mindspore.mint.addbmm</code>	对 <i>batch1</i> 和 <i>batch2</i> 应用批量矩阵乘法后进行规约加, <i>input</i> 和最终的结果相加。	Ascend
<code>mindspore.mint.addmm</code>	对输入的两个二维矩阵 <i>mat1</i> 与 <i>mat2</i> 相乘, 并将结果与 <i>input</i> 相加。	Ascend
<code>mindspore.mint.baddbmm</code>	对输入的两个三维矩阵 <i>batch1</i> 与 <i>batch2</i> 相乘, 并将结果与 <i>input</i> 相加。	Ascend
<code>mindspore.mint.bmm</code>	基于 <i>batch</i> 维度的两个 <i>Tensor</i> 的矩阵乘法, 仅支持三维。	Ascend
<code>mindspore.mint.dot</code>	计算两个 <i>1D</i> <i>Tensor</i> 的点积。	Ascend
<code>mindspore.mint.inverse</code>	计算输入矩阵的逆。	Ascend
<code>mindspore.mint.matmul</code>	计算两个输入的矩阵乘积。	Ascend
<code>mindspore.mint.meshgrid</code>	从给定的 <i>Tensor</i> 生成网格矩阵。	Ascend
<code>mindspore.mint.mm</code>	计算两个矩阵的乘积。	Ascend
<code>mindspore.mint.outer</code>	计算 <i>input</i> 和 <i>vec2</i> 的外积。	Ascend
<code>mindspore.mint.trace</code>	返回 <i>input</i> 的主对角线方向上的总和。	Ascend

mindspore.mint.addbmm

`mindspore.mint.addbmm(input, batch1, batch2, *, beta=1, alpha=1)`
对 *batch1* 和 *batch2* 应用批量矩阵乘法后进行规约加, *input* 和最终的结果相加。*alpha* 是 *batch1* 和 *batch2* 矩阵乘法的乘数, *beta* 是 *input* 的乘数。如果 *beta* 是 0, 那么 *input* 将会被忽略。

$$output = \beta input + \alpha (\sum_{i=0}^{b-1} batch1_i @ batch2_i)$$

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (*Tensor*) - 被添加的 *Tensor*。

- **batch1** (Tensor) - 矩阵乘法中的第一个 Tensor。
- **batch2** (Tensor) - 矩阵乘法中的第二个 Tensor。

关键字参数：

- **beta** (Union[int, float], 可选) - *input* 的乘数。默认值：1。
- **alpha** (Union[int, float], 可选) - *batch1 @ batch2* 的乘数。默认值：1。

返回：

Tensor, 和 *input* 具有相同的 dtype。

异常：

- **TypeError** - 如果 *alpha*, *beta* 不是 int 或者 float。
- **ValueError** - 如果 *batch1*, *batch2* 不能进行批量矩阵乘法。
- **ValueError** - 如果 *batch1* 或 *batch2* 的不是三维 Tensor。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> m = np.ones((3, 3)).astype(np.float32)
>>> arr1 = np.arange(24).astype(np.float32).reshape((2, 3, 4))
>>> arr2 = np.arange(24).astype(np.float32).reshape((2, 4, 3))
>>> a = Tensor(arr1)
>>> b = Tensor(arr2)
>>> c = Tensor(m)
>>> output = mint.addbmm(c, a, b)
>>> print(output)
[[ 949. 1009. 1069.]
 [1285. 1377. 1469.]
 [1621. 1745. 1869.]]
```

mindspore.mint.addmm

`mindspore.mint.addmm(input, mat1, mat2, *, beta=1, alpha=1)`

对输入的两个二维矩阵 *mat1* 与 *mat2* 相乘，并将结果与 *input* 相加。计算公式定义如下：

$$output = \beta input + \alpha(mat1 @ mat2)$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 待增加的 Tensor。其形状必须可广播与 *mat1@mat2* 一致。
- **mat1** (Tensor) - 第一个待乘矩阵。必须为 2-D 的 Tensor，类型与 *input* 一致。
- **mat2** (Tensor) - 第二个待乘矩阵。必须为 2-D 的 Tensor，类型与 *input* 一致。

关键字参数：

- **beta** (Union[float, int], 可选) - 输入的乘数。默认值: 1。
- **alpha** (Union[float, int], 可选) - $mat1 @ mat2$ 的系数, 默认值: 1。

返回:

Tensor, 其数据类型与 *input* 相同, 其 shape 和 $mat1 @ mat2$ 相同。

异常:

- **TypeError** - *input*、*mat1* 或 *mat2* 的类型不是 Tensor。
- **TypeError** - *input*、*mat1* 或 *mat2* 数据类型不一致。
- **ValueError** - *mat1* 或 *mat2* 不是二维 Tensor。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.ones([3, 3]).astype(np.float32))
>>> mat1 = Tensor(np.ones([3, 4]).astype(np.float32))
>>> mat2 = Tensor(np.ones([4, 3]).astype(np.float32))
>>> output = mint.addmm(input, mat1, mat2)
>>> print(output)
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
```

mindspore.mint.baddbmm

`mindspore.mint.baddbmm(input, batch1, batch2, *, beta=1, alpha=1)`

对输入的两个三维矩阵 *batch1* 与 *batch2* 相乘, 并将结果与 *input* 相加。计算公式定义如下:

$$out_i = \beta input_i + \alpha (batch1_i @ batch2_i)$$

参数:

- **input** (Tensor) - 输入 Tensor。当 *batch1* 是 (C, W, T) 的 Tensor 而且 *batch2* 是一个 (C, T, H) 的 Tensor 时, 输入必须为可以被广播为 (C, W, H) 形状的 Tensor。
- **batch1** (Tensor) - 公式中的 *batch1*。必须为 3-D 的 Tensor, 类型与 *input* 一致。
- **batch2** (Tensor) - 公式中的 *batch2*。必须为 3-D 的 Tensor, 类型与 *input* 一致。

关键字参数:

- **beta** (Union[float, int], 可选) - 输入的乘数。默认值: 1。
- **alpha** (Union[float, int], 可选) - $batch1 @ batch2$ 的系数, 默认值: 1。

返回:

Tensor, 其数据类型与 *input* 相同, 维度为 (C, W, H) 。

异常:

- **TypeError** - *input*、*batch1* 或 *batch2* 的类型不是 Tensor。
- **TypeError** - *input*、*batch1* 或 *batch2* 数据类型不一致。

- **ValueError** - *batch1* 或 *batch2* 不是三维 Tensor。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.ones([1, 3, 3]).astype(np.float32))
>>> batch1 = Tensor(np.ones([1, 3, 4]).astype(np.float32))
>>> batch2 = Tensor(np.ones([1, 4, 3]).astype(np.float32))
>>> output = mint.baddbmm(input, batch1, batch2)
>>> print(output)
[[[5. 5. 5.]
  [5. 5. 5.]
  [5. 5. 5.]]]
```

mindspore.mint.bmm

`mindspore.mint.bmm(input, mat2)`

基于 *batch* 维度的两个 Tensor 的矩阵乘法，仅支持三维。

$$\text{output} = \text{input} @ \text{mat2}$$

参数：

- **input** (Tensor) - 输入相乘的第一个 Tensor。必须是三维 Tensor，shape 为 (b, n, m) 。
- **mat2** (Tensor) - 输入相乘的第二个 Tensor。必须是三维 Tensor，shape 为 (b, m, p) 。

返回：

Tensor，输出 Tensor 的 shape 为 (b, n, p) 。其中每个矩阵是输入批次中相应矩阵的乘积。

异常：

- **ValueError** - *input* 或 *mat2* 的维度不为 3。
- **ValueError** - *input* 第三维的长度不等于 *mat2* 第二维的长度。
- **ValueError** - *input* 的 *batch* 维长度不等于 *mat2* 的 *batch* 维长度。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> a = Tensor(np.ones(shape=[2, 3, 4]), mindspore.float32)
>>> b = Tensor(np.ones(shape=[2, 4, 5]), mindspore.float32)
>>> output = mint.bmm(a, b)
>>> print(output)
[[[4. 4. 4. 4. 4.]
  [4. 4. 4. 4. 4.]
  [4. 4. 4. 4. 4.]]]
```

(下页继续)

(续上页)

```
[[4. 4. 4. 4. 4.]
 [4. 4. 4. 4. 4.]
 [4. 4. 4. 4. 4.]]]
```

mindspore.mint.dot

`mindspore.mint.dot(input, other)`

计算两个 1DTensor 的点积。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 点积的第一个输入, 须为 1D。
- **other** (Tensor) - 点积的第二个输入, 须为 1D。

返回：

Tensor, shape 是 [], 类型与 input 一致。

异常：

- **TypeError** - *input* 和 *other* 的数据类型不是 tensor。
- **TypeError** - *input* 或 *other* 的数据类型不是 float16, float32, 或 bfloat16。
- **RuntimeError** - *input* 和 *other* 的数据类型不一致。
- **RuntimeError** - *input* 和 *other* 的 shape 不一致。
- **RuntimeError** - *input* 或 *other* 不是 1D。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> x = Tensor([2.0, 3.0], mindspore.float32)
>>> y = Tensor([2.0, 1.0], mindspore.float32)
>>> output = mint.dot(x, y)
>>> print(output)
7.0
>>> print(output.dtype)
Float32
```


mindspore.mint.inverse

`mindspore.mint.inverse(input)`

计算输入矩阵的逆。

参数:

- **input** (Tensor) - 计算的矩阵。*input* 至少是两维的, 最后两个维度大小相同, 并且矩阵需要可逆。

返回:

Tensor, 其类型和 shape 与 *input* 相同。

异常:

- **ValueError** - *input* 最后两个维度的大小不相同。
- **ValueError** - *input* 不是空并且它的维度小于 2 维。
- **ValueError** - *input* 的维度大于 6。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> x = Tensor([[1., 2.], [3., 4.]], mstype.float32)
>>> print(mint.inverse(x))
[[-2.   1. ]
 [ 1.5 -0.5]]
```

mindspore.mint.matmul

`mindspore.mint.matmul(input, other)`

计算两个输入的矩阵乘积。

说明:

- *input* 和 *other* 的数据类型必须一致, 不支持 Scalar, 两者须支持广播。
- Ascend 平台, *input* 和 *other* 的秩必须在 1 到 6 之间。

参数:

- **input** (Tensor) - 第一个输入 tensor。
- **other** (Tensor) - 第二个输入 tensor。

返回:

Tensor 或 Scalar

支持平台:

Ascend

样例:

```

>>> import mindspore
>>> # case 1 : Reasonable application of broadcast mechanism.
>>> input = mindspore.ops.arange(24, dtype=mindspore.float32).reshape(2, 3, 4)
>>> other = mindspore.ops.arange(20, dtype=mindspore.float32).reshape(4, 5)
>>> output = mindspore.mint.matmul(input, other)
>>> print(output)
[[[ 70,  76,  82,  88,  94],
  [ 190,  212,  234,  256,  278],
  [ 310,  348,  386,  424,  462]],
 [[ 430,  484,  538,  592,  646],
  [ 550,  620,  690,  760,  830],
  [ 670,  756,  842,  928, 1014]]]
>>>
>>> # case 2 : The rank of `input` is 1.
>>> input = mindspore.ops.ones([1, 2])
>>> other = mindspore.ops.ones([2])
>>> mindspore.mint.matmul(input, other)
Tensor(shape=[1], dtype=Float32, value= [ 2.00000000e+00])

```

mindspore.mint.meshgrid

`mindspore.mint.meshgrid(*tensors, indexing='ij')`

从给定的 Tensor 生成网格矩阵。

给定 N 个一维 Tensor，对每个 Tensor 做扩张操作，返回 N 个 N 维的 Tensor。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **tensors** (Union(tuple[Tensor], list[Tensor])) - 静态图下为 N 个一维 Tensor，输入的 Tensor 个数应大于 1。动态图下为 N 个零维或一维 Tensor，输入的 Tensor 个数应大于 0。数据类型为 Number。

关键字参数：

- **indexing** (str, 可选) - 影响输出的网格矩阵的 size。可选值为：'xy' 或 'ij'。对于长度为 M 和 N 的二维输入，取值为 'xy' 时，输出的 shape 为 (N, M) ，取值为 'ij' 时，输出的 shape 为 (M, N) 。以长度为 M ， N 和 P 的三维输入，取值为 'xy' 时，输出的 shape 为 (N, M, P) ，取值为 'ij' 时，输出的 shape 为 (M, N, P) 。默认值：'ij'。

返回：

Tensor，N 个 N 维 tensor 对象的元组。数据类型与输入相同。

异常：

- **TypeError** - `indexing` 不是 str 或 `tensors` 不是元组。
- **ValueError** - `indexing` 的取值既不是 'xy' 也不是 'ij'。

支持平台：

Ascend

样例：

```

>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.array([1, 2, 3, 4]).astype(np.int32))
>>> y = Tensor(np.array([5, 6, 7]).astype(np.int32))
>>> z = Tensor(np.array([8, 9, 0, 1, 2]).astype(np.int32))
>>> output = mint.meshgrid(x, y, z, indexing='xy')
>>> print(output)
(Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2],
  [3, 3, 3, 3, 3],
  [4, 4, 4, 4, 4]],
 [[1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2],
  [3, 3, 3, 3, 3],
  [4, 4, 4, 4, 4]],
 [[1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2],
  [3, 3, 3, 3, 3],
  [4, 4, 4, 4, 4]]]),
 Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5]],
 [[6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6]],
 [[7, 7, 7, 7, 7],
  [7, 7, 7, 7, 7],
  [7, 7, 7, 7, 7],
  [7, 7, 7, 7, 7]]]),
 Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]],
 [[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]],
 [[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]]])

```

mindspore.mint.mm

`mindspore.mint.mm(input, mat2)`

计算两个矩阵的乘积。

如果 *input* 是一个 $(n \times m)$ 的 Tensor, *mat2* 是一个 $(m \times p)$ 的 Tensor, *out* 则会是一个 $(n \times p)$ 的 Tensor。

说明: 此函数不能支持广播。若需要可广播的方法, 请参考 `mindspore.mint.matmul()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 矩阵相乘的第一个矩阵, *input* 的最后一维度必须和 *mat2* 的第一维度相等。
- **mat2** (Tensor) - 矩阵相乘的第二个矩阵, *input* 的最后一维度必须和 *mat2* 的第一维度相等。

返回:

Tensor, 输入的矩阵乘积。

异常:

- **ValueError** - *input* 的最后一维度和 *mat2* 的倒数第二维度不相等。
- **TypeError** - *input* 或者 *mat2* 不是一个 Tensor。
- **TypeError** - *input* 或者 *mat2* 的数据类型不是 float16、float32、bfloat16 之一。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> import numpy as np
>>> x1 = ms.Tensor(np.random.rand(2, 3), ms.float32)
>>> x2 = ms.Tensor(np.random.rand(3, 4), ms.float32)
>>> out = mint.mm(x1, x2)
>>> print(out.shape)
(2, 4)
```

mindspore.mint.outer

`mindspore.mint.outer(input, vec2)`

计算 *input* 和 *vec2* 的外积。如果向量 *input* 长度为 n , *vec2* 长度为 m , 则输出矩阵 shape 为 (n, m) 。

警告: 这是一个实验性 API, 后续可能修改或删除。

说明: 该函数不支持广播。

参数：

- **input** (Tensor) - 输入一维向量。
- **vec2** (Tensor) - 输入一维向量。

返回：

out, 两个一维向量的外积, 是一个二维矩阵。

异常：

- **TypeError** - 如果 *input* 或 *vec2* 不是 Tensor。
- **TypeError** - *input* 和 *vec2* 隐式转换后的数据类型不是 float16、float32、float64、bool、uint8、int8、int16、int32、int64、complex64、complex128、bfloat16 之一
- **ValueError** - 如果 *input* 或 *vec2* 的维度不是 1。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> input = Tensor(np.array([7, 8, 9]), mindspore.int32)
>>> vec2 = Tensor(np.array([7, 10, 11]), mindspore.int32)
>>> out = mint.outer(input, vec2)
>>> print(out)
[[49 70 77]
 [56 80 88]
 [63 90 99]]
```

mindspore.mint.trace

mindspore.mint.trace(*input*)

返回 *input* 的主对角线方向上的总和。

参数：

- **input** (Tensor) - 二维 Tensor。

返回：

Tensor, 当 *input* 为数据类型为整型或 bool 时其数据类型为 int64, 反之与 *input* 一致, 含有一个元素。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **ValueError** - 如果 *input* 的维度不是 2。
- **TypeError** - *input* 的数据类型不是 float16、float32、float64、bool、uint8、int8、int16、int32、int64、complex64、complex128、bfloat16 之一。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[10, 11, 12], [13, 14, 15], [16, 17, 18]]), mindspore.float32)
>>> output = mint.trace(input)
>>> print(output)
42.0
>>> input = Tensor(np.arange(1, 13).reshape(3, 4), mindspore.float32)
>>> output = mint.trace(input)
>>> print(output)
18.0
>>> input = Tensor(np.arange(12, 0, -1).reshape(4, 3), mindspore.float32)
>>> output = mint.trace(input)
>>> print(output)
24.0
```

4.3.5 其他运算

接口名	概述	支持平台
<code>mindspore.mint.broadcast_to</code>	将输入的 <code>shape</code> 广播到目标 <code>shape</code> 。	Ascend
<code>mindspore.mint.cdists</code>	计算两个 <code>Tensor</code> 每对行向量之间的 <code>p-norm</code> 距离。	Ascend
<code>mindspore.mint.cummax</code>	返回一个元组 (最值、索引)，其中最值是输入 <code>Tensor input</code> 沿维度 <code>dim</code> 的累积最大值，索引是每个最大值的索引位置。	Ascend
<code>mindspore.mint.cummin</code>	返回一个元组 (最值、索引)，其中最值是输入 <code>Tensor input</code> 沿维度 <code>dim</code> 的累积最小值，索引是每个最小值的索引位置。	Ascend
<code>mindspore.mint.cumsum</code>	计算输入 <code>Tensor input</code> 沿轴 <code>dim</code> 的累积和。	Ascend
<code>mindspore.mint.diag</code>	如果 <code>input</code> 是向量 (1-D 张量)，则返回一个二维张量，其中 <code>input</code> 的元素作为对角线。	Ascend
<code>mindspore.mint.flatten</code>	沿着从 <code>start_dim</code> 到 <code>end_dim</code> 的维度，对输入 <code>Tensor</code> 进行展平。	Ascend
<code>mindspore.mint.flip</code>	沿指定维度翻转 <code>tensor</code> 中元素。	Ascend
<code>mindspore.mint.repeat_interleave</code>	沿着轴重复 <code>Tensor</code> 的元素，类似 <code>mindspore.numpy.repeat()</code> 。	Ascend
<code>mindspore.mint.searchsorted</code>	返回元素能够插入输入 <code>tensor</code> 的位置索引，以维持原 <code>tensor</code> 的递增顺序。	Ascend
<code>mindspore.mint.tril</code>	将指定对角线上方的元素设置为 0。	Ascend
<code>mindspore.mint.triangular_solve</code>	求解正上三角形或下三角形可逆矩阵 <code>A</code> 和包含多个元素的右侧边 <code>b</code> 的方程组的解。	Ascend

mindspore.mint.broadcast_to

`mindspore.mint.broadcast_to(input, shape)`

将输入的 `shape` 广播到目标 `shape`。输入维度必须小于等于目标维度，设输入 `shape` 为 $(x_1, x_2, \dots, x_m)$ ，目标 `shape` 为 $(*, y_1, y_2, \dots, y_m)$ ，其中 `*` 为任意额外的维度。广播规则如下：

依次比较 x_m 与 y_m 、 x_{m-1} 与 y_{m-1} 、 $\dots$ 、 x_1 与 y_1 的值确定是否可以广播以及广播后输出 `shape` 对应维的值。

- 如果相等，则这个值即为目标 `shape` 该维的值。比如说输入 `shape` 为 $(2, 3)$ ，目标 `shape` 为 $(2, 3)$ ，则输出 `shape` 为 $(2, 3)$ 。
- 如果不相等，分以下三种情况：
 - 情况一：如果目标 `shape` 该维的值为 -1，则输出 `shape` 该维的值为对应输入 `shape` 该维的值。比如说输入 `shape` 为 $(3, 3)$ ，目标 `shape` 为 $(-1, 3)$ ，则输出 `shape` 为 $(3, 3)$ 。
 - 情况二：如果目标 `shape` 该维的值不为 -1，但是输入 `shape` 该维的值为 1，则输出 `shape` 该维的值为目标 `shape` 该维的值。比如说输入 `shape` 为 $(1, 3)$ ，目标 `shape` 为 $(8, 3)$ ，则输出 `shape` 为 $(8, 3)$ 。
 - 情况三：如果两个 `shape` 对应值不满足以上情况则说明不支持由输入 `shape` 广播到目标 `shape`。

至此输出 `shape` 后面 `m` 维就确定好了，现在看一下前面 `*` 维，有以下两种情况：

- 如果额外的 `*` 维中不含有 -1，则输入 `shape` 从低维度补充维度使之与目标 `shape` 维度一致，比如说目标 `shape` 为 $(3, 1, 4, 1, 5, 9)$ ，输入 `shape` 为 $(1, 5, 9)$ ，则输入 `shape` 增维变成 $(1, 1, 1, 1, 5, 9)$ ，根据上面提到的情况二可以得出输出 `shape` 为 $(3, 1, 4, 1, 5, 9)$ ；
- 如果额外的 `*` 维中含有 -1，说明此时该 -1 对应一个不存在的维度，不支持广播。比如说目标 `shape` 为 $(3, -1, 4, 1, 5, 9)$ ，输入 `shape` 为 $(1, 5, 9)$ ，此时不进行增维处理，而是直接报错。

参数：

- **input** (Tensor) - 输入 tensor。
- **shape** (tuple) - 目标 shape。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> shape = (2, 3)
>>> x = mindspore.tensor([1, 2, 3], mindspore.float32)
>>> output = mindspore.mint.broadcast_to(x, shape)
>>> print(output)
[[1. 2. 3.]
 [1. 2. 3.]]
>>> shape = (-1, 2)
>>> x = mindspore.tensor([[1], [2]], mindspore.float32)
>>> output = mindspore.mint.broadcast_to(x, shape)
>>> print(output)
[[1. 1.]
 [2. 2.]]
```

mindspore.mint.cdists

`mindspore.mint.cdists(x1, x2, p=2.0, compute_mode='use_mm_for_euclid_dist_if_necessary')`
 计算两个 Tensor 每对行向量之间的 p-norm 距离。

警告：这是一个实验性 API，后续可能修改或删除。

说明：Ascend 上支持的输入数据类型为 float16 和 float32。

参数：

- **x1** (Tensor) - 输入 Tensor，shape 为 (B, P, M) ， B 表示 0 或者正整数。 B 维度为 0 时该维度被忽略，shape 为 (P, M) 。
- **x2** (Tensor) - 输入 Tensor，shape 为 (B, R, M) ，与 $x1$ 的数据类型一致。
- **p** (float, 可选) - 计算向量对 p-norm 距离的 P 值， $P \geq 0$ 。默认值：2.0。
- **compute_mode** (string, 可选) - 指定计算模式。目前设置此参数无效果。默认值：'use_mm_for_euclid_dist_if_necessary'。

返回：

Tensor，p-范数距离，数据类型与 $x1$ 一致，shape 为 (B, P, R) 。

异常：

- **TypeError** - $x1$ 或 $x2$ 不是 Tensor。
- **TypeError** - $x1$ 或 $x2$ 的数据类型不符合上述“说明”中的要求。
- **TypeError** - p 不是 float32。
- **ValueError** - p 是负数。
- **ValueError** - $x1$ 与 $x2$ 维度不同。
- **ValueError** - $x1$ 与 $x2$ 的维度既不是 2，也不是 3。
- **ValueError** - $x1$ 和 $x2$ 的批次维无法广播。
- **ValueError** - $x1$ 和 $x2$ 的列数不一样。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[[[1.0, 1.0], [2.0, 2.0]]]]).astype(np.float32))
>>> y = Tensor(np.array([[[[3.0, 3.0], [3.0, 3.0]]]]).astype(np.float32))
>>> output = mint.cdists(x, y, 2.0)
>>> print(output)
[[[2.8284273 2.8284273]
  [1.4142137 1.4142137]]]
```


mindspore.mint.cummax

`mindspore.mint.cummax(input, dim)`

返回一个元组 (最值、索引)，其中最值是输入 Tensor *input* 沿维度 *dim* 的累积最大值，索引是每个最大值的索引位置。

$$y_i = \max(x_1, x_2, \dots, x_i)$$

说明：Ascend 不支持 O2 模式。

参数：

- **input** (Tensor) - 输入 Tensor，要求维度大于 0。
- **dim** (int) - 算子操作的维度，维度的大小范围是 $[-input.ndim, input.ndim - 1]$ 。

返回：

一个包含两个 Tensor 的元组，分别表示累积最大值和对应索引。每个输出 Tensor 的形状和输入 Tensor 的形状相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *dim* 不是 int。
- **ValueError** - 如果 *dim* 不在范围 $[-input.ndim, input.ndim - 1]$ 内。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.array([[3, 4, 6, 10], [1, 6, 7, 9], [4, 3, 8, 7], [1, 3, 7, 9]]).astype(np.
    ↳ float32))
>>> output = mint.cummax(x, dim=0)
>>> print(output[0])
[[ 3.  4.  6. 10.]
 [ 3.  6.  7. 10.]
 [ 4.  6.  8. 10.]
 [ 4.  6.  8. 10.]]
>>> print(output[1])
[[0 0 0 0]
 [0 1 1 0]
 [2 1 2 0]
 [2 1 2 0]]
```

mindspore.mint.cummin

`mindspore.mint.cummin(input, dim)`

返回一个元组（最值、索引），其中最值是输入 Tensor *input* 沿维度 *dim* 的累积最小值，索引是每个最小值的索引位置。

$$y_i = \min(x_1, x_2, \dots, x_i)$$

说明：Ascend 不支持 O2 模式。

参数：

- **input** (Tensor) - 输入 Tensor，要求维度大于 0。
- **dim** (int) - 算子操作的维度，维度的大小范围是 $[-input.ndim, input.ndim - 1]$ 。

返回：

一个包含两个 Tensor 的元组，分别表示累积最小值和对应索引。每个输出 Tensor 的形状和输入 Tensor 的形状相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *input* 的 Tensor 类型是复数或 bool。
- **TypeError** - 如果 *dim* 不是 int。
- **ValueError** - 如果 *dim* 不在范围 $[-input.ndim, input.ndim - 1]$ 内。

支持平台：

Ascend

样例：

```
>>> from mindspore import Tensor, mint
>>> import mindspore
>>> a = Tensor([-0.2284, -0.6628, 0.0975, 0.2680, -1.3298, -0.4220], mindspore.float32)
>>> output = mint.cummin(a, dim=0)
>>> print(output[0])
[-0.2284 -0.6628 -0.6628 -0.6628 -1.3298 -1.3298]
>>> print(output[1])
[0 1 1 1 4 4]
```

mindspore.mint.cumsum

`mindspore.mint.cumsum(input, dim, dtype=None)`

计算输入 Tensor *input* 沿轴 *dim* 的累积和。

$$y_i = x_1 + x_2 + x_3 + \dots + x_i$$

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (int) - 累积和计算的轴。

- **dtype** (*mindspore.dtype*, 可选) - 输出的数据类型。如果不为 `None`, 则输入会转化为 *dtype*, 防止数值溢出; 如果为 `None`, 则输出和输入的数据类型一致。默认值: `None`。

返回:

Tensor, 和输入 Tensor 的 shape 相同。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。
- **ValueError** - 如果 *dim* 超出范围。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> x = Tensor(np.array([[3, 4, 6, 10], [1, 6, 7, 9], [4, 3, 8, 7], [1, 3, 7, 9]]).astype(np.
↳float32))
>>> # case 1: along the dim 0
>>> y = mint.cumsum(x, 0)
>>> print(y)
[[ 3.  4.  6. 10.]
 [ 4. 10. 13. 19.]
 [ 8. 13. 21. 26.]
 [ 9. 16. 28. 35.]]
>>> # case 2: along the dim 1
>>> y = mint.cumsum(x, 1)
>>> print(y)
[[ 3.  7. 13. 23.]
 [ 1.  7. 14. 23.]
 [ 4.  7. 15. 22.]
 [ 1.  4. 11. 20.]
```

mindspore.mint.diag

`mindspore.mint.diag(input, diagonal=0)`

如果 *input* 是向量 (1-D 张量), 则返回一个二维张量, 其中 *input* 的元素作为对角线。

如果 *input* 是矩阵 (2-D 张量), 则返回具有 *input* 对角线元素的 1-D 张量。

参数 *diagonal* 控制要考虑的对角线:

- 如果 *diagonal* = 0, 则它是主对角线。
- 如果 *diagonal* > 0, 则它位于主对角线的上方。
- 如果 *diagonal* < 0, 则它位于主对角线下方。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 Tensor。

- **diagonal** (int, 可选) - 输入 Tensor, 默认值为 0。

返回:

Tensor, 具有与输入 Tensor 相同的数据类型, 其 shape 由 *diagonal* 决定:

- 如果输入 *input* 的 shape 为 (x_0) : 输出 shape 为 $(x_0 + |diagonal|, x_0 + |diagonal|)$ 的二维张量。
- 如果输入 *input* 的 shape 为 (x_0, x_1) : 输出 shape 为主对角线上下平移 ($|diagonal|$) 个单位后所剩元素的长度的一维张量。

异常:

- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *input* 的 shape 不是 1-D 和 2-D。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, mint
>>> input = Tensor([1, 2, 3, 4]).astype('int32')
>>> output = mint.diag(input)
>>> print(output)
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
```

mindspore.mint.flatten

`mindspore.mint.flatten(input, start_dim=0, end_dim=-1)`

沿着从 *start_dim* 到 *end_dim* 的维度, 对输入 Tensor 进行展平。

参数:

- **input** (Tensor) - 输入 Tensor。
- **start_dim** (int, 可选) - 要展平的第一个维度。默认值: 0。
- **end_dim** (int, 可选) - 要展平的最后一个维度。默认值: -1。

返回:

Tensor。如果没有维度被展平, 返回原始的输入 *input*, 否则返回展平后的 Tensor。如果 *input* 是零维 Tensor, 将会返回一个一维 Tensor。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *start_dim* 或 *end_dim* 不是 int 类型。
- **ValueError** - 规范化后, *start_dim* 大于 *end_dim*。
- **ValueError** - *start_dim* 或 *end_dim* 不在 $[-input.dim, input.dim-1]$ 范围内。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input_x = Tensor(np.ones(shape=[1, 2, 3, 4]), mindspore.float32)
>>> output = mint.flatten(input_x)
>>> print(output.shape)
(24,)
```

mindspore.mint.flip

`mindspore.mint.flip(input, dims)`

沿指定维度翻转 tensor 中元素。

参数：

- **input** (Tensor) - 输入 tensor。
- **dims** (Union[list[int], tuple[int]]) - 翻转的维度。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> input = mindspore.tensor(mindspore.ops.arange(1, 9).reshape((2, 2, 2)))
>>> output = mindspore.mint.flip(input, (0, 2))
>>> print(output)
[[[6 5]
  [8 7]]
 [[2 1]
  [4 3]]]
```

mindspore.mint.repeat_interleave

`mindspore.mint.repeat_interleave(input, repeats, dim=None, *, output_size=None)`

沿着轴重复 Tensor 的元素，类似 `mindspore.numpy.repeat()`。

警告： 仅 Atlas A2 训练系列产品支持。

参数：

- **input** (Tensor) - 进行重复操作的入参 Tensor，类型必须为 float16, float32, int8, uint8, int16, int32 或者 int64。
- **repeats** (Union[int, tuple, list, Tensor]) - 指定复制次数，为正数。
- **dim** (int, 可选) - 指定复制轴，默认值：None。如果为 None，输入 Tensor 会被展平并且输出结果也会被展平。

关键字参数：

- **output_size** (int, 可选) - 给定轴的总输出大小 (即参数 repeats 元素之和)，默认值：None。

返回：

Tensor, 值沿指定轴复制。如果输入的 shape 为 $(s_1, s_2, \dots, s_n)$, 轴为 i , 则输出的 shape 为 $(s_1, s_2, \dots, s_i * repeats, \dots, s_n)$ 。输出的数据类型与输入相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[0, 1, 2], [3, 4, 5]]), mindspore.int32)
>>> output = mint.repeat_interleave(input, repeats=2, dim=0)
>>> print(output)
[[0 1 2]
 [0 1 2]
 [3 4 5]
 [3 4 5]]
```

mindspore.mint.searchsorted

`mindspore.mint.searchsorted(sorted_sequence, values, *, out_int32=False, right=False, side=None, sorter=None)`

返回元素能够插入输入 tensor 的位置索引，以维持原 tensor 的递增顺序。

参数：

- **sorted_sequence** (Tensor) - 输入 tensor。如果未提供 *sorter*，最内层的维度上须为递增的序列。
- **values** (Tensor) - 要插入元素的值。

关键字参数：

- **out_int32** (bool, 可选) - 输出数据类型是否为 `mindspore.int32`。如果为 `False`，则输出数据类型将为 `mindspore.int64`。默认 `False`。
- **right** (bool, 可选) - 搜索策略。如果为 `True`，则返回找到的最后一个合适的索引；如果为 `False`，则返回第一个合适的索引。默认 `False`。
- **side** (str, 可选) - 跟参数 *right* 功能一致，如果参数值为 `left`，相当于 *right* 为 `False`。如果参数值为 `right`，相当于 *right* 为 `True`。如果值为 `left` 但是 *right* 为 `True` 则报错。默认 `None`。
- **sorter** (Tensor, 可选) - 按 *sorted_sequence* 最内层维度升序排序的索引序列，与未排序的 *sorted_sequence* 共同使用。CPU 和 GPU 只支持 `None`。默认 `None`。

返回：

Tensor

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> sorted_sequence = mindspore.tensor([1, 2, 2, 3, 4, 5, 5])
>>> values = mindspore.tensor([2])
>>> mindspore.mint.searchsorted(sorted_sequence, values)
Tensor(shape=[1], dtype=Int64, value= [1])
```

`mindspore.mint.triangular_solve`

`mindspore.mint.triangular_solve(b, A, upper=True, transpose=False, unitriangular=False)`

求解正上三角形或下三角形可逆矩阵 A 和包含多个元素的右侧边 b 的方程组的解。

用符号表示，它求解方程 $AX = b$ ，并假设矩阵 A 是一个方阵，且为上三角矩阵（如果 `upper = False`，则为下三角矩阵），并且其对角线上没有零元素。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **b** (Tensor) - shape 是 $(*, M, K)$ 的 Tensor，其中 $*$ 表示任意数量的维度。
- **A** (Tensor) - shape 是 $(*, M, M)$ 的 Tensor，其中 $*$ 表示任意数量的维度。
- **upper** (bool, 可选) - 矩阵 A 是为上三角矩阵或下三角矩阵。默认值：True。
- **transpose** (bool, 可选) - 求解方程 $op(A)X = b$ ，其中如果此标志为 True，则 $op(A) = A^T$ ；如果为 False，则 $op(A) = A$ 。默认值：False。
- **unitriangular** (bool, 可选) - 矩阵 A 是否为单位三角矩阵。如果为 True，则假设矩阵 A 的对角线元素为 1，并且不会从矩阵 A 中引用这些对角线元素。默认值：False。

返回：

包含 X 和 A 的 tuple。

异常：

- **TypeError** - 如果参数 b 不是 Tensor。
- **TypeError** - 如果参数 A 不是 Tensor。
- **TypeError** - 如果 `upper` 不是 bool。
- **TypeError** - 如果 `transpose` 不是 bool。
- **TypeError** - 如果 `unitriangular` 不是 bool。
- **ValueError** - 如果 b 或者 A 的维度不在 $[2, 6]$ 范围内。
- **ValueError** - 如果 b 和 A 的 shape 不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import mint
>>> from mindspore import Tensor
>>> b = Tensor(np.ones((2, 3, 4), dtype=np.float32))
>>> A = Tensor(np.ones((2, 3, 3), dtype=np.float32))
>>> output = mint.triangular_solve(b, A)
>>> print(output[0])
[[[ 0.  0.  0.  0.]
  [ 0.  0.  0.  0.]
  [ 1.  1.  1.  1.]]
 [[ 0.  0.  0.  0.]
```

(下页继续)

(续上页)

```
[ 0.  0.  0.  0.]
[ 1.  1.  1.  1.]]]
```

4.4 mindspore.mint.nn

4.4.1 卷积层

接口名	概述	支持平台
<code>mindspore.mint.nn.Conv2d</code>	二维卷积层。	Ascend
<code>mindspore.mint.nn.Conv3d</code>	三维卷积层。	Ascend
<code>mindspore.mint.nn.ConvTranspose2d</code>	将 2D 转置卷积运算应用于由多个输入平面组成的输入图像。	Ascend
<code>mindspore.mint.nn.Fold</code>	将提取出的滑动局部区域块还原成更大的输出 Tensor。	Ascend
<code>mindspore.mint.nn.Unfold</code>	从一个 batch 的输入 Tensor 中提取滑动局部块。	Ascend

mindspore.mint.nn.Conv2d

class mindspore.mint.nn.**Conv2d**(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', dtype=None*)

二维卷积层。

对输入 Tensor 计算二维卷积，通常输入的 shape 为 $(N, C_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C 为通道数， H 为特征图的高度， W 为特征图的宽度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中， bias 为输出偏置， ccor 为 [cross-correlation](#) 操作， weight 为卷积核的值， X 为输入的特征图。

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中， $\text{bias}(C_{out_j})$ 为第 j 个输出通道的偏置， $\text{weight}(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。卷积核 shape 为 $(\text{kernel\_size}[0], \text{kernel\_size}[1])$ ，其中 $\text{kernel\_size}[0]$ 和 $\text{kernel\_size}[1]$ 是卷积核的高度和宽度。若考虑到输入输出通道以及 groups ，则完整卷积核的 shape 为 $(C_{out}, C_{in}/\text{groups}, \text{kernel\_size}[0], \text{kernel\_size}[1])$ ，其中 groups 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

参数：

- in_channels** (int) - Conv2d 层输入 Tensor 的空间维度。
- out_channels** (int) - Conv2d 层输出 Tensor 的空间维度。
- kernel_size** (Union[int, tuple[int], list[int]]) - 指定二维卷积核的高度和宽度。数据类型为 int 或两个 int 的 tuple。一个 int 表示卷积核的高度和宽度均为该值。两个 int 的 tuple 分别表示卷积核的高度和宽度。

- **stride** (Union[int, tuple[int], list[int]], 可选) - 二维卷积核的移动步长。数据类型为 int 或者长度为 2 的 int 的 tuple。一个 int 表示在高度和宽度方向的移动步长均为该值。两个 int 的 tuple 分别表示在高度和宽度方向的移动步长。默认值: 1。
- **padding** (Union[int, tuple[int], list[int], str], 可选) - 输入的高度和宽度方向上填充的数量。数据类型为 int 或包含 2 个 int 的 tuple 或 string { "valid", "same" }。如果 padding 是一个 int, 那么 $padding_{\{H\}}$ 和 $padding_{\{W\}}$ 的填充都等于 padding。如果 padding 是一个有 2 个 int 的 tuple, 那么 $padding_{\{H\}}$ 和 $padding_{\{W\}}$ 的填充分别等于 $padding[0]$ 和 $padding[1]$ 。值应该要大于等于 0。默认值: 0。
 - "same": 在输入的四周填充, 使得当 stride 为 1 时, 输入和输出的 shape 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在底部/右侧。若设置该模式, stride 的值必须为 1。
 - "valid": 不对输入进行填充, 返回输出可能的最大高度和宽度, 不能构成一个完整 stride 的额外的像素将被丢弃。
- **padding_mode** (str, 可选) - 指定填充模式, 填充值为 0。可选值为 "zeros", "reflect", "circular" 或 "replicate"。默认值: "zeros"。
- **dilation** (Union[int, tuple[int], list[int]], 可选) - 卷积核膨胀尺寸。可以为单个 int, 或者由两个/四个 int 组成的 tuple。单个 int 表示在高度和宽度方向的膨胀尺寸均为该值。两个 int 组成的 tuple 分别表示在高度和宽度方向的膨胀尺寸。若为四个 int, N、C 两维度 int 默认为 1, H、W 两维度分别对应高度和宽度上的膨胀尺寸。假设 $dilation = (d0, d1)$, 则卷积核在高度方向间隔 $d0 - 1$ 个元素进行采样, 在宽度方向间隔 $d1 - 1$ 个元素进行采样。高度和宽度上取值范围分别为 $[1, H]$ 和 $[1, W]$ 。默认值: 1。
- **groups** (int, 可选) - 将过滤器拆分为组, $in\_channels$ 和 $out\_channels$ 必须可被 groups 整除。如果组数等于 $in\_channels$ 和 $out\_channels$, 这个二维卷积层也被称为二维深度卷积层。默认值: 1。需要满足以下约束:
 - $(C_{in} \% groups == 0)$
 - $(C_{out} \% groups == 0)$
 - $(C_{out} \geq groups)$
 - $(kernel\_size[1] = C_{in}/groups)$
- **bias** (bool, 可选) - Conv2d 层是否添加偏置参数。默认值: True。
- **dtype** (mindspore.dtype, 可选) - Parameters 的 dtype。默认值: None, 使用 mstype.float32。

输入:

- **x** (Tensor) - Shape 为 $(N, C_{in}, H_{in}, W_{in})$ 或者 (C_{in}, H_{in}, W_{in}) 的 Tensor。

输出:

Tensor, shape 为 $(N, C_{out}, H_{out}, W_{out})$ 或者 $(C_{out}, H_{out}, W_{out})$ 。

padding 为 "same" 时:

$$\begin{aligned} H_{out} &= \left\lceil \frac{H_{in}}{stride[0]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in}}{stride[1]} \right\rceil \end{aligned}$$

padding 为 "valid" 时:

$$\begin{aligned} H_{out} &= \left\lceil \frac{H_{in} - dilation[0] \times (kernel\_size[0] - 1)}{stride[0]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} - dilation[1] \times (kernel\_size[1] - 1)}{stride[1]} \right\rceil \end{aligned}$$

padding 为 int 或 tuple 时:

$$\begin{aligned} H_{out} &= \left\lfloor \frac{H_{in} + padding[0] + padding[1] - (kernel\_size[0] - 1) \times dilation[0] - 1}{stride[0]} + 1 \right\rfloor \\ W_{out} &= \left\lfloor \frac{W_{in} + padding[2] + padding[3] - (kernel\_size[1] - 1) \times dilation[1] - 1}{stride[1]} + 1 \right\rfloor \end{aligned}$$

异常:

- **ValueError** - 输入特征图的大小与参数应满足输出公式, 以确保输出特征图大小为正, 否则会报错。
- **RuntimeError** - Ascend 上受不同型号 NPU 芯片上 L1 缓存大小限制, 用例尺寸或 Kernel Size 过大。
- **TypeError** - 如果 `in_channels`, `out_channels` 或者 `groups` 不是 int。
- **TypeError** - 如果 `kernel_size`, `stride`, 或者 `dilation` 既不是 int 也不是 tuple。
- **ValueError** - 如果 `in_channels`, `out_channels`, `kernel_size`, `stride` 或者 `dilation` 小于 1。
- **ValueError** - 如果 `padding` 小于 0。
- **ValueError** - 如果 `padding` 是 "same", `stride` 不等于 1。
- **ValueError** - 输入参数不满足卷积输出公式。
- **ValueError** - `kernel_size` 不能超过输入特征图的大小。
- **ValueError** - `padding` 值不能导致计算区域超出输入大小。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> net = mint.nn.Conv2d(120, 240, 4, bias=False)
>>> x = Tensor(np.ones([1, 120, 1024, 640]), mindspore.float32)
>>> output = net(x).shape
>>> print(output)
(1, 240, 1021, 637)
```

mindspore.mint.nn.Conv3d

class mindspore.mint.nn.Conv3d(*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', dtype=None*)

三维卷积层。

对输入 Tensor 计算三维卷积。通常, 输入 Tensor 的 shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$, 其中 N 为 batch size, C 为通道数, D, H, W 分别为特征图的深度、高度和宽度。

根据以下公式计算输出:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} ccor(weight(C_{out_j}, k), X(N_i, k))$$

其中, $bias$ 为输出偏置, $ccor$ 为 cross-correlation 操作, $weight$ 为卷积核的值, X 为输入的特征图。

- i 对应 batch 数, 其范围为 $[0, N - 1]$, 其中 N 为输入 batch。

- j 对应输出通道, 其范围为 $[0, C_{out} - 1]$, 其中 C_{out} 为输出通道数, 该值也等于卷积核的个数。
- k 对应输入通道数, 其范围为 $[0, C_{in} - 1]$, 其中 C_{in} 为输入通道数, 该值也等于卷积核的通道数。

因此, 上面的公式中, $bias(C_{out_j})$ 为第 j 个输出通道的偏置, $weight(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片, $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。

卷积核 shape 为 $(kernel\_size[0], kernel\_size[1], kernel\_size[2])$, 其中 $kernel\_size[0]$ 、 $kernel\_size[1]$ 和 $kernel\_size[2]$ 分别是卷积核的深度、高度和宽度。若考虑到输入输出通道以及 $groups$, 则完整卷积核的 shape 为 $(C_{out}, C_{in}/groups, kernel\_size[0], kernel\_size[1], kernel\_size[2])$, 其中 $groups$ 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层, 请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

参数的约束细节, 请参考 `mindspore.mint.nn.functional.conv3d()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **in_channels** (int) - Conv3d 层输入 Tensor 的空间维度。
- **out_channels** (int) - Conv3d 层输出 Tensor 的空间维度。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 指定三维卷积核的高度和宽度。数据类型是 int 或两个 int 的 tuple。int 表示卷积核的高度和宽度。两个 int 的 tuple 分别表示卷积核的高度和宽度。
- **stride** (Union[int, tuple[int], list[int]], 可选) - 三维卷积核的移动步长。数据类型为 int 或三个 int 的 tuple。若为一个 int, 则表示在深度、高度和宽度方向的移动步长均为该 int 值; 若为三个 int 的 tuple, 则分别表示在深度、高度和宽度方向的移动步长。默认值: 1。
- **padding** (Union[int, tuple[int], list[int], str], 可选) - 输入的深度、高度和宽度方向的填充数。数据类型是 int 或 string{"valid", "same"} 或三个 int 的 tuple。该值应大于或等于 0。默认值: 0。
 - "same": 在输入的边缘加上衬垫, 这样当 *stride* 设置为 "1" 时, 输入和输出的形状是相同的。填充量由运算符内部计算。如果填充量是偶数, 则均匀分布在输入周围; 如果填充量为奇数, 则多余的填充量会流向右侧/底部。如果设置了此模式, 则 *padding* 必须为 0。
 - "valid": 输入没有填充, 输出返回最大可能的高度和宽度。无法完成整个步幅的额外像素将被丢弃。如果设置了此模式, 则 *padding* 必须为 0。
- **padding_mode** (str, 可选) - 使用填充值 0 指定填充模式。它可以设置为: "zeros"、"reflect"、"circular" 或 "replicate"。默认值: "zeros"。
- **dilation** (Union[int, tuple[int], list[int]], 可选) - 控制内核点之间的空间。默认值: 1。
- **groups** (int, 可选) - 拆分过滤成组, *in_channel* 和 *out_channels* 必须能被 *groups* 整除。如果组等于 *in_channels* 和 *out_channels*。默认值: 1。
- **bias** (bool, 可选) - Conv3d 层是否具有偏置参数。默认值: True。
- **dtype** (`mindspore.dtype`, 可选) - Parameters 的 dtype。默认值: None, 使用 `mstype.float32`。

输入:

- **x** (Tensor) - shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ 或 $(C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor。

输出:

Tensor, shape 为 $(N, C_{out}, D_{out}, H_{out}, W_{out})$ 或 $(C_{out}, D_{out}, H_{out}, W_{out})$ 。

padding 为 "same" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in}}{\text{stride}[0]} \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in}}{\text{stride}[1]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in}}{\text{stride}[2]} \right\rceil \end{aligned}$$

padding 为 "valid" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in} - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1)}{\text{stride}[0]} + 1 \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in} - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1)}{\text{stride}[1]} + 1 \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} - \text{dilation}[2] \times (\text{kernel\_size}[2] - 1)}{\text{stride}[2]} + 1 \right\rceil \end{aligned}$$

异常:

- **TypeError** - *in_channels*、*out_channels* 或 *groups* 不是 int。
- **TypeError** - *kernel_size*、*stride*、*padding* 或 *dilation* 既不是 int 也不是 tuple。
- **ValueError** - *in_channels*、*out_channels*、*kernel_size*、*stride* 或 *dilation* 小于 1。
- **ValueError** - *padding* 小于 0。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> net = mint.nn.Conv3d(120, 10, 4)
>>> x = Tensor(np.ones([1, 120, 10, 23, 34]), mindspore.float32)
>>> output = net(x).shape
>>> print(output)
(1, 10, 7, 20, 31)
```

mindspore.mint.nn.ConvTranspose2d

```
class mindspore.mint.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
                                         output_padding=0, groups=1, bias=True, dilation=1,
                                         padding_mode='zeros', dtype=None)
```

将 2D 转置卷积运算应用于由多个输入平面组成的输入图像。

这个模块可以看作是 **Conv2d** 相对于其输入的梯度。它也被称为分数跨步卷积或反卷积（尽管它不是实际的反卷积操作，其不是计算卷积的真逆函数）。

所有参数中的 *kernel_size*、*stride*、*padding*、*output_padding* 可以为:

- 单个整数-在这种情况下，该值同时被用于 H 和 W 维度
- 一个由两个整数组成的 tuple-在这种情况下，第一个整数用于 H 维度，第二个整数用于 W 维度。

警告:

- 这是一个实验性 API，后续可能修改或删除。
- 在输入非连续场景下，`output_padding` 必须小于 `stride`。
- 在 Atlas 训练系列产品上，float32 类型输入时，仅支持 `groups` 为 1。

参数:

- **in_channels** (int) - 输入图像中的通道数。
- **out_channels** (int) - 卷积生成的通道数。
- **kernel_size** (Union[int, tuple(int)]) - 卷积核的大小。
- **stride** (Union[int, tuple(int)], 可选) - 卷积的步长。默认值: 1。
- **padding** (Union[int, tuple(int)], 可选) - $dilation * (kernel\_size - 1) - padding$ 零填充将添加到输入中每个维度的两侧。默认值: 0。
- **output_padding** (Union[int, tuple(int)], 可选) - 在输出形状中每个维度的一侧增加额外的尺寸。`output_padding` 的值必须小于 `stride` 或 `dilation`。默认值: 0。
- **groups** (int, 可选) - 从输入通道到输出通道的分块数。默认值: 1。
- **bias** (bool, 可选) - 如果值为 True，则添加一个可学习的偏置至输出中。默认值: True。
- **dilation** (Union[int, tuple(int)], 可选) - 内核元素之间的间距。默认值: 1。
- **padding_mode** (str, 可选) - 指定填充值的填充模式。目前仅支持 zeros。默认值: zeros。
- **dtype** (mindspore.dtype, 可选) - 模型参数的类型。默认值: None，此时模型参数类型为 `mstype.float32`。

可学习参数:

- **weigh** (Parameter) - 模型的可学习权重，其 shape 为 $(in\_channels, \frac{out\_channels}{groups}, kernel\_size[0], kernel\_size[1])$ 。其值从分布 $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ 中采样得到，其中 $k = \frac{groups}{C_{out} * \prod_{i=0}^1 kernel\_size[i]}$ 。
- **bias** (Parameter) - 模型的可学习偏置，其 shape 为 $(out\_channels,)$ 。如果 `bias` 为 True，则其值从分布 $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ 中采样得到，其中 $k = \frac{groups}{C_{out} * \prod_{i=0}^1 kernel\_size[i]}$ 。

输入:

- **input** (Tensor) - Tensor，其 shape 为 $(N, C_{in}, H_{in}, W_{in})$ 或 (C_{in}, H_{in}, W_{in}) 。

输出:

Tensor，shape 为 $(N, C_{out}, H_{out}, W_{out})$ 或 $(C_{out}, H_{out}, W_{out})$ ，其中:

$$H_{out} = (H_{in} - 1) \times stride[0] - 2 \times padding[0] + dilation[0] \times (kernel\_size[0] - 1) + output\_padding[0] + 1$$

$$W_{out} = (W_{in} - 1) \times stride[1] - 2 \times padding[1] + dilation[1] \times (kernel\_size[1] - 1) + output\_padding[1] + 1$$

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> # With square kernels and equal stride
>>> m = mint.nn.ConvTranspose2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = mint.nn.ConvTranspose2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> input = mint.randn(20, 16, 50, 100)
>>> output = m(input)
>>> # exact output size can be also specified as an argument
>>> input = mint.randn(1, 16, 12, 12)
>>> downsample = mint.nn.Conv2d(16, 16, 3, stride=2, padding=1)
>>> upsample = mint.nn.ConvTranspose2d(16, 16, 3, stride=2, padding=1)
>>> h = downsample(input)
>>> h.shape
(1, 16, 6, 6)
>>> output = upsample(h, output_size=input.shape)
>>> output.shape
(1, 16, 12, 12)
```

mindspore.mint.nn.Fold

class mindspore.mint.nn.Fold(output_size, kernel_size, dilation=1, padding=0, stride=1)

将提取出的滑动局部区域块还原成更大的输出 Tensor。

更多参考详见[mindspore.mint.nn.functional.fold\(\)](#)。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> fold = mint.nn.Fold([8, 8], [2, 2], [2, 2], [2, 2], [2, 2])
>>> input = Tensor(input_data=np.random.rand(16, 64, 25), dtype=mstype.float32)
>>> output = fold(input)
>>> print(output.shape)
(16, 16, 8, 8)
```

mindspore.mint.nn.Unfold

class mindspore.mint.nn.Unfold(kernel_size, dilation=1, padding=0, stride=1)

从一个 batch 的输入 Tensor 中提取滑动局部块。

更多参考详见[mindspore.mint.nn.functional.unfold\(\)](#)。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
```

(下页继续)

(续上页)

```
>>> input = Tensor(np.random.rand(4, 4, 32, 32), mindspore.float64)
>>> unfold = mint.nn.Unfold(kernel_size=3, dilation=1, stride=1)
>>> output = unfold(input)
>>> print(output.shape)
(4, 36, 900)
```

4.4.2 归一化层

接口名	概述	支持平台
<code>mindspore.mint.nn.BatchNorm1d</code>	在二维或三维输入上应用批归一化。	Ascend
<code>mindspore.mint.nn.BatchNorm2d</code>	在四维输入上应用批归一化。	Ascend
<code>mindspore.mint.nn.BatchNorm3d</code>	在五维输入上应用批归一化。	Ascend
<code>mindspore.mint.nn.GroupNorm</code>	在 mini-batch 输入上进行组归一化。	Ascend
<code>mindspore.mint.nn.LayerNorm</code>	在 mini-batch 输入上应用层归一化 (Layer Normalization)。	Ascend
<code>mindspore.mint.nn.SyncBatchNorm</code>	在 N 维输入上进行跨设备同步批归一化 (Batch Normalization, BN)。	Ascend

mindspore.mint.nn.BatchNorm1d

`class mindspore.mint.nn.BatchNorm1d (num_features: int, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True, dtype=None)`

在二维或三维输入上应用批归一化。具体内容见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

在 mini-batch 上按维度计算平均值和标准方差。 γ 和 β 是一个大小为 C 的可学习参数向量。默认情况下, γ 中的元素被赋值为 1。 β 的元素则被赋值为 0。

警告：该接口不支持动态 Rank。这是一个实验性 API，后续可能修改或删除。

参数：

- num_features** (int) - 输入的 shape (N, C, L) 中的 C 值。
- eps** (float, 可选) - ϵ 加在分母上的值，以确保数值稳定。默认值：1e-5。
- momentum** (float, 可选) - 动态均值和动态方差所使用的动量。当想计算累计平均值时，可设置为 None。默认值：0.1。
- affine** (bool, 可选) - bool 类型。设置为 True 时，可学习到反射参数值。默认值：True。
- track_running_stats** (bool, 可选) - bool 类型。设置为 True 时，会跟踪运行时的均值和方差。当设置为 False 时，则不会跟踪这些统计信息。且在 tran 和 eval 模式下，该 cell 总是使用 batch 的统计信息。
- dtype** (`mindspore.dtype`, 可选) - Parameters 的 dtype。默认值：None。

输入：

- input** (Tensor) - 输入 shape 为 (N, C) 或 (N, C, L) 的 Tensor，其中 N 为 batch， C 为特征数量或通道数量， L 为序列长度。

输出：

Tensor, 类型和 shape 与 *input* 一致。

异常：

- **TypeError** - *num_features* 不是整数。
- **TypeError** - *eps* 不是浮点数。
- **ValueError** - *num_features* 小于 1。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input_x = mindspore.Tensor([[0.7, 0.5, 0.5, 0.6], [0.5, 0.4, 0.6, 0.9]])
>>> net = mint.nn.BatchNorm1d(4)
>>> output = net(input_x)
>>> print(output)
[[ 0.99950075  0.9980011 -0.9980068 -0.9997783]
 [-0.9995012 -0.99799967  0.9980068  0.9997778]]
```

mindspore.mint.nn.BatchNorm2d

class mindspore.mint.nn.BatchNorm2d (*num_features: int, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True, dtype=None*)

在四维输入上应用批归一化。具体内容见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

在 mini-batch 上按维度计算平均值和标准方差。 γ 和 β 是一个大小为 C 的可学习参数向量。默认情况下, γ 中的元素被赋值为 1。 β 的元素则被赋值为 0。

警告：

- 该接口不支持动态 Rank。
- 这是一个实验性 API, 后续可能修改或删除。

参数：

- **num_features** (int) - 输入的 shape (N, C, H, W) 中的 C 值。
- **eps** (float, 可选) - ϵ 加在分母上的值, 以确保数值稳定。默认值: $1e-5$ 。
- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。当想计算累计平均值时, 可设置为 `None`。默认值: 0.1 。
- **affine** (bool, 可选) - bool 类型。设置为 `True` 时, 可学习到反射参数值。默认值: `True`。
- **track_running_stats** (bool, 可选) - bool 类型。设置为 `True` 时, 会跟踪运行时的均值和方差。当设置为 `False` 时, 则不会跟踪这些统计信息。且在 `tran` 和 `eval` 模式下, 该 cell 总是使用 `batch` 的统计信息。默认值: `True`。

- **dtype** (*mindspore.dtype*, 可选) - Parameters 的 dtype。默认值: None。

输入:

- **input** (Tensor) - 输入 shape 为 (N, C, H, W) 。

输出:

Tensor, 类型和 shape 与 *input* 一致。

异常:

- **TypeError** - *num_features* 不是整数。
- **TypeError** - *eps* 不是浮点数。
- **ValueError** - *num_features* 小于 1。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input_x = mindspore.Tensor([0.3, 0.4, 0.5, 0.3])
>>> input_x = input_x.reshape((2, 2, 1, 1))
>>> net = mint.nn.BatchNorm2d(2)
>>> output = net(input_x)
>>> print(output)
[[[-0.99950075]]
 [[0.9980087]]]
[[[0.999501]]
 [[-0.9980097]]]
```

mindspore.mint.nn.BatchNorm3d

class mindspore.mint.nn.BatchNorm3d (*num_features: int, eps=1e-5, momentum=0.1, affine=True, track_running_stats=True, dtype=None*)

在五维输入上应用批归一化。具体内容见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

在 mini-batch 上按维度计算平均值和标准方差。 γ 和 β 是一个大小为 C 的可学习参数向量。默认情况下, γ 中的元素被赋值为 1。 β 的元素则被赋值为 0。

警告: 该接口不支持动态 Rank。这是一个实验性 API, 后续可能修改或删除。

参数:

- **num_features** (int) - 输入的 shape (N, C, D, H, W) 中的 C 值。
- **eps** (float, 可选) - ϵ 加在分母上的值, 以确保数值稳定。默认值: $1e-5$ 。
- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。当想计算累计平均值时, 可设置为 None。默认值: 0.1。

- **affine** (bool, 可选) - bool 类型。设置为 True 时，可学习到反射参数值。默认值：True。
- **track_running_stats** (bool, 可选) - bool 类型。设置为 True 时，会跟踪运行时的均值和方差。当设置为 False 时，则不会跟踪这些统计信息。且在 tran 和 eval 模式下，该 cell 总是使用 batch 的统计信息。
- **dtype** (*mindspore.dtype*, 可选) - Parameters 的 dtype。默认值：None。

输入：

- **input** (Tensor) - 输入 shape 为 (N, C, D, H, W) 。

输出：

Tensor，类型和 shape 与 *input* 一致。

异常：

- **TypeError** - *num_features* 不是整数。
- **TypeError** - *eps* 不是浮点数。
- **ValueError** - *num_features* 小于 1。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input_x = mindspore.Tensor([0.1, 0.9, 1.2, 2.3])
>>> input_x = input_x.reshape((1, 2, 1, 1, 2))
>>> net = mint.nn.BatchNorm3d(2)
>>> output = net(input_x)
>>> print(output)
[[[[-0.9999688 0.99996865]]]
 [[[-0.9999833 06.9999831]]]]]
```

mindspore.mint.nn.GroupNorm

class mindspore.mint.nn.GroupNorm (*num_groups, num_channels, eps=1e-05, affine=True, dtype=None*)

在 mini-batch 输入上进行组归一化。

Group Normalization 被广泛用于递归神经网络中。适用单个训练用例的 mini-batch 输入归一化，详见论文 [Group Normalization](#)。

Group Normalization 把通道划分为组，然后计算每一组之内的均值和方差，以进行归一化。其中 γ 是通过训练学习出的 scale 值， β 是通过训练学习出的 shift 值。

公式如下，

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中， γ 为 *weight*， β 为 *bias*， ϵ 为 *eps*。

参数：

- **num_groups** (int) - 沿通道维度待划分的组数。
- **num_channels** (int) - 输入的通道维度数。

- **eps** (float, 可选) - 添加到分母中的值, 以确保数值稳定。默认值: $1e-05$ 。
- **affine** (bool, 可选) - 当被设置为 `True` 时, 参数 γ 和 β 是可学习的。默认值: `True`。
- **dtype** (`mindspore.dtype`, 可选) - `Parameter` 的类型。默认值: `None`。

输入:

- **input** (Tensor) - 输入特征的 shape 为 $(N, C, *)$, 其中 $*$ 任意数量的附加维度。

输出:

Tensor, 被标准化和缩放偏移后的 Tensor, 具有与 *input* 相同的 shape 和数据类型。

异常:

- **TypeError** - `num_groups` 或 `num_channels` 不是 int。
- **TypeError** - `eps` 不是 float。
- **TypeError** - `affine` 不是 bool。
- **ValueError** - `num_groups` 或 `num_channels` 小于 1。
- **ValueError** - `num_channels` 未被 `num_groups` 整除。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> group_norm_op = ms.mint.nn.GroupNorm(2, 2)
>>> x = ms.Tensor(np.ones([1, 2, 4, 4], np.float32))
>>> output = group_norm_op(x)
>>> print(output)
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]]]
```

mindspore.mint.nn.LayerNorm

class `mindspore.mint.nn.LayerNorm` (*normalized_shape*, *eps*= $1e-5$, *elementwise_affine*=`True`, *bias*=`True`, *dtype*=`None`)
在 mini-batch 输入上应用层归一化 (Layer Normalization)。

层归一化在递归神经网络中被广泛的应用。适用单个训练用例的 mini-batch 输入上应用归一化, 详见论文 [Layer Normalization](#)。

与批归一化 (Batch Normalization) 不同, 层归一化在训练和测试时执行完全相同的计算。应用于所有通道和像素, 即使 `batch_size=1` 也适用。其中 γ 是通过训练学习出的 scale 值, β 是通过训练学习出的 shift 值。公式如下:

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **normalized_shape** (Union(tuple[int], list[int], int)) - 对 x 执行归一化的 shape。
- **eps** (float, 可选) - 添加到分母中的值 (ϵ)，以确保数值稳定。默认值： $1e-5$ 。
- **elementwise_affine** (bool, 可选) - 是否需要仿射变换。当被设置为 True 时，权重参数初始化为 1，偏差初始化为 0。默认值：True。
- **bias** (bool, 可选) - 当被设置为 False 时，不会学习结果性偏差（仅 *elementwise_affine* 值为 True 时生效）。默认值：True。
- **dtype** (*mindspore.dtype*, 可选) - Parameters 的 dtype。默认值：None。

输入：

- **x** (Tensor) - x 的 shape 为 $(N, *)$ ，* 等于 *normalized_shape*。

输出：

Tensor，归一化后的 Tensor，shape 和数据类型与 x 相同。

异常：

- **TypeError** - *eps* 不是 float。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> x = ms.Tensor(np.ones([20, 5, 10, 10]), ms.float32)
>>> shape1 = x.shape[1:]
>>> m = ms.mint.nn.LayerNorm(shape1)
>>> output = m(x).shape
>>> print(output)
(20, 5, 10, 10)
```

mindspore.mint.nn.SyncBatchNorm

class mindspore.mint.nn.SyncBatchNorm (num_features: int, eps: float = 1e-5, momentum: float = 0.1, affine: bool = True, track_running_stats: bool = True, process_group: Optional[str] = None, dtype=None)

在 N 维输入上进行跨设备同步批归一化 (Batch Normalization, BN)。

同步 BN 是跨设备的。BN 的实现仅对每个设备中的数据进行归一化。同步 BN 将归一化组内的输入。描述见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。使用 mini-batch 数据和和学习参数进行训练，参数见如下公式。

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **num_features** (int) - 指定输入 Tensor 的通道数量，输入 Tensor 的 size 为 $(N, C, +)$ 。
- **eps** (float, 可选) - ϵ 添加到分母中的值，以确保数值稳定。默认值： $1e-5$ 。
- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。默认值：0.1。
- **affine** (bool, 可选) - bool 类型。设置为 True 时， γ 和 β 为可学习参数。设置为 False 时， γ 和 β 为不可学习参数。默认值：True。
- **track_running_stats** (bool, 可选) - bool 类型。设置为 True 时，会跟踪运行时的均值和方差。当设置为 False 时，则不会跟踪这些统计信息。且在 tran 和 eval 模式下，该 cell 总是使用 batch 的统计信息。默认值：True。
- **process_group** (str, 可选) - 统计数据的同步在每个进程组内单独进行。默认行为是全局同步。默认值：None。
- **dtype** (*mindspore.dtype*, 可选) - Parameters 的 dtype。默认值：None。

输入：

- **x** (Tensor) - shape 为 $(N, C_{in}, +)$ 的 Tensor。

输出：

Tensor，归一化后的 Tensor，shape 为 $(N, C_{out}, +)$ 。

异常：

- **TypeError** - *num_features* 不是 int。
- **TypeError** - *eps* 不是 float。
- **ValueError** - *num_features* 小于 1。
- **ValueError** - *momentum* 不在范围 $[0, 1]$ 内。
- **ValueError** - *process_group* 中的 rank ID 不在 $[0, \text{rank\_size})$ 范围内。

支持平台：

Ascend

样例：

说明：在运行以下示例之前，您需要配置通信环境变量。

对于 Ascend 设备，用户需要准备 rank table，设置 rank_id 和 device_id。这里，示例使用 msrun 通过单个命令跨节点拉取多进程分布式任务行指令。请参阅 [Ascend 教程](#) 了解更多详情。

此示例应在多个设备上运行。

```
>>> # Firstly, preparing test_syncbn.py:
>>> import numpy as np
>>> import mindspore
>>> import mindspore.context as context
>>> from mindspore.mint.nn.layer import SyncBatchNorm
>>> from mindspore import Tensor
```

(下页继续)

(续上页)

```
>>> from mindspore.communication import init, create_group, get_local_rank
>>> init()
>>> context.set_context(mode=context.PYNATIVE_MODE, device_target="Ascend")
>>> group = "0-1"
>>> rank_ids = [0, 1]
>>> create_group(group, rank_ids)
>>> sync_batch_norm = SyncBatchNorm(num_features=2, process_group=group, dtype=mindspore.
↪float32)
>>> sync_batch_norm.set_train(False)
>>> input_x = Tensor(np.linspace(0, 5, 2*2*2*2), mindspore.float32).reshape(2, 2, 2, 2)
>>> output_data = sync_batch_norm(input_x)
>>> # Then, executing the command such as the following:
>>> # msrun --worker_num=2 --local_worker_num=2 --master_port=8975 --log_dir=msrun_log --
↪join=True
>>> # --cluster_time_out=100 pytest -s -v test_syncbn.py
```

4.4.3 非线性激活层 (加权和, 非线性)

接口名	概述	支持平台
<code>mindspore.mint.nn.ELU</code>	指数线性单元激活函数 (Exponential Linear Unit activation function)。	Ascend
<code>mindspore.mint.nn.GELU</code>	高斯误差线性单元激活函数 (Gaussian Error Linear Unit)。	Ascend
<code>mindspore.mint.nn.GLU</code>	计算输入 Tensor 的门线性单元激活函数 (Gated Linear Unit activation function) 值。	Ascend
<code>mindspore.mint.nn.Hardshrink</code>	逐元素计算 Hard Shrink 激活函数。	Ascend
<code>mindspore.mint.nn.Hardsigmoid</code>	逐元素计算 Hard Sigmoid 激活函数。	Ascend
<code>mindspore.mint.nn.Hardswish</code>	逐元素计算 Hard Swish 激活函数。	Ascend
<code>mindspore.mint.nn.LogSigmoid</code>	逐元素计算 LogSigmoid 激活函数。	Ascend
<code>mindspore.mint.nn.LogSoftmax</code>	在指定轴上对输入 Tensor 应用 LogSoftmax 函数。	Ascend
<code>mindspore.mint.nn.Mish</code>	逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。	Ascend
<code>mindspore.mint.nn.PReLU</code>	逐元素计算 PReLU (PReLU Activation Operator) 激活函数。	Ascend
<code>mindspore.mint.nn.ReLU</code>	逐元素计算 ReLU (Rectified Linear Unit activation function) 修正线性单元激活函数。	Ascend
<code>mindspore.mint.nn.ReLU6</code>	ReLU6 激活函数。	Ascend
<code>mindspore.mint.nn.SELU</code>	逐元素计算激活函数 SELU (Scaled exponential Linear Unit)。	Ascend
<code>mindspore.mint.nn.SiLU</code>	逐元素计算 SiLU 激活函数。	Ascend
<code>mindspore.mint.nn.Sigmoid</code>	逐元素计算 Sigmoid 激活函数。	Ascend
<code>mindspore.mint.nn.Softmax</code>	将 Softmax 函数应用于 n 维输入 Tensor。	Ascend
<code>mindspore.mint.nn.Softshrink</code>	逐元素计算 Soft Shrink 激活函数。	Ascend
<code>mindspore.mint.nn.Tanh</code>	逐元素计算 Tanh (双曲正切值) 激活函数, 返回一个新的 Tensor。	Ascend

mindspore.mint.nn.ELU

class mindspore.mint.nn.ELU (*alpha=1.0, inplace=False*)

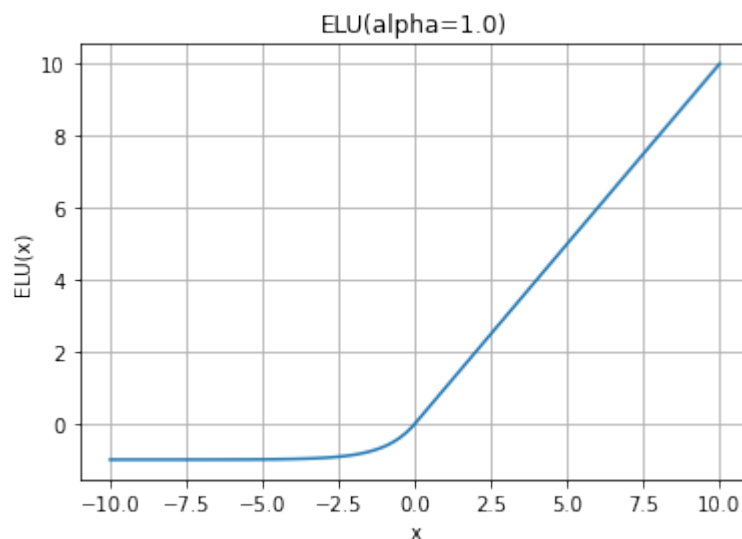
指数线性单元激活函数 (Exponential Linear Unit activation function)。

对输入的每个元素计算 ELU。该激活函数定义如下：

$$ELU_i = \begin{cases} x_i, & \text{if } x_i \geq 0; \\ \alpha * (\exp(x_i) - 1), & \text{otherwise.} \end{cases}$$

其中, x_i 表示输入的元素, α 表示 *alpha* 参数, *alpha* 决定 ELU 的平滑度。

ELU 函数图：



警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **alpha** (float, 可选) - ELU 的 alpha 值，数据类型为 float。默认值：1.0。
- **inplace** (bool, 可选) - 是否使用原地更新模式，数据类型为 bool。默认值：False。

输入：

- **input** (Tensor) - 用于计算 ELU 的任意维度的 Tensor。

输出：

Tensor，输出的 shape 和数据类型与 *input* 相同。

异常：

- **RuntimeError** - 如果 *input* 的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - 如果 *alpha* 的数据类型不是 float。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float32)
>>> elu = mint.nn.ELU()
>>> result = elu(input)
>>> print(result)
[-0.63212055 -0.86466473  0.  2.  1.]
```

mindspore.mint.nn.GELU

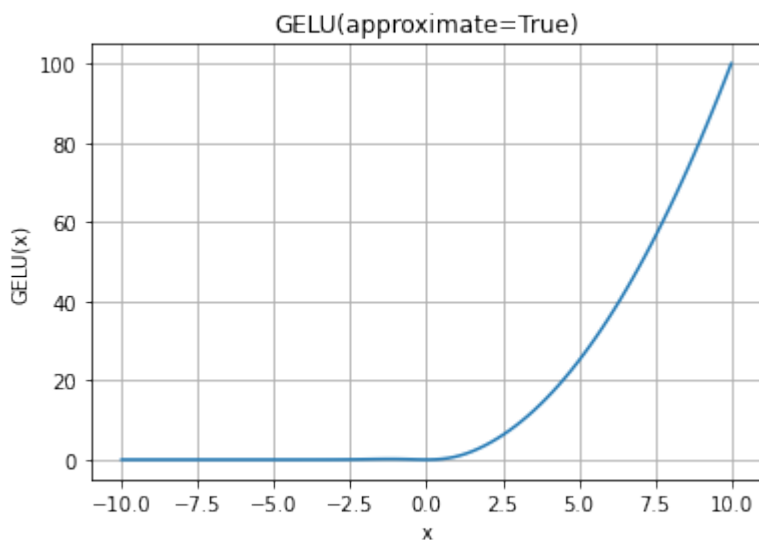
class mindspore.mint.nn.GELU (*approximate='none'*)

高斯误差线性单元激活函数 (Gaussian Error Linear Unit)。

GELU 的详细描述可以参考论文: [Gaussian Error Linear Units \(GELUs\)](#), 或论文: [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)。

更多参考详见 `mindspore.mint.nn.functional.gelu()`。

GELU 函数图:



支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]), mindspore.float32)
>>> gelu = mint.nn.GELU()
>>> output = gelu(input)
>>> print(output)
[[-1.58655241e-01  3.99987316e+00 -0.00000000e+00]
 [ 1.95449972e+00 -1.41860323e-06  9.00000000e+00]]
>>> gelu = mint.nn.GELU(approximate="tanh")
>>> output = gelu(input)
```

(下页继续)

(续上页)

```
>>> print(output)
[[-1.58808023e-01  3.99992990e+00 -3.10779147e-21]
 [ 1.95459759e+00 -2.29180174e-07  9.00000000e+00]]
```

mindspore.mint.nn.GLU

class mindspore.mint.nn.GLU(*dim=-1*)

计算输入 Tensor 的门线性单元激活函数 (Gated Linear Unit activation function) 值。

$$GLU(a, b) = a \otimes \sigma(b)$$

其中, a 表示输入 Tensor *input* 拆分后的前半元素, b 表示 *input* 拆分后的另一半元素。这里 σ 为 sigmoid 函数, $\otimes$ 是 Hadamard 乘积。请参考 [Language Modeling with Gated Convluational Networks](#)。

参数:

- **dim** (int, 可选) - 指定分割 *input* 的轴, 取值范围 $[-r, r)$, 其中 r 为 *input* 的维度数。默认值: -1 , 输入 *input* 的最后一维。

输入:

- **input** (Tensor) - 待计算的 Tensor, 数据类型为浮点型, shape 为 $(*_1, N, *_2)$, 其中 $*$ 为任意额外维度, 且要求 N 为偶数。 N 为 *input* 在被 *dim* 选中的维度上的大小。

输出:

Tensor, 数据类型与输入 *input* 相同, shape 为 $(*_1, M, *_2)$, 其中 $M = N/2$ 。

异常:

- **TypeError** - *input* 不是 Tensor 或 *dim* 不是整型。
- **IndexError** - *dim* 的值超出了 $[-r, r)$ 的范围, 其中 r 为 *input* 的维度数。
- **RuntimeError** - *input* 数据类型不被支持。
- **RuntimeError** - *input* 在被 *dim* 选中的维度上的长度不为偶数。

支持平台:

Ascend

样例:

```
>>> from mindspore import mint, Tensor
>>> glu = mint.nn.GLU()
>>> input = Tensor([[0.1, 0.2, 0.3, 0.4], [0.5, 0.6, 0.7, 0.8]])
>>> output = glu(input)
>>> print(output)
[[0.05744425 0.11973753]
 [0.33409387 0.41398472]]
```

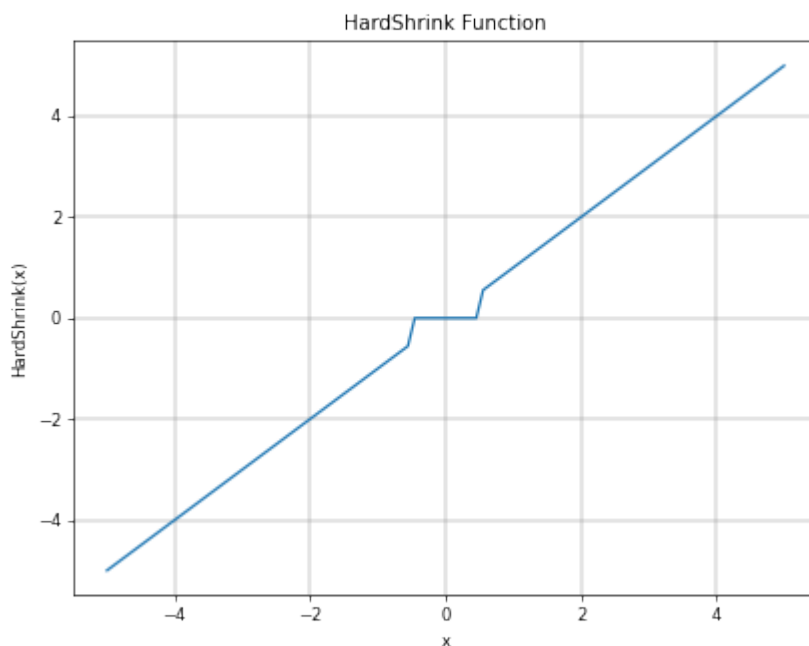
`mindspore.mint.nn.Hardshrink`

class `mindspore.mint.nn.Hardshrink` (*lambd=0.5*)

逐元素计算 Hard Shrink 激活函数。公式定义如下：

$$\text{HardShrink}(x) = \begin{cases} x, & \text{if } x > \lambda \\ x, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

HardShrink 函数图：



参数：

- **lambd** (number, 可选) - Hard Shrink 公式定义的阈值 λ 。默认值：0.5。

输入：

- **input** (Tensor) - Hard Shrink 的输入。支持数据类型：
 - Ascend: float16、float32、bfloat16。

输出：

Tensor, shape 和数据类型与输入相同。

异常：

- **TypeError** - *lambd* 不是 float、int 或 bool。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的 dtype 不是 float16、float32 或 bfloat16。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[0.5, 1, 2.0], [0.0533, 0.0776, -2.1233]]), mindspore.float32)
>>> hshrink = mint.nn.Hardshrink()
>>> output = hshrink(input)
>>> print(output)
[[ 0.    1.    2.   ]
 [ 0.    0.   -2.1233]]
```

mindspore.mint.nn.Hardsigmoid

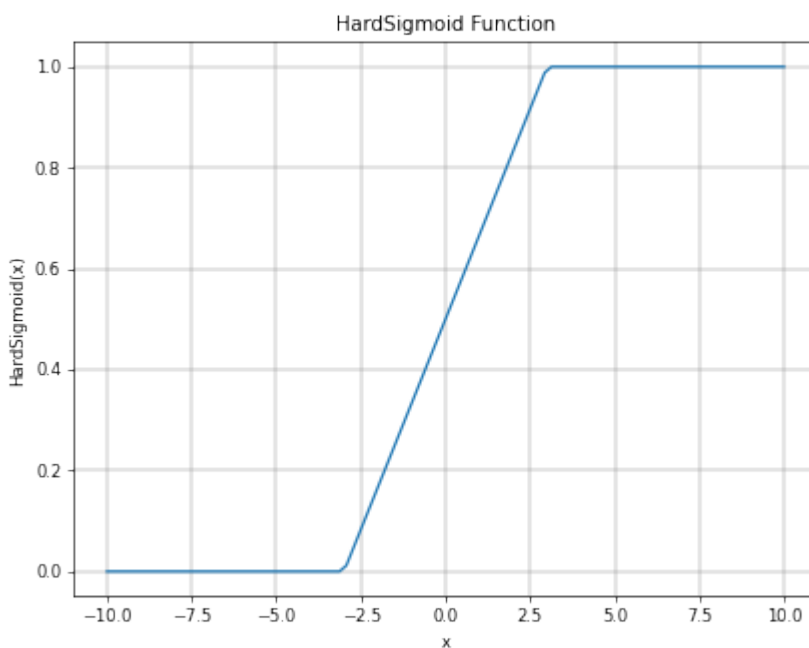
class mindspore.mint.nn.Hardsigmoid

逐元素计算 Hard Sigmoid 激活函数。

Hard Sigmoid 定义为：

$$\text{HardSigmoid}(input) = \begin{cases} 0, & \text{if } input \leq -3, \\ 1, & \text{if } input \geq +3, \\ input/6 + 1/2, & \text{otherwise} \end{cases}$$

HardSigmoid 函数图：



输入：

- **input** (Tensor) - Hard Sigmoid 的输入。

输出：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> hsigmoid = mint.nn.Hardsigmoid()
>>> result = hsigmoid(input)
>>> print(result)
[0.3333 0.1666 0.5      0.8335 0.6665]
```

mindspore.mint.nn.Hardswish

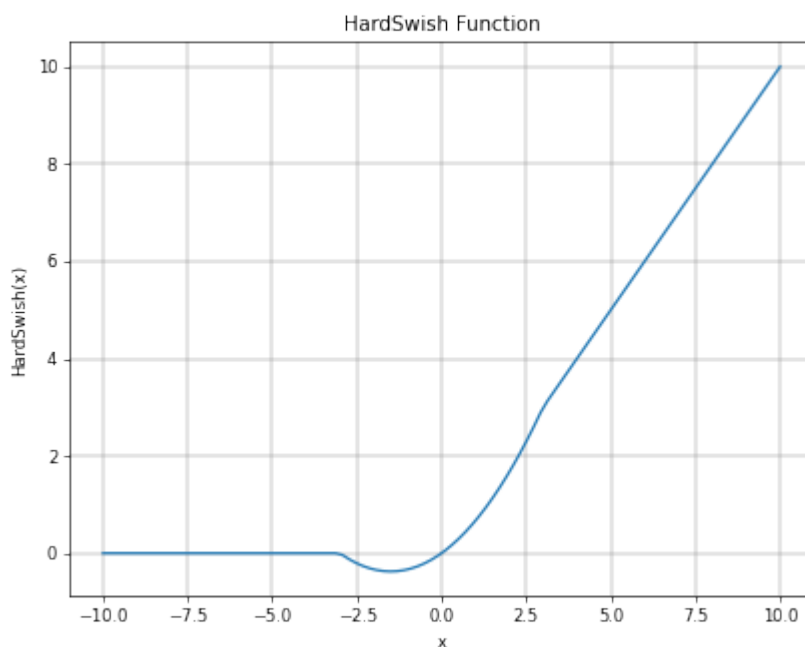
class mindspore.mint.nn.Hardswish

逐元素计算 Hard Swish 激活函数。

Hard Swish 定义如下：

$$\text{HardSwish}(\text{input}) = \begin{cases} 0, & \text{if } \text{input} \leq -3, \\ \text{input}, & \text{if } \text{input} \geq +3, \\ \text{input} * (\text{input} + 3)/6, & \text{otherwise} \end{cases}$$

HardSwish 函数图：



输入：

- **input** (Tensor) - Hard Swish 的输入。

输出：

Tensor，具有与 *input* 相同的数据类型和 shape。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> hswish = mint.nn.Hardswish()
>>> result = hswish(input)
>>> print(result)
[-0.3333 -0.3333  0.      1.667  0.6665]
```

mindspore.mint.nn.LogSigmoid

class mindspore.mint.nn.LogSigmoid

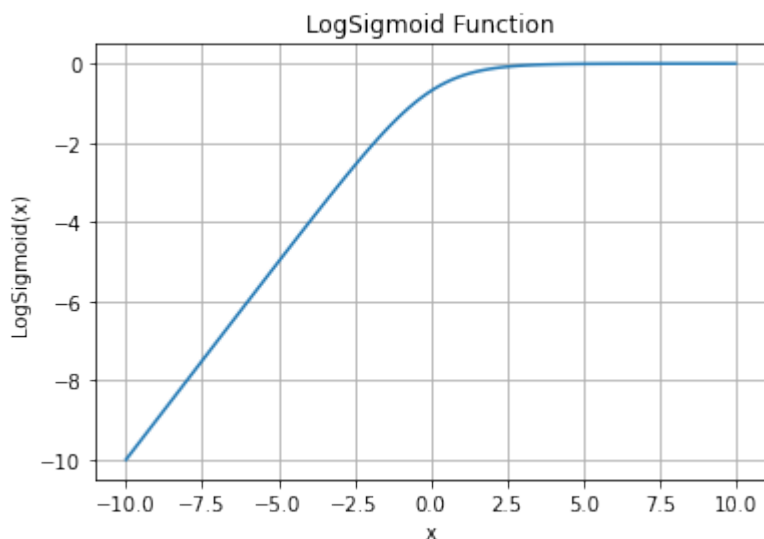
逐元素计算 LogSigmoid 激活函数。输入是任意格式的 Tensor。

LogSigmoid 定义为：

$$\text{LogSigmoid}(x_i) = \log\left(\frac{1}{1 + \exp(-x_i)}\right),$$

其中， x_i 是输入 Tensor 的一个元素。

LogSigmoid 函数图：



警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **input** (Tensor) - LogSigmoid 的输入，数据类型为 bfloat16, float16 或 float32。shape 为 (*)，其中 * 表示任意的附加维度。

输出：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型既不是 bfloat16, float16 也不是 float32。
- **TypeError** - *input* 不是一个 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> net = mint.nn.LogSigmoid()
>>> input = Tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> output = net(input)
>>> print(output)
[-0.31326166 -0.12692806 -0.04858734]
```

mindspore.mint.nn.LogSoftmax

class mindspore.mint.nn.**LogSoftmax** (*dim=None*)

在指定轴上对输入 Tensor 应用 LogSoftmax 函数。假设在指定轴上， x 对应每个元素 x_i ，则 LogSoftmax 函数如下所示：

$$\text{output}(x_i) = \log \left(\frac{\exp(x_i)}{\sum_{j=0}^{N-1} \exp(x_j)} \right),$$

其中， N 为 Tensor 长度。

参数：

- **dim** (int, 可选) - 指定进行 Log softmax 运算的轴。默认值：None。

返回：

Tensor，和输入 Tensor 的 shape 相同。

异常：

- **ValueError** - 如果 *dim* 超出范围 $[-\text{len}(\text{input.shape}), \text{len}(\text{input.shape})$ 。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> log_softmax = mint.nn.LogSoftmax(dim=-1)
>>> output = log_softmax(x)
>>> print(output)
[[-5.00672150e+00 -6.72150636e-03 -1.20067215e+01]
 [-7.00091219e+00 -1.40009127e+01 -9.12250078e-04]]
```

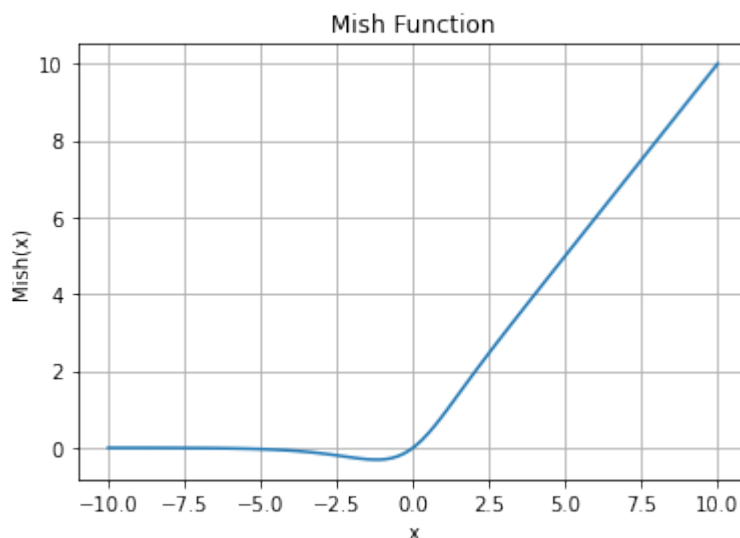
mindspore.mint.nn.Mish

class mindspore.mint.nn.Mish

逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。

更多参考详见 `mindspore.mint.nn.functional.mish()`。

Mish 激活函数图：



支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> x = Tensor(np.array([[ -1.1,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> mish = mint.nn.Mish()
>>> output = mish(x)
>>> print(output)
[[-3.0764845e-01  3.9974124e+00 -2.6832507e-03]
 [ 1.9439589e+00 -3.3576239e-02  8.9999990e+00]]
```

mindspore.mint.nn.PReLU

class mindspore.mint.nn.PReLU (num_parameters=1, init=0.25, dtype=None)

逐元素计算 PReLU (PReLU Activation Operator) 激活函数。

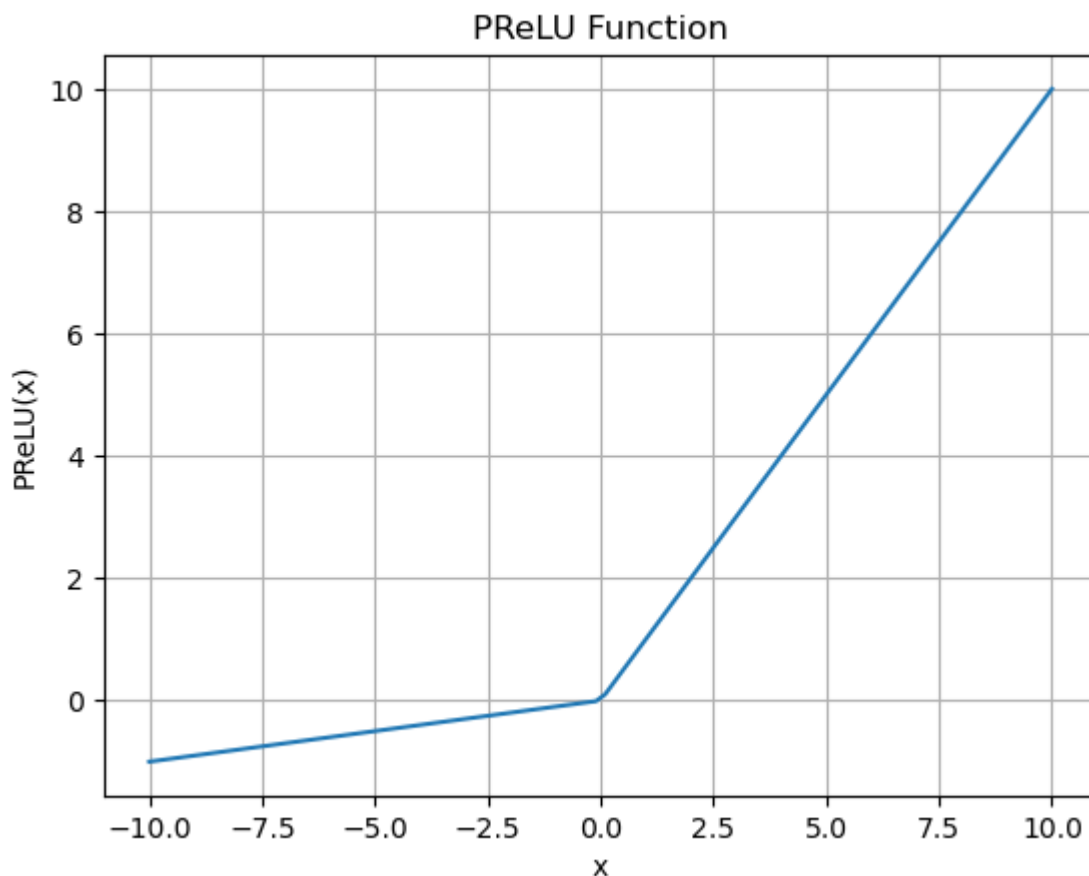
公式定义为：

$$PReLU(x_i) = \max(0, x_i) + w * \min(0, x_i),$$

其中 x_i 是输入的 Tensor。

这里 w 是一个可学习的参数，默认初始值 0.25。当带参数调用时每个通道上学习一个 w 。如果不带参数调用时，则将在所有通道中共享单个参数 w 。

PReLU 函数图：



说明：通道数是输入的第二个维度值。当输入的维度小于 2 时，则没有通道维度并且通道数等于 1。

参数：

- **num_parameters** (int, 可选) - 可训练参数 w 的数量。虽然这个参数可以接受一个 int 作为输入，但只有两个值是合法的，值是 1 或输入 Tensor *input* 的通道数。默认值：1。
- **init** (float, 可选) - 参数的初始值。默认值：0.25。
- **dtype** (mindspore.dtype, 可选) - 参数的 dtype。默认值：None。支持的数据类型是 {float16, float32, bfloat16}。

输入:

- **input** (Tensor) - PReLU 的输入 Tensor。

输出:

Tensor, 数据类型和 shape 与 *input* 相同。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> x = Tensor(np.array([[[[0.1, 0.6], [0.9, 0.9]]]]), mindspore.float32)
>>> prelu = mint.nn.PReLU()
>>> output = prelu(x)
>>> print(output)
[[[0.1 0.6]
  [0.9 0.9]]]
```

mindspore.mint.nn.ReLU

class mindspore.mint.nn.ReLU

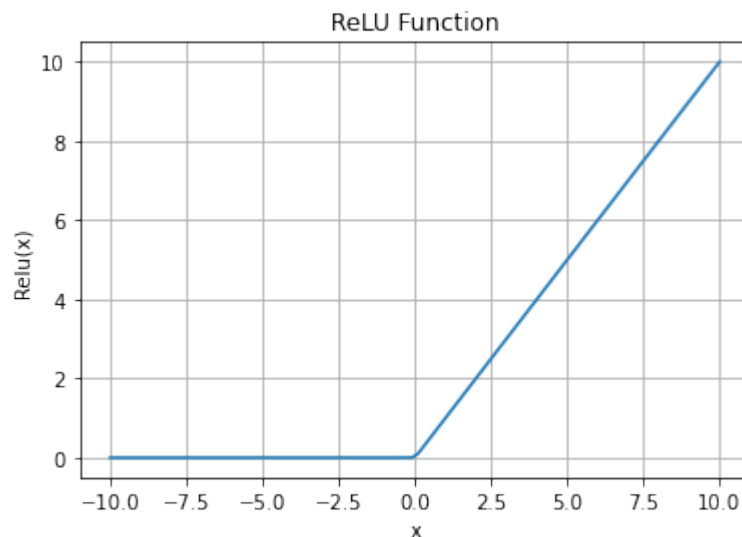
逐元素计算 ReLU (Rectified Linear Unit activation function) 修正线性单元激活函数。

$$\text{ReLU}(\text{input}) = (\text{input})^+ = \max(0, \text{input}),$$

逐元素求 $\max(0, \text{input})$ 。

说明: 负数输出值会被修改为 0, 正数输出不受影响。

ReLU 激活函数图:



输入:

- **input** (Tensor) - 用于计算 ReLU 的任意维度的 Tensor。

输出：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *input* 的数据类型不支持。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([-1, 2, -3, 2, -1]), mindspore.float16)
>>> relu = mint.nn.ReLU()
>>> output = relu(input)
>>> print(output)
[0. 2. 0. 2. 0.]
```

mindspore.mint.nn.ReLU6

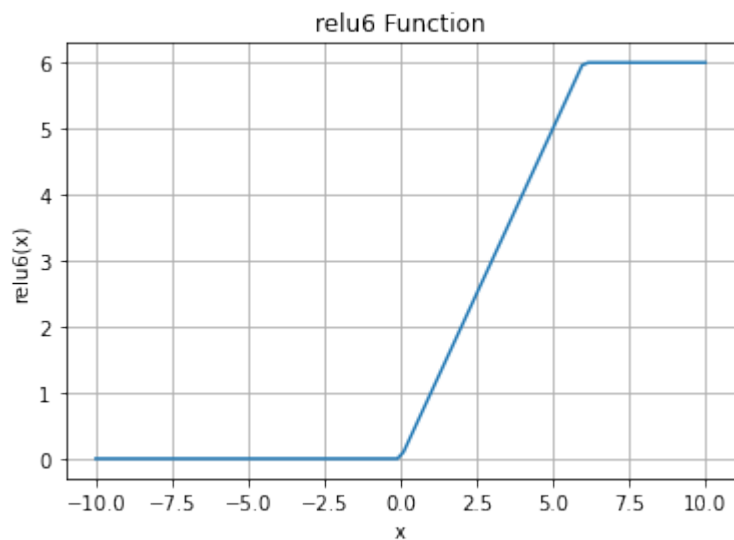
class mindspore.mint.nn.ReLU6 (*inplace=False*)

ReLU6 激活函数。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.mint.nn.functional.relu6()`。

ReLU6 激活函数图：



支持平台：

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> relu6 = mint.nn.ReLU6()
>>> output = relu6(input)
>>> print(output)
[[0.  4.  0.]
 [2.  0.  6.]]
```

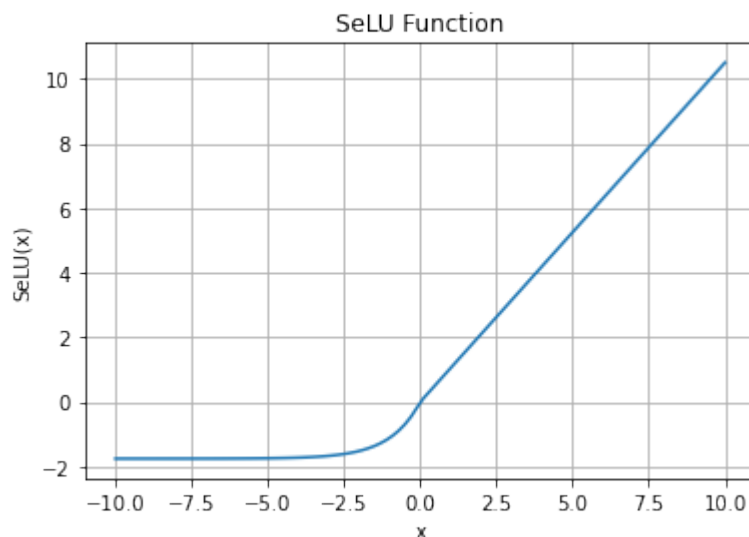
mindspore.mint.nn.SELU

class mindspore.mint.nn.SELU

逐元素计算激活函数 SELU (Scaled exponential Linear Unit)。

更多参考详见 `mindspore.mint.nn.functional.selu()`。

SELU 函数图:



支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> selu = mint.nn.SELU()
>>> output = selu(input)
>>> print(output)
[[-1.1113307  4.202804 -1.7575096]
 [ 2.101402 -1.7462534  9.456309  ]]
```

`mindspore.mint.nn.SiLU`

class `mindspore.mint.nn.SiLU`

逐元素计算 SiLU 激活函数。有时也被称作 Swish 函数。

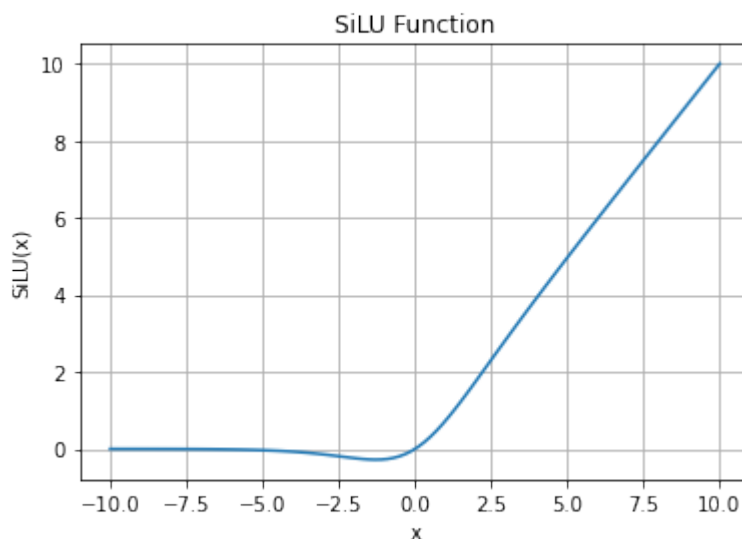
SiLU 函数定义为：

$$\text{SiLU}(x) = x * \sigma(x),$$

其中 x_i 是输入的元素， $\sigma(x)$ 是 Sigmoid 函数。

$$\text{sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)},$$

SiLU 函数图：



警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **input** (Tensor) - *input* 即为上述公式中的 x 。数据类型为 `float16` 或 `float32` 的输入。任意维度的 Tensor。

输出：

Tensor，数据类型和 `shape` 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型既不是 `float16` 也不是 `float32`。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, 2, -3, 2, -1]), mindspore.float16)
>>> silu = mint.nn.SiLU()
>>> output = silu(input)
>>> print(output)
[-0.269  1.762 -0.1423  1.762 -0.269]
```

mindspore.mint.nn.Sigmoid

class mindspore.mint.nn.Sigmoid

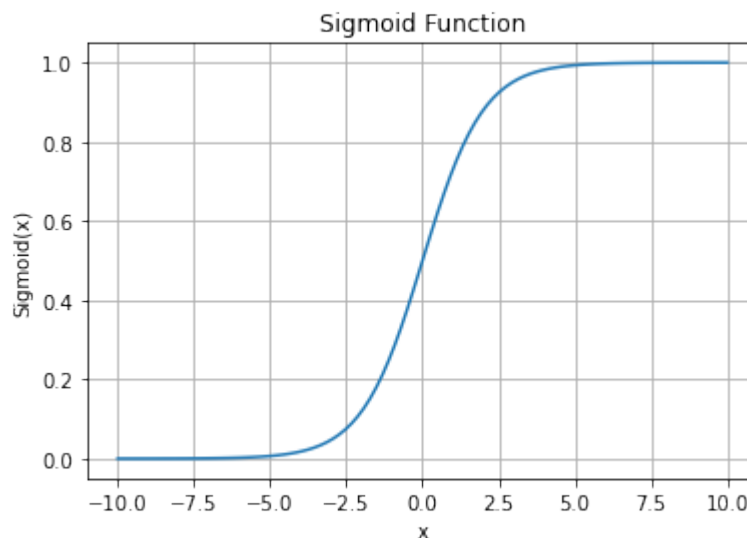
逐元素计算 Sigmoid 激活函数。

Sigmoid 函数定义为：

$$\text{sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)},$$

其中 x_i 是 x 的一个元素。

Sigmoid 函数图：



输入：

- **input** (Tensor) - *input* 即为上述公式中的 x 。数据类型为 float16、float32、float64、complex64 或 complex128 的 Sigmoid 输入。任意维度的 Tensor。

输出：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> sigmoid = mint.nn.Sigmoid()
>>> output = sigmoid(input)
>>> print(output)
[0.2688  0.11914 0.5      0.881   0.7305 ]
```

mindspore.mint.nn.Softmax

class mindspore.mint.nn.Softmax(*dim=None*)

将 Softmax 函数应用于 *n* 维输入 Tensor。

更多参考详见 `mindspore.mint.nn.functional.softmax()`。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> softmax = mint.nn.Softmax()
>>> output = softmax(input)
>>> print(output)
[0.03168 0.01166 0.0861  0.636   0.2341 ]
```

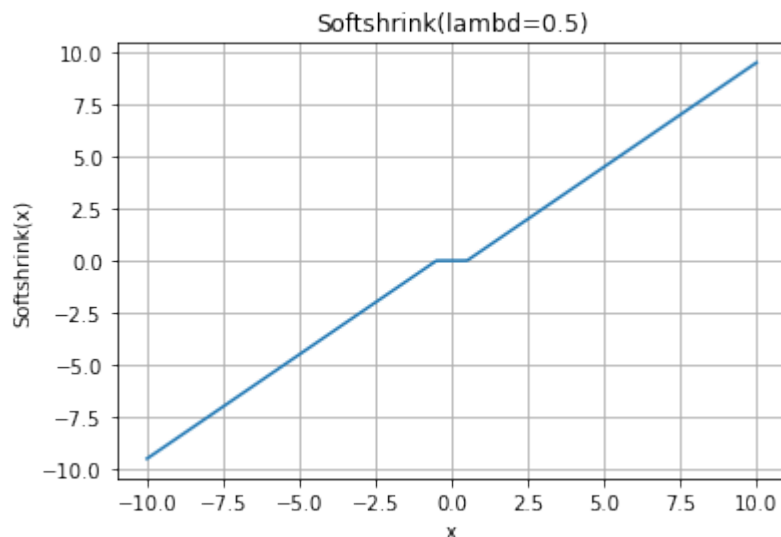
mindspore.mint.nn.Softshrink

class mindspore.mint.nn.Softshrink(*lambd=0.5*)

逐元素计算 Soft Shrink 激活函数。公式定义如下:

$$\text{SoftShrink}(x) = \begin{cases} x - \lambda, & \text{if } x > \lambda \\ x + \lambda, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

SoftShrink 激活函数图:



参数：

- **lambd** (number, 可选) - Soft Shrink 公式定义的阈值 λ ，必须不小于零。默认值：0.5。

输入：

- **input** (Tensor) - Soft Shrink 的输入。支持数据类型：
 - Ascend: float16、float32、bfloat16。

输出：

Tensor, shape 和数据类型与输入相同。

异常：

- **TypeError** - *lambd* 不是 float、int 或 bool。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的 dtype 不是 float16、float32 或 bfloat16。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[ 0.5297,  0.7871,  1.1754], [ 0.7836,  0.6218, -1.1542]]),
↳mindspore.float16)
>>> softshrink = mint.nn.Softshrink()
>>> output = softshrink(input)
>>> print(output)
[[ 0.02979  0.287   0.676  ]
 [ 0.2837   0.1216 -0.6543  ]]
```

mindspore.mint.nn.Tanh**class mindspore.mint.nn.Tanh**

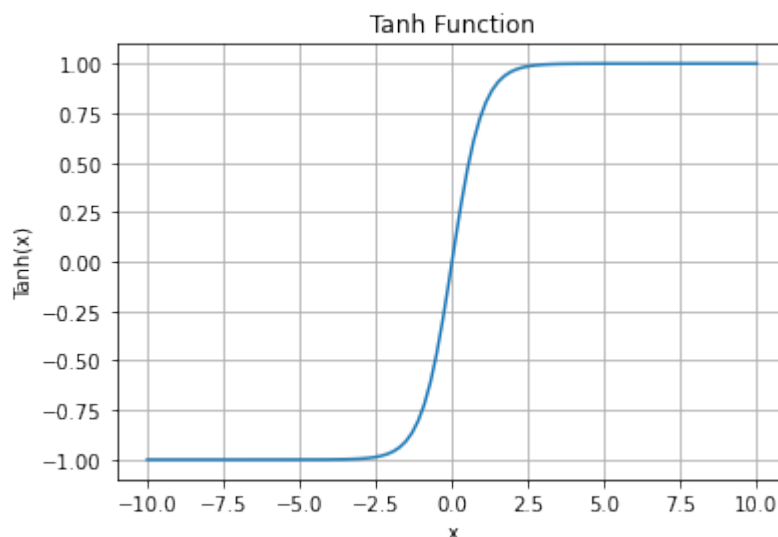
逐元素计算 Tanh（双曲正切值）激活函数，返回一个新的 Tensor。

Tanh 函数定义为：

$$\tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 Tensor 的元素。

Tanh 函数图：



警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **input** (Tensor) - 任意维度的 Tensor，数据类型为 float16 或 float32 的输入。

输出：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([1, 2, 3, 2, 1]), mindspore.float16)
>>> tanh = mint.nn.Tanh()
```

(下页继续)

(续上页)

```
>>> output = tanh(input)
>>> print(output)
[0.7617 0.964 0.995 0.964 0.7617]
```

4.4.4 嵌入层

接口名	概述	支持平台
<code>mindspore.mint.nn.Embedding</code>	以 <code>input</code> 中的值作为索引，从 <code>weight</code> 中查询对应的 <code>embedding</code> 向量。	Ascend

mindspore.mint.nn.Embedding

`class mindspore.mint.nn.Embedding (num_embeddings, embedding_dim, padding_idx=None, max_norm=None, norm_type=2.0, scale_grad_by_freq=False, sparse=False, _weight=None, _freeze=False, dtype=None)`
以 `input` 中的值作为索引，从 `weight` 中查询对应的 `embedding` 向量。

- 警告：
- 这是一个实验性 API，后续可能修改或删除。
 - 在 Ascend 后端，`input` 的值非法将导致不可预测的行为。

参数：

- **num_embeddings** (int) - embedding 词典的大小。
- **embedding_dim** (int) - 每个 embedding 向量的长度。
- **padding_idx** (int, 可选) - 如果值不为 None，则对应的 embedding 向量不会在训练中更新。且当 Embedding 层新构建时，`padding_idx` 处 embedding 向量值将默认为 0。取值范围：`[-num_embeddings, num_embeddings)`。默认值：None。
- **max_norm** (float, 可选) - 如果给定非 None 值，则先求出 `input` 指定位置的 `weight` 的 p-范数结果（p 的值通过 `norm_type` 指定），然后对 `result > max_norm` 位置的 `weight` 进行原地更新，更新公式： $\frac{max\_norm}{result+1e^{-7}}$ 。默认值：None。
- **norm_type** (float, 可选) - 指定 p-范数计算中的 p 值。默认值：2.0。
- **scale_grad_by_freq** (bool, 可选) - 如果值为 True，则反向梯度值会按照 `input` 中索引值重复的次数进行缩放。默认值：False。
- **sparse** (bool, 可选) - 如果值为 True，则 `weight` 为稀疏 Tensor，该场景暂未支持。默认值：False。
- **_weight** (Tensor, 可选) - 用于初始化 Embedding 的权重 `weight`。如果为 None，则权重将为从正态分布初始化 $N(\sigma=1.0, \text{mean}=0.0)$ 。默认值：None。
- **_freeze** (bool, 可选) - 是否冻结该模型的可学习参数 `weight`。默认值 False。
- **dtype** (mindspore.dtype, 可选) - 指定 Embedding 权重 `weight` 的类型。当 `_weight` 不是 None 时该参数无效。默认值：None。

可变参数：

- **weight** (Parameter) - 模型的可学习参数, shape 为 $(num\_embeddings, embedding\_dim)$, 从正态分布 $N(\sigma=1.0, \mu=0.0)$ 或 `_weight` 初始化。

输入:

- **input** (Tensor) - 用于检索 embedding 向量的索引。数据类型为 int32 或 int64, 取值范围: $[0, num\_embeddings)$ 。

输出:

Tensor, 数据类型与 `weight` 保持一致, shape 为 $(*input.shape, embedding\_dim)$ 。

异常:

- **TypeError** - `num_embeddings` 不是 int。
- **TypeError** - `embedding_dim` 不是 int。
- **ValueError** - `padding_idx` 的取值不在有效范围。
- **TypeError** - `max_norm` 不是 float。
- **TypeError** - `norm_type` 不是 float。
- **TypeError** - `scale_grad_by_freq` 不是 bool 型。
- **ValueError** - `weight.shape` 非法。
- **TypeError** - `dtype` 不是 `mindspore.dtype`。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> mindspore.set_seed(0)
>>> input = Tensor([[1, 0, 1, 1], [0, 0, 1, 0]])
>>> embedding = mint.nn.Embedding(num_embeddings=10, embedding_dim=3)
>>> output = embedding(input)
>>> print(output)
[[[ 0.6712398  0.5407775  1.0317237]
  [-0.49091062 -0.42302188 -1.4807187]
  [ 0.6712398  0.5407775  1.0317237]
  [ 0.0024154  0.5407775  1.0317237]]
 [[-0.49091062 -0.42302188 -1.4807187]
  [-0.49091062 -0.42302188 -1.4807187]
  [ 0.6712398  0.5407775  1.0317237]
  [-0.49091062 -0.42302188 -1.4807187]]]
```

4.4.5 线性层

接口名	概述	支持平台
<code>mindspore.mint.nn.Linear</code>	全连接层。	Ascend

mindspore.mint.nn.Linear

class mindspore.mint.nn.**Linear** (*in_features, out_features, bias=True, weight_init=None, bias_init=None, dtype=None*)
全连接层。

适用于输入的密集连接层。公式如下：

$$\text{outputs} = X * \text{kernel} + \text{bias}$$

其中 X 是输入 Tensor， kernel 是一个权重矩阵，其数据类型与 X 相同， bias 是一个偏置向量，其数据类型与 X 相同 (仅当参数 bias 为 `True` 时)。

警告： 在 Ascend 硬件平台下，设置 PYNATIVE 或 KBK 模式时，如果 bias 为 `False`， x 不可以大于 6D。

参数：

- in_features** (int) - Linear 层输入 Tensor 的空间维度。
- out_features** (int) - Linear 层输出 Tensor 的空间维度。
- bias** (bool, 可选) - 是否使用偏置向量 bias 。默认值：`True`。
- weight_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 权重参数的初始化方法。数据类型与 x 相同。str 的值引用自函数 *initializer*。默认值：`None`，权重使用 `HeUniform` 初始化。
- bias_init** (Union[Tensor, str, Initializer, numbers.Number], 可选) - 偏置参数的初始化方法。数据类型与 x 相同。str 的值引用自函数 *initializer*。默认值：`None`，偏差使用 `Uniform` 初始化。
- dtype** (*mindspore.dtype*, 可选) - Parameter 的数据类型。默认值：`None`。如果 dtype 为 `None`，初始化方法时 dtype 会被设为 `mstype.float32`。当 weight_init 是 Tensor 时，Parameter 的数据类型与 weight_init 数据类型一致，其他情况 Parameter 的数据类型跟 dtype 一致，同理 bias_init 。

输入：

- x** (Tensor) - shape 为 $(*, \text{in\_features})$ 的 Tensor。参数中的 in_features 应等于输入中的 in_features 。

输出：

shape 为 $(*, \text{out\_features})$ 的 Tensor。

异常：

- TypeError** - in_features 或 out_features 不是整数。
- TypeError** - bias 不是 bool 值。
- ValueError** - weight_init 的 shape 长度不等于 2， weight_init 的 $\text{shape}[0]$ 不等于 out_features ，或者 weight_init 的 $\text{shape}[1]$ 不等于 in_features 。
- ValueError** - bias_init 的 shape 长度不等于 1 或 bias_init 的 $\text{shape}[0]$ 不等于 out_features 。
- RuntimeError** - 在 Ascend 硬件平台下，设置 PYNATIVE 或 KBK 模式时， bias 为 `False` 且 x 大于 6D。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> import numpy as np
>>> x = Tensor(np.array([[180, 234, 154], [244, 48, 247]]), mindspore.float32)
>>> net = mint.nn.Linear(3, 4)
>>> output = net(x)
>>> print(output.shape)
(2, 4)
```

4.4.6 Dropout 层

接口名	概述	支持平台
<code>mindspore.mint.nn.Dropout</code>	随机丢弃层。	Ascend
<code>mindspore.mint.nn.Dropout2d</code>	在训练期间，以服从伯努利分布的概率 p 随机将输入 <code>Tensor</code> 的某些通道归零（对于 <code>shape</code> 为 <code>NCHW</code> 的四维 <code>Tensor</code> ，其通道特征图指的是后两维 <code>HW</code> 的二维特征图）。	Ascend

`mindspore.mint.nn.Dropout`

`class mindspore.mint.nn.Dropout (p=0.5, inplace=False)`
随机丢弃层。

`Dropout` 是一种正则化手段，通过阻止神经元节点间的相关性来减少过拟合。该操作根据丢弃概率 p ，在训练过程中随机将一些神经元输出设置为 0。并且训练过程中返回值会乘以 $\frac{1}{1-p}$ 。在推理过程中，此层返回与输入 x 相同的 `Tensor`。

论文 [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#) 中提出了该技术，并证明其能有效地减少过度拟合，防止神经元共适应。更多详细信息，请参见 [Improving neural networks by preventing co-adaptation of feature detectors](#)。

说明:

- 训练过程中，每步对同一通道（或神经元）独立进行丢弃。
- p 表示输入 `Tensor` 中元素设置成 0 的概率。

参数:

- **p** (float, 可选) - 输入神经元丢弃概率。例如， $p=0.9$ ，即删除 90% 的神经元。默认值: 0.5。
- **inplace** (bool, 可选) - 是否启用原地更新功能。若为 `True`，则启用原地更新功能。默认值: `False`。

输入:

- **x** (`Tensor`) - `Dropout` 的输入。

输出:

`Tensor`，输出为 `Tensor`，其 `shape` 与 x 的 `shape` 相同。

异常:

- **TypeError** - p 数据类型不是 float。
- **ValueError** - x 的 shape 长度小于 1。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> x = Tensor(np.ones([2, 2, 3]), mindspore.float32)
>>> net = mint.nn.Dropout(p=0.2)
>>> net.set_train()
>>> output = net(x)
>>> print(output.shape)
(2, 2, 3)
```

mindspore.mint.nn.Dropout2d

class mindspore.mint.nn.Dropout2d ($p=0.5$)

在训练期间, 以服从伯努利分布的概率 p 随机将输入 Tensor 的某些通道归零 (对于 shape 为 $NCHW$ 的四维 Tensor, 其通道特征图指的是后两维 HW 的二维特征图)。例如, 在批处理输入中 i_{th} 批, j_{th} 通道的 $input[i, j]$ 2D Tensor 是一个待处理数据。每个通道将会独立依据伯努利分布概率 p 来确定是否被清零。

Dropout2d 可以提高通道特征映射之间的独立性。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多参考详见 `mindspore.mint.nn.functional.dropout2d()`。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> dropout = mint.nn.Dropout2d(p=0.5)
>>> x = Tensor(np.ones([2, 1, 2, 3]), mindspore.float32)
>>> output = dropout(x)
>>> print(output.shape)
(2, 1, 2, 3)
```

4.4.7 池化层

接口名	概述	支持平台
<code>mindspore.mint.nn.AdaptiveAvgPool1d</code>	对由多个输入平面组成的输入信号，运用 1D 自适应平均池化。	Ascend
<code>mindspore.mint.nn.AdaptiveAvgPool2d</code>	对由多个输入平面组成的输入信号，运用 2D 自适应平均池化。	Ascend
<code>mindspore.mint.nn.AdaptiveAvgPool3d</code>	对输入 Tensor，运用三维的自适应平均池化操作。	Ascend
<code>mindspore.mint.nn.AdaptiveMaxPool1d</code>	对由多个输入平面组成的输入信号应用 1D 自适应最大池化。	Ascend
<code>mindspore.mint.nn.AvgPool2d</code>	对输入张量应用二维平均池化，可视为二维输入平面的组合。	Ascend
<code>mindspore.mint.nn.AvgPool3d</code>	对输入张量应用三维平均池化，可视为三维输入平面的组合。	Ascend
<code>mindspore.mint.nn.MaxUnpool2d</code>	<i>Maxpool2d</i> 的逆过程。	Ascend

mindspore.mint.nn.AdaptiveAvgPool1d

class `mindspore.mint.nn.AdaptiveAvgPool1d` (*output_size*)
对由多个输入平面组成的输入信号，运用 1D 自适应平均池化。
对于任何输入大小，输出大小均为 L_{out} 。输出特征的数量等于输入平面的数量。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- output_size** (int) - 目标输出的 size L_{out} 。

输入：

- input** (Tensor) - 输入特征的 shape 为 (N, C, L_{in}) 或 (C, L_{in}) 。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[2, 1, 2], [2, 3, 5]]), mindspore.float16)
>>> net = mint.nn.AdaptiveAvgPool1d(3)
>>> output = net(input)
>>> print(output)
[[2. 1. 2.]
 [2. 3. 5.]]
```

`mindspore.mint.nn.AdaptiveAvgPool2d`

class `mindspore.mint.nn.AdaptiveAvgPool2d` (*output_size*)

对由多个输入平面组成的输入信号，运用 2D 自适应平均池化。

对于任何输入大小，输出大小均为 $H \times W$ 。输出特征的数量等于输入平面的数量。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **output_size** (`Union(int, tuple[int])`) - $H \times W$ 形式的图像的目标输出尺寸。可以是 `tuple HW`，也可以是正方形图像 $H \times H$ 的单个 H 。 H 和 W 可以是 `int` 或 `None`，这意味着大小将与输入相同。

输入：

- **input** (`Tensor`) - 输入特征的 shape 为 $(N, C, *)$ ，其中 $*$ 任意数量的附加维度。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[[[2, 1, 2], [2, 3, 5]]]]), mindspore.float16)
>>> net = mint.nn.AdaptiveAvgPool2d((2, 2))
>>> output = net(input)
>>> print(output)
[[[1.5 1.5]
  [2.5 4. ]]]
```

`mindspore.mint.nn.AdaptiveAvgPool3d`

class `mindspore.mint.nn.AdaptiveAvgPool3d` (*output_size*)

对输入 `Tensor`，运用三维的自适应平均池化操作。即对于输入任何尺寸，指定输出的尺寸均为 (D, H, W) 。但输入和输出特征的数量不会变化。

假设输入 *input* 最后三维大小分别为 (inD, inH, inW) ，则输出的最后三维大小分别为 $(outD, outH, outW)$ 。运算如下：

$$\forall \quad od \in [0, outD - 1], oh \in [0, outH - 1], ow \in [0, outW - 1]$$

$$output[od, oh, ow] = \text{mean}(input[i_{startD} : i_{endD} + 1, i_{startH} : i_{endH} + 1, i_{startW} : i_{endW} + 1])$$

where,

$$i_{startD} = \left\lceil \frac{od * inD}{outD} \right\rceil$$

$$i_{endD} = \left\lceil \frac{(od+1) * inD}{outD} \right\rceil$$

$$i_{startH} = \left\lceil \frac{oh * inH}{outH} \right\rceil$$

$$i_{endH} = \left\lceil \frac{(oh+1) * inH}{outH} \right\rceil$$

$$i_{startW} = \left\lceil \frac{ow * inW}{outW} \right\rceil$$

$$i_{endW} = \left\lceil \frac{(ow+1) * inW}{outW} \right\rceil$$

警告： 对于 Ascend，仅支持 Atlas A2 训练系列产品。这是一个实验性 API，后续可能修改或删除。

参数：

- **output_size** (Union[int, tuple]) - 指定输出特征图的尺寸，可以是 tuple (D, H, W)，也可以是一个 int 值 D 来表示输出尺寸为 (D, D, D)。 D 、 H 和 W 可以是 int 值或者 None，其中 None 表示输出大小与对应的输入的大小相同。

输入：

- **input** (Tensor) - AdaptiveAvgPool3d 的输入，是 4D 或者 5D 的 Tensor。

输出：

Tensor，与输入 *input* 的数据类型相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *input* 维度不是 4D 或者 5D。
- **ValueError** - *output_size* 的值不是正数。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> import numpy as np
>>> # case 1: output_size=(3, 3, 4)
>>> output_size=(3, 3, 4)
>>> input_x_val = np.random.randn(4, 3, 5, 6, 7)
>>> input_x = ms.Tensor(input_x_val, ms.float32)
>>> net = mint.nn.AdaptiveAvgPool3d(output_size)
>>> output = net(input_x)
>>> print(output.shape)
(4, 3, 3, 3, 4)
>>> # case 2: output_size=4
>>> output_size=5
>>> input_x_val = np.random.randn(2, 3, 8, 6, 12)
>>> input_x = ms.Tensor(input_x_val, ms.float32)
>>> net = mint.nn.AdaptiveAvgPool3d(output_size)
>>> output = net(input_x)
>>> print(output.shape)
(2, 3, 5, 5, 5)
>>> # case 3: output_size=(None, 4, 5)
>>> output_size=(None, 4, 5)
>>> input_x_val = np.random.randn(4, 1, 9, 10, 8)
>>> input_x = ms.Tensor(input_x_val, ms.float32)
>>> net = mint.nn.AdaptiveAvgPool3d(output_size)
>>> output = net(input_x)
>>> print(output.shape)
(4, 1, 9, 4, 5)
```


mindspore.mint.nn.AdaptiveMaxPool1d

class mindspore.mint.nn.**AdaptiveMaxPool1d** (*output_size*, *return_indices=False*)

对由多个输入平面组成的输入信号应用 1D 自适应最大池化。

对于任何输入大小，输出大小为 L_{out} 。输出特征的数量等于输入平面的数量。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **output_size** (Union[int, tuple]) - 目标输出的 size L_{out} 。
- **return_indices** (bool, 可选) - 如果为 True，输出最大值的索引，默认值为 False。

输入：

- **input** (Tensor) - 输入特征的 shape 为 (N, C, L_{in}) 或 (C, L_{in}) 。

输出：

Union(Tensor, tuple(Tensor, Tensor))。

- 如果 *return_indices* 为 False，返回 Tensor，其 shape 为 (N, C_{in}, L_{out}) ，数据类型与 *input* 相同。
- 如果 *return_indices* 为 True，则是一个包含了两个 Tensor 的 Tuple，表示计算结果以及生成 max 值的位置。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 中的数据不是 float16、float32、float64。
- **TypeError** - *output_size* 不是 int 或者 tuple。
- **TypeError** - *return_indices* 不是 bool。
- **ValueError** - *output_size* 是 tuple，但大小不是 1。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[2, 1, 2], [2, 3, 5]]), mindspore.float16)
>>> net = mint.nn.AdaptiveMaxPool1d(3)
>>> output = net(input)
>>> print(output)
[[2. 1. 2.]
 [2. 3. 5.]]
```

mindspore.mint.nn.AvgPool2d

class mindspore.mint.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)

对输入张量应用二维平均池化，可视为二维输入平面的组合。

更多参考详见[mindspore.mint.nn.functional.avg_pool2d\(\)](#)。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> input = Tensor(np.arange(1 * 3 * 3 * 4).reshape(1, 3, 3, 4), mstype.float32)
>>> net = mint.nn.AvgPool2d(kernel_size=2, stride=1)
>>> output = net(input)
>>> print(output.shape)
(1, 3, 2, 3)
```

mindspore.mint.nn.AvgPool3d

class mindspore.mint.nn.AvgPool3d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)

对输入张量应用三维平均池化，可视为三维输入平面的组合。

警告： 这是一个实验性 API，后续可能修改或删除。

更多参考详见[mindspore.mint.nn.functional.avg_pool3d\(\)](#)。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> pool = ms.mint.nn.AvgPool3d(kernel_size=3, stride=1)
>>> x = ms.ops.randn(1, 2, 4, 4, 5).astype(ms.float32)
>>> output = pool(x)
>>> print(output.shape)
(1, 2, 2, 2, 3)
>>> x1 = ms.ops.randn(6, 5, 7, 7, 5).astype(ms.float32)
>>> pool2 = ms.mint.nn.AvgPool3d(4, stride=2, padding=(2, 2, 1), divisor_override=10)
>>> output2 = pool2(x1)
>>> print(output2.shape)
(6, 5, 4, 4, 2)
```

mindspore.mint.nn.MaxUnpool2d

class mindspore.mint.nn.MaxUnpool2d (*kernel_size, stride=None, padding=0*)

Maxpool2d 的逆过程。

MaxUnpool2d 在计算过程中，保留最大值位置的元素，并将非最大值位置元素设置为 0。支持的输入数据格式为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) ，输出数据的格式为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) ，计算公式如下：

$$\begin{aligned} H_{out} &= (H_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0] \\ W_{out} &= (W_{in} - 1) \times stride[1] - 2 \times padding[1] + kernel\_size[1] \end{aligned}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。int 类型表示池化核的长宽相同。tuple 类型中的两个值分别代表池化核的长和宽。
- **stride** (Union[int, tuple[int]], 可选) - 池化操作的移动步长，int 类型表示长宽方向的移动步长相同。tuple 中的两个值分别代表长宽方向移动的步长。默认值：None，表示移动步长为 *kernel_size*。
- **padding** (Union[int, tuple[int]], 可选) - 填充值。默认值：0。若为 int 类型，则长宽方向的填充大小相同，均为 *padding*。若为 tuple 类型，则 tuple 中的两个值分别代表长宽方向填充的大小。

输入：

- **input** (Tensor) - 待求逆的 Tensor。shape 为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 *input* 相同。取值范围需满足 $[0, H_{in} \times W_{in} - 1]$ 。数据类型必须是 int32 或 int64。
- **output_size** (tuple[int], 可选) - 输出 shape。默认值：None。如果 *output_size* 为 ()，那么输出 shape 根据 *kernel_size*、*stride* 和 *padding* 计算得出。如果 *output_size* 不为 ()，那么 *output_size* 必须满足格式 (N, C, H, W) ， (C, H, W) 或 (H, W) ，取值范围需满足： $[(N, C, H_{out} - stride[0], W_{out} - stride[1]), (N, C, H_{out} + stride[0], W_{out} + stride[1])]$ 。

输出：

shape 为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) 的 Tensor，数据类型与输入 *input* 相同。

异常：

- **TypeError** - *input* 或 *indices* 的数据类型不支持。
- **TypeError** - *kernel_size*、*stride* 或 *padding* 既不是整数也不是 tuple。
- **ValueError** - *stride*、*padding* 或 *kernel_size* 的值不是非负的。
- **ValueError** - *input* 和 *indices* 的 shape 不一致。
- **ValueError** - *input* 的长度不为 3 或 4。
- **ValueError** - *output_size* 的类型不是 tuple。
- **ValueError** - *output_size* 的取值与根据 *kernel_size*、*stride*、*padding* 计算得到的结果差距太大。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> indices = Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> net = mint.nn.MaxUnpool2d(1, stride=1, padding=0)
>>> output = net(input, indices)
>>> print(output.asnumpy())
[[[0. 1.]
  [8. 9.]]]]
```

4.4.8 填充层

接口名	概述	支持平台
<code>mindspore.mint.nn.ConstantPad1d</code>	根据参数 <i>padding</i> 以及 <i>value</i> 对输入 <i>input</i> 最后 1 维进行填充。	Ascend
<code>mindspore.mint.nn.ConstantPad2d</code>	根据参数 <i>padding</i> 以及 <i>value</i> 对输入 <i>input</i> 最后 2 维进行填充。	Ascend
<code>mindspore.mint.nn.ConstantPad3d</code>	根据参数 <i>padding</i> 以及 <i>value</i> ，对输入 <i>input</i> 最后 3 维进行填充。	Ascend
<code>mindspore.mint.nn.ReflectionPad1d</code>	使用输入边界的反射，对输入的最后 1 维 <i>input</i> 进行填充。	Ascend
<code>mindspore.mint.nn.ReflectionPad2d</code>	使用输入边界的反射，对输入的最后 2 维 <i>input</i> 进行填充。	Ascend
<code>mindspore.mint.nn.ReflectionPad3d</code>	使用输入边界的反射，对输入的最后 3 维 <i>input</i> 进行填充。	Ascend
<code>mindspore.mint.nn.ReplicationPad1d</code>	使用输入边界值，对输入的最后 1 维 <i>input</i> 进行填充。	Ascend
<code>mindspore.mint.nn.ReplicationPad2d</code>	使用输入边界值，对输入的最后 2 维 <i>input</i> 进行填充。	Ascend
<code>mindspore.mint.nn.ReplicationPad3d</code>	使用输入边界值，对输入的最后 3 维 <i>input</i> 进行填充。	Ascend
<code>mindspore.mint.nn.ZeroPad1d</code>	根据参数 <i>padding</i> ，对输入 <i>input</i> 最后 1 维填充零。	Ascend
<code>mindspore.mint.nn.ZeroPad2d</code>	根据参数 <i>padding</i> ，对输入 <i>input</i> 最后 2 维填充零。	Ascend
<code>mindspore.mint.nn.ZeroPad3d</code>	根据参数 <i>padding</i> ，对输入 <i>input</i> 最后 3 维填充零。	Ascend

mindspore.mint.nn.ConstantPad1d

class mindspore.mint.nn.ConstantPad1d (*padding*, *value*)
根据参数 *padding* 以及 *value* 对输入 *input* 最后 1 维进行填充。
更多参考详见mindspore.mint.nn.functional.pad()。

警告：这是一个实验性 API，后续可能修改或删除。

- 参数：
- padding (Union[int, tuple, list]) - 指定填充的大小。

- **value** (Union[int, float]) - 指定填充值。

输入：

- **input** (Tensor) - 输入 Tensor, shape 为 $(N, *)$, 其中 $*$ 表示任意维度。

输出：

Tensor, 填充后的 Tensor。

异常：

- **TypeError** - *padding* 不是一个 integer 或包含 2 个 int 的 tuple 或者 list。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *value* 既不是 int, 也不是 float。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list, 且长度和 Tensor 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> # padding is tuple
>>> padding = (0, 1)
>>> value = 0.5
>>> pad1d = ms.mint.nn.ConstantPad1d(padding, value)
>>> out = pad1d(x)
>>> print(out)
[[[ [1.  1.  1.  1.  0.5]
      [1.  1.  1.  1.  0.5]
      [1.  1.  1.  1.  0.5]]
  [[1.  1.  1.  1.  0.5]
    [1.  1.  1.  1.  0.5]
    [1.  1.  1.  1.  0.5]]]]
>>> print(out.shape)
(1, 2, 3, 5)
>>> # padding is int
>>> padding = 1
>>> value = 0.5
>>> pad1d = ms.mint.nn.ConstantPad1d(padding, value)
>>> out = pad1d(x)
>>> print(out)
[[[ [0.5 1.  1.  1.  1.  0.5]
      [0.5 1.  1.  1.  1.  0.5]
      [0.5 1.  1.  1.  1.  0.5]]
  [[0.5 1.  1.  1.  1.  0.5]
    [0.5 1.  1.  1.  1.  0.5]
    [0.5 1.  1.  1.  1.  0.5]]]]
>>> print(out.shape)
(1, 2, 3, 6)
```

mindspore.mint.nn.ConstantPad2d

class mindspore.mint.nn.ConstantPad2d (*padding*, *value*)

根据参数 *padding* 以及 *value* 对输入 *input* 最后 2 维进行填充。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (Union[int, tuple, list]) - 指定填充的大小。
- **value** (Union[int, float]) - 指定填充值。

输入：

- **input** (Tensor) - 输入 Tensor，shape 为 $(N, *)$ ，其中 $*$ 表示任意维度。

输出：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 不是一个 integer 或包含 4 个 int 的 tuple 或者 list。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *value* 既不是 int，也不是 float。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list，且长度和 Tensor 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> padding = (1, 1, 0, 1)
>>> value = 0.5
>>> pad2d = ms.mint.nn.ConstantPad2d(padding, value)
>>> out = pad2d(x)
>>> print(out)
[[[ [0.5  1.  1.  1.  1.  0.5]
      [0.5  1.  1.  1.  1.  0.5]
      [0.5  1.  1.  1.  1.  0.5]
      [0.5  0.5 0.5 0.5 0.5 0.5]]
  [ [0.5  1.  1.  1.  1.  0.5]
      [0.5  1.  1.  1.  1.  0.5]
      [0.5  1.  1.  1.  1.  0.5]
      [0.5  0.5 0.5 0.5 0.5 0.5]]]]
>>> print(out.shape)
(1, 2, 4, 6)
```

mindspore.mint.nn.ConstantPad3d

class mindspore.mint.nn.ConstantPad3d (*padding*, *value*)

根据参数 *padding* 以及 *value*，对输入 *input* 最后 3 维进行填充。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (Union[int, tuple, list]) - 指定填充的大小。
- **value** (Union[int, float]) - 指定填充值。

输入：

- **input** (Tensor) - 输入 Tensor。shape 为 $(N, *)$ ，其中 $*$ 表示任意维度。

输出：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 不是一个 int，或包含 6 个 int 的 tuple，或 list。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *value* 既不是 int，也不是 float。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list，且长度和 Tensor 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> padding = (1, 1, 0, 1, 1, 0)
>>> value = 0.5
>>> pad3d = ms.mint.nn.ConstantPad3d(padding, value)
>>> out = pad3d(x)
>>> print(out)
[[[ [0.5 0.5 0.5 0.5 0.5 0.5]
      [0.5 0.5 0.5 0.5 0.5 0.5]
      [0.5 0.5 0.5 0.5 0.5 0.5]
      [0.5 0.5 0.5 0.5 0.5 0.5]]
  [ [0.5 1.  1.  1.  1.  0.5]
      [0.5 1.  1.  1.  1.  0.5]
      [0.5 1.  1.  1.  1.  0.5]
      [0.5 0.5 0.5 0.5 0.5 0.5]]
  [ [0.5 1.  1.  1.  1.  0.5]
      [0.5 1.  1.  1.  1.  0.5]
      [0.5 1.  1.  1.  1.  0.5]]]
```

(下页继续)

(续上页)

```
[0.5 0.5 0.5 0.5 0.5 0.5]]]]
>>> print(out.shape)
(1, 3, 4, 6)
```

mindspore.mint.nn.ReflectionPad1d

class mindspore.mint.nn.ReflectionPad1d(padding)

使用输入边界的反射, 对输入的最后 1 维 *input* 进行填充。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **padding** (Union[int, tuple, list]) - 指定填充的大小。

输入:

- **input** (Tensor) - 输入 Tensor, 2D 或 3D。shape 为 (C, W_{in}) 或 (N, C, W_{in}) 。

输出:

Tensor, 填充后的 Tensor。

异常:

- **TypeError** - *padding* 不是一个 integer, 或包含 2 个 int 的 tuple, 或 list。
- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list, 且长度和 Tensor 的维度不匹配。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = ms.Tensor(np.array([[0, 1, 2, 3], [4, 5, 6, 7]]).astype(np.float32))
>>> # x has shape (1, 2, 4)
>>> padding = (3, 1)
>>> # The first and the second dimension of x remain the same.
>>> # The third dimension of x: W_out = W_in + pad_left + pad_right = 4 + 3 + 1 = 8
>>> pad1d = ms.mint.nn.ReflectionPad1d(padding)
>>> out = pad1d(x)
>>> # The shape of out is (1, 2, 8)
>>> print(out)
[[[3. 2. 1. 0. 1. 2. 3. 2.]
  [7. 6. 5. 4. 5. 6. 7. 6.]]]
```


`mindspore.mint.nn.ReflectionPad2d`

class `mindspore.mint.nn.ReflectionPad2d(padding)`

使用输入边界的反射, 对输入的最后 2 维 *input* 进行填充。

更多参考详见`mindspore.mint.nn.functional.pad()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **padding** (`Union[int, tuple, list]`) - 指定填充的大小。

输入:

- **input** (`Tensor`) - 输入 `Tensor`, 3D 或 4D。shape 为 (C, H_{in}, W_{in}) 或 (N, C, H_{in}, W_{in}) 。

输出:

填充后的 `Tensor`。

异常:

- **TypeError** - *padding* 不是一个 integer, 或包含 4 个 int 的 tuple, 或 list。
- **TypeError** - *input* 不是 `Tensor`。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list, 且长度和 `Tensor` 的维度不匹配。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = ms.Tensor(np.array([[[0, 1, 2], [3, 4, 5], [6, 7, 8]]]).astype(np.float32))
>>> # x has shape (1, 3, 3)
>>> padding = (1, 1, 2, 0)
>>> pad2d = ms.mint.nn.ReflectionPad2d(padding)
>>> # The first dimension of x remains the same.
>>> # The second dimension of x: H_out = H_in + pad_up + pad_down = 3 + 1 + 1 = 5
>>> # The third dimension of x: W_out = W_in + pad_left + pad_right = 3 + 2 + 0 = 5
>>> out = pad2d(x)
>>> # The shape of out is (1, 5, 5)
>>> print(out)
[[[7. 6. 7. 8. 7.]
  [4. 3. 4. 5. 4.]
  [1. 0. 1. 2. 1.]
  [4. 3. 4. 5. 4.]
  [7. 6. 7. 8. 7.]]]]
```

mindspore.mint.nn.ReflectionPad3d

class mindspore.mint.nn.ReflectionPad3d(padding)

使用输入边界的反射, 对输入的最后 3 维 *input* 进行填充。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **padding** (Union[int, tuple, list]) - 指定填充的大小。

输入:

- **input** (Tensor) - 输入 Tensor, 4D 或 5D。shape 为 $(N, D_{in}, H_{in}, W_{in})$ 或 $(N, C, D_{in}, H_{in}, W_{in})$ 。

输出:

Tensor, 填充后的 Tensor。

异常:

- **TypeError** - *padding* 不是一个 integer, 或包含 6 个 int 的 tuple, 或 list。
- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list, 且长度和 Tensor 的维度不匹配。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> arr = np.arange(8).astype(np.float32).reshape((1, 2, 2, 2))
>>> x = ms.Tensor(arr)
>>> # x has shape (1, 2, 2, 2)
>>> padding = (1, 1, 1, 0, 0, 1)
>>> pad3d = ms.mint.nn.ReflectionPad3d(padding)
>>> out = pad3d(x)
>>> # The first dimension of x remains the same.
>>> # The second dimension of x: D_out = D_in + pad_front + pad_back = 2 + 0 + 1 = 3
>>> # The third dimension of x: H_out = H_in + pad_up + pad_down = 2 + 1 + 0 = 3
>>> # The last dimension of x: W_out = W_in + pad_left + pad_right = 2 + 1 + 1 = 4
>>> # The shape of out is (1, 3, 3, 4)
>>> print(out)
[[[ [3. 2. 3. 2.]
      [1. 0. 1. 0.]
      [3. 2. 3. 2.]]
  [ [7. 6. 7. 6.]
      [5. 4. 5. 4.]
      [7. 6. 7. 6.]]
  [ [3. 2. 3. 2.]
      [1. 0. 1. 0.]
      [3. 2. 3. 2.]]]]
```

`mindspore.mint.nn.ReplicationPad1d`

class `mindspore.mint.nn.ReplicationPad1d` (*padding*)

使用输入边界值，对输入的最后一维 *input* 进行填充。

更多参考详见`mindspore.mint.nn.functional.pad()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (`Union[int, tuple, list]`) - 指定填充的大小。

输入：

- **input** (`Tensor`) - 输入 `Tensor`，维度为二维或三维。`shape` 为 (C, W_{in}) 或 (N, C, W_{in}) 。

输出：

填充后的 `Tensor`。

异常：

- **TypeError** - *padding* 不是一个 `int`，或包含 2 个 `int` 的 `tuple` 或 `list`。
- **TypeError** - *input* 不是 `Tensor`。
- **ValueError** - *padding* 是 `tuple` 或 `list`，且长度和 `Tensor` 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> pad1d = ms.mint.nn.ReplicationPad1d(2)
>>> input = ms.Tensor(np.arange(0, 8).reshape(1, 2, 4), ms.float32)
>>> print(input)
[[[0. 1. 2. 3.]
  [4. 5. 6. 7.]]]
>>> out = pad1d(input)
>>> print(out)
[[[0. 0. 0. 1. 2. 3. 3. 3.]
  [4. 4. 4. 5. 6. 7. 7. 7.]]]
>>> pad1d = ms.mint.nn.ReplicationPad1d((3, 1))
>>> out = pad1d(input)
>>> print(out)
[[[0. 0. 0. 0. 1. 2. 3. 3.]
  [4. 4. 4. 4. 5. 6. 7. 7.]]]
```

mindspore.mint.nn.ReplicationPad2d

class mindspore.mint.nn.ReplicationPad2d (*padding*)

使用输入边界值，对输入的最后 2 维 *input* 进行填充。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (Union[int, tuple, list]) - 指定填充的大小。

输入：

- **input** (Tensor) - 输入 Tensor，3D 或 4D。shape 为 (C, H_{in}, W_{in}) 或 (N, C, H_{in}, W_{in}) 。

输出：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 不是一个 integer，或包含 4 个 int 的 tuple，或 list。
- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *padding* 是 tuple 或 list，且长度和 Tensor 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> pad2d = ms.mint.nn.ReplicationPad2d(2)
>>> input = ms.Tensor(np.arange(0, 9).reshape(1, 1, 3, 3), ms.float32)
>>> print(input)
[[[0. 1. 2.]
  [3. 4. 5.]
  [6. 7. 8.]]]]
>>> out = pad2d(input)
>>> print(out)
[[[0. 0. 0. 1. 2. 2. 2.]
  [0. 0. 0. 1. 2. 2. 2.]
  [0. 0. 0. 1. 2. 2. 2.]
  [3. 3. 3. 4. 5. 5. 5.]
  [6. 6. 6. 7. 8. 8. 8.]
  [6. 6. 6. 7. 8. 8. 8.]
  [6. 6. 6. 7. 8. 8. 8.]]]]
>>> pad2d = ms.mint.nn.ReplicationPad2d((1, 1, 2, 0))
>>> out = pad2d(input)
>>> print(out)
[[[0. 0. 1. 2. 2.]
  [0. 0. 1. 2. 2.]
  [0. 0. 1. 2. 2.]
  [3. 3. 4. 5. 5.]
  [6. 6. 7. 8. 8.]]]]
```

`mindspore.mint.nn.ReplicationPad3d`

class `mindspore.mint.nn.ReplicationPad3d` (*padding*)

使用输入边界值，对输入的最后 3 维 *input* 进行填充。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (`Union[int, tuple, list]`) - 指定填充的大小。

输入：

- **input** (`Tensor`) - 输入 `Tensor`，4D 或 5D。shape 为 $(N, D_{in}, H_{in}, W_{in})$ 或 $(N, C, D_{in}, H_{in}, W_{in})$ 。

输出：

`Tensor`，填充后的 `Tensor`。

异常：

- **TypeError** - *padding* 不是一个 integer，或包含 6 个 int 的 tuple，或 list。
- **TypeError** - *input* 不是 `Tensor`。
- **ValueError** - *padding* 是 tuple 或 list，且长度和 `Tensor` 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> pad3d = ms.mint.nn.ReplicationPad3d(1)
>>> input = ms.Tensor(np.arange(0, 9).reshape(1, 1, 1, 3, 3), ms.float32)
>>> out = pad3d(input)
>>> print(out)
[[[[[0. 0. 1. 2. 2.]
      [0. 0. 1. 2. 2.]
      [3. 3. 4. 5. 5.]
      [6. 6. 7. 8. 8.]
      [6. 6. 7. 8. 8.]]
     [0. 0. 1. 2. 2.]
     [0. 0. 1. 2. 2.]
     [3. 3. 4. 5. 5.]
     [6. 6. 7. 8. 8.]
     [6. 6. 7. 8. 8.]]
     [0. 0. 1. 2. 2.]
     [0. 0. 1. 2. 2.]
     [3. 3. 4. 5. 5.]
     [6. 6. 7. 8. 8.]
     [6. 6. 7. 8. 8.]]]]]]
```

mindspore.mint.nn.ZeroPad1d

class mindspore.mint.nn.ZeroPad1d(*padding*)

根据参数 *padding*，对输入 *input* 最后 1 维填充零。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (Union[int, tuple, list]) - 指定填充的大小。

输入：

- **input** (Tensor) - 输入 Tensor。shape 为 $(N, *)$ ，其中 * 表示任意维度。

输出：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 既不是一个 int，也不是包含 2 个 int 的 tuple，或 list。
- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list，且长度和 Tensor 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> # padding is tuple
>>> padding = (0, 1)
>>> pad1d = ms.mint.nn.ZeroPad1d(padding)
>>> out = pad1d(x)
>>> print(out)
[[[ [1.  1.  1.  1.  0.]
      [1.  1.  1.  1.  0.]
      [1.  1.  1.  1.  0.]]
  [[1.  1.  1.  1.  0.]
    [1.  1.  1.  1.  0.]
    [1.  1.  1.  1.  0.]
    [1.  1.  1.  1.  0.]
    [1.  1.  1.  1.  0.]
    [1.  1.  1.  1.  0.]
    [1.  1.  1.  1.  0.]]]]]
>>> print(out.shape)
(1, 2, 3, 5)
>>> # padding is int
>>> padding = 1
>>> pad1d = ms.mint.nn.ZeroPad1d(padding)
>>> out = pad1d(x)
>>> print(out)
[[[ [0. 1.  1.  1.  1.  0.]
```

(下页继续)

(续上页)

```
[0. 1. 1. 1. 1. 0.]
[0. 1. 1. 1. 1. 0.]]
[[0. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]]]]
>>> print(out.shape)
(1, 2, 3, 6)
```

`mindspore.mint.nn.ZeroPad2d`

class `mindspore.mint.nn.ZeroPad2d(padding)`

根据参数 *padding*，对输入 *input* 最后 2 维填充零。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (Union[int, tuple, list]) - 指定填充的大小。

输入：

- **input** (Tensor) - 输入 Tensor，shape 为 $(N, *)$ ，其中 * 表示任意维度。

输出：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 不是一个 integer，或包含 4 个 int 的 tuple，或 list。
- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list，且长度和 Tensor 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> padding = (1, 1, 0, 1)
>>> pad = ms.mint.nn.ZeroPad2d(padding)
>>> out = pad(x)
>>> print(out)
[[[0. 1. 1. 1. 1. 0.]
  [0. 1. 1. 1. 1. 0.]
  [0. 1. 1. 1. 1. 0.]
  [0. 0. 0. 0. 0. 0.]
```

(下页继续)

(续上页)

```
[[0. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]
 [0. 1. 1. 1. 1. 0.]
 [0. 0. 0. 0. 0. 0.]]]]
>>> print(out.shape)
(1, 2, 4, 6)
```

mindspore.mint.nn.ZeroPad3d

class mindspore.mint.nn.ZeroPad3d(padding)

根据参数 *padding*，对输入 *input* 最后 3 维填充零。

更多参考详见 `mindspore.mint.nn.functional.pad()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **padding** (Union[int, tuple, list]) - 指定填充的大小。

输入：

- **input** (Tensor) - 输入 Tensor，shape 为 $(N, *)$ ，其中 * 表示任意维度。

输出：

Tensor，填充后的 Tensor。

异常：

- **TypeError** - *padding* 不是一个 integer，或包含 6 个 int 的 tuple，或 list。
- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *padding* 含有负数。
- **ValueError** - *padding* 是 tuple 或 list，且长度和 Tensor 的维度不匹配。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> x = np.ones(shape=(1, 2, 3, 4)).astype(np.float32)
>>> x = ms.Tensor(x)
>>> padding = (1, 1, 0, 1, 1, 0)
>>> pad3d = ms.mint.nn.ZeroPad3d(padding)
>>> out = pad3d(x)
>>> print(out)
[[[0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0.]
  [0. 0. 0. 0. 0. 0.]]
 [[0. 1. 1. 1. 1. 0.]
```

(下页继续)

(续上页)

```
[0. 1.  1.  1.  1.  0.]
[0. 1.  1.  1.  1.  0.]
[0. 0.  0.  0.  0.  0.]]
[[[0. 1.  1.  1.  1.  0.]
  [0. 1.  1.  1.  1.  0.]
  [0. 1.  1.  1.  1.  0.]
  [0. 0.  0.  0.  0.  0.]]]]
>>> print(out.shape)
(1, 3, 4, 6)
```

4.4.9 损失函数

接口名	概述	支持平台
<code>mindspore.mint.nn.BCELoss</code>	计算目标值和预测值之间的二值交叉熵损失值。	Ascend
<code>mindspore.mint.nn.BCEWithLogitsLoss</code>	输入 <i>input</i> 经过 <code>sigmoid</code> 激活函数后作为预测值，用此预测值计算和目标值之间的二值交叉熵损失。	Ascend
<code>mindspore.mint.nn.CrossEntropyLoss</code>	获取预测值和目标值之间的交叉熵损失。	Ascend
<code>mindspore.mint.nn.KLDivLoss</code>	计算输入 <i>input</i> 和 <i>target</i> 的 Kullback-Leibler 散度。	Ascend
<code>mindspore.mint.nn.L1Loss</code>	<code>L1Loss</code> 用于计算预测值和目标值之间的平均绝对误差。	Ascend
<code>mindspore.mint.nn.MSELoss</code>	用于计算预测值与标签值之间的均方误差。	Ascend
<code>mindspore.mint.nn.NLLLoss</code>	获取预测值和目标值之间的负对数似然损失。	Ascend
<code>mindspore.mint.nn.SmoothL1Loss</code>	计算平滑 L1 损失，该 L1 损失函数有稳健性。	Ascend

mindspore.mint.nn.BCELoss

class mindspore.mint.nn.BCELoss (weight=None, reduction='mean')

计算目标值和预测值之间的二值交叉熵损失值。

将预测值设置为 *x*，目标值设置为 *y*，输出损失设置为 $\ell(x, y)$ 。

则公式如下：

$$L = \{l_1, \dots, l_n, \dots, l_N\}^T, \quad l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

其中 *N* 是批次大小。公式如下：

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

说明：预测值一般是 `sigmoid` 函数的输出。因为是二分类，所以目标值应是 0 或者 1。如果 *x_n* 是 0 或 1，则上述损失函数是无意义的。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。与输入数据的 **shape** 和数据类型相同。默认值：`None`。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 `'none'`、`'mean'`、`'sum'`，默认值：`'mean'`。
 - `"none"`：不应用规约方法。
 - `"mean"`：计算输出元素的加权平均值。
 - `"sum"`：计算输出元素的总和。

输入：

- **input** (Tensor) - 预测值 Tensor，**shape** $(N,*)$ ，其中 $*$ 代表任意数量的附加维度。数据类型必须为 `float16`、`float32` 或 `bfloat16`（仅 Atlas A2 训练系列产品支持）。
- **target** (Tensor) - 目标值 Tensor，**shape** $(N,*)$ ，其中 $*$ 代表任意数量的附加维度。与 *input* 的 **shape** 和数据类型相同。

输出：

Tensor，数据类型与 *input* 相同。如果 *reduction* 为 `'none'`，则 **shape** 与 *input* 相同。否则，输出为 `Scalar` 的 Tensor。

异常：

- **TypeError** - *input*，*target* 或 *weight*（如果给定）的数据类型不是 `float16`、`float32` 或 `bfloat16`。
- **ValueError** - *reduction* 不为 `'none'`、`'mean'` 或 `'sum'`。
- **ValueError** - *input* 的 **shape** 与 *target* 或 *weight*（如果给定）不同。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> import numpy as np
>>> weight = ms.Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 3.3, 2.2]]), ms.float32)
>>> loss = mint.nn.BCELoss(weight=weight, reduction='mean')
>>> input = ms.Tensor(np.array([[0.1, 0.2, 0.3], [0.5, 0.7, 0.9]]), ms.float32)
>>> target = ms.Tensor(np.array([[0, 1, 0], [0, 0, 1]]), ms.float32)
>>> output = loss(input, target)
>>> print(output)
1.8952923
```

mindspore.mint.nn.BCEWithLogitsLoss

class mindspore.mint.nn.BCEWithLogitsLoss (*weight=None, reduction='mean', pos_weight=None*)

输入 *input* 经过 sigmoid 激活函数后作为预测值, 用此预测值计算和目标值之间的二值交叉熵损失。

将输入 *input* 设置为 X , 输入 *target* 为 Y , 输出为 L 。则公式如下:

$$p_{ij} = \text{sigmoid}(X_{ij}) = \frac{1}{1 + e^{-X_{ij}}}$$

$$L_{ij} = -[Y_{ij} \cdot \log(p_{ij}) + (1 - Y_{ij}) \cdot \log(1 - p_{ij})]$$

然后,

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数:

- **weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。如果不是 `None`, 其 `shape` 需要能广播到与 *target* 的 `shape` 保持一致, 数据类型必须为 `float16`、`float32` 或 `bfloat16` (仅 Atlas A2 训练系列产品支持)。默认值: `None`。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 `'none'`、`'mean'`、`'sum'`, 默认值: `'mean'`。
 - `'none'`: 不应用规约方法。
 - `'mean'`: 计算输出元素的加权平均值。
 - `'sum'`: 计算输出元素的总和。
- **pos_weight** (Tensor, 可选) - 指定正样本的权重。是一个长度等于分类数的向量。如果不是 `None`, 其 `shape` 需要能广播到与 *target* 的 `shape` 保持一致, 数据类型必须为 `float16`、`float32` 或 `bfloat16` (仅 Atlas A2 训练系列产品支持)。默认值: `None`。

输入:

- **input** (Tensor) - 输入预测值 Tensor, `shape (N,*)`, 其中 `*` 代表任意数量的附加维度。其数据类型为 `float16`、`float32` 或 `bfloat16` (仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 输入目标值 Tensor, `shape (N,*)`, 其中 `*` 代表任意数量的附加维度。与 *input* 的 `shape` 和数据类型相同。

输出:

Tensor 或 Scalar, 如果 *reduction* 为 `'none'`, 其 `shape` 需和 *input* 相同。否则, 将返回 Scalar。

异常:

- **TypeError** - *input* 或 *target* 的不为 Tensor。
- **TypeError** - *weight* 或 *pos_weight* 是 Parameter。
- **TypeError** - *reduction* 的数据类型不是 string。
- **ValueError** - *weight* 或 *pos_weight* 不能广播到 `shape` 为 *input* 的 Tensor。
- **ValueError** - *reduction* 不为 `'none'`、`'mean'` 或 `'sum'`。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> import numpy as np
>>> input = ms.Tensor(np.array([[[-0.8, 1.2, 0.7], [-0.1, -0.4, 0.7]]]).astype(np.float32))
>>> target = ms.Tensor(np.array([[0.3, 0.8, 1.2], [-0.6, 0.1, 2.2]]).astype(np.float32))
>>> loss = mint.nn.BCEWithLogitsLoss()
>>> output = loss(input, target)
>>> print(output)
0.3463612
```

mindspore.mint.nn.CrossEntropyLoss

class mindspore.mint.nn.**CrossEntropyLoss** (weight=None, ignore_index=-100, reduction='mean', label_smoothing=0.0)

获取预测值和目标值之间的交叉熵损失。

cross_entropy 方法支持两种不同的目标值 (target):

- 类别索引 (int), 取值范围为 $[0, C)$, 其中 C 为类别数, 当 reduction 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n, y_n})}{\sum_{c=1}^C \exp(x_{n, c})} \cdot \mathbb{1}\{y_n \neq \text{ignore\_index}\}$$

其中, x 表示预测值, y 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 reduction 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{1}{\sum_{n=1}^N w_{y_n} \cdot \mathbb{1}\{y_n \neq \text{ignore\_index}\}} \sum_{n=1}^N l_n, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

- 类别概率 (float), 用于目标值为多个类别标签的情况。当 reduction 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = - \sum_{c=1}^C w_c \log \frac{\exp(x_{n, c})}{\sum_{i=1}^C \exp(x_{n, i})} y_{n, c}$$

其中, x 表示预测值, y 表示目标值, w 表示权重, N 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 reduction 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N l_n}{N}, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

说明: 动态 shape、动态 rank 和可变量输入不支持在 严格图模式 (jit_syntax_level=mindspore.STRICT) 下执行。

参数:

- **weight** (Tensor, 可选) - 指定各类别的权重。若值不为 None, 则 **shape** 为 $(C,)$ 。数据类型仅支持 float16、float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。默认值: None。
- **ignore_index** (int, 可选) - 指定 **target** 中需要忽略的值(一般为填充值), 使其不对梯度产生影响。仅在目标值为类别索引下生效, 在类别概率下请设置为负数。默认值: -100。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。
- **label_smoothing** (float, 可选) - 标签平滑值。用于计算 Loss 时, 防止模型过拟合的正则化手段。取值范围为 $[0.0, 1.0]$ 。默认值: 0.0。

输入:

- **input** (Tensor) - 输入预测值。**shape** 为 (N) 、 (N, C) 或 (N, C, H, W) (针对二维数据), 或 $(N, C, d_1, d_2, \dots, d_K)$ (针对高维数据)。**input** 需为对数概率。数据类型仅支持 float16、float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 输入目标值。若目标值为类别索引, 则 **shape** 为 $()$ 、 (N) 或 $(N, d_1, d_2, \dots, d_K)$, 数据类型仅支持 int32 或 int64。若目标值为类别概率, 则 **shape** 为 $(N,)$ 、 (N, C) 或 $(N, C, d_1, d_2, \dots, d_K)$, 数据类型仅支持 float16、float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。

输出:

Tensor, 数据类型与 *input* 相同。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> # Case 1: Indices labels
>>> inputs = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> target = ms.Tensor(np.array([1, 0, 4]), ms.int32)
>>> op = ms.mint.nn.CrossEntropyLoss()
>>> output = op(inputs, target)
>>> # Case 2: Probability labels
>>> inputs = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> target = ms.Tensor(np.random.randn(3, 5), ms.float32)
>>> op = ms.mint.nn.CrossEntropyLoss()
>>> output = op(inputs, target)
```

mindspore.mint.nn.KLDivLoss

class mindspore.mint.nn.KLDivLoss (reduction='mean', log_target=False)

计算输入 *input* 和 *target* 的 Kullback-Leibler 散度。

对于相同 shape 的 Tensor *x* 和 *y* , KLDivLoss 的计算公式如下:

$$L(x, y) = y \cdot (\log y - x)$$

则

$$\ell(x, y) = \begin{cases} L(x, y), & \text{if reduction = 'none';} \\ \text{mean}(L(x, y)), & \text{if reduction = 'mean';} \\ \text{sum}(L(x, y))/x.\text{shape}[0], & \text{if reduction = 'batchmean';} \\ \text{sum}(L(x, y)), & \text{if reduction = 'sum'}. \end{cases}$$

其中 *x* 表示 *input* , *y* 表示 *target* , $\ell(x, y)$ 表示输出。

说明: 仅当 *reduction* 设置为 "batchmean" 时, 输出才与 Kullback-Leibler 散度的数学定义一致。

参数:

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式。默认值: 'mean' 。
- **log_target** (bool, 可选) - 指定输入 *target* 是否在对数空间内。默认值: False 。

输入:

- **input** (Tensor) - 输入 Tensor。数据类型必须为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 目标 Tensor, 其数据类型与 *input* 相同。*target* 的 shape 和 *input* 的 shape 能广播。

输出:

Tensor, 数据类型与 *input* 相同。如果 *reduction* 为 'none' , 则 shape 与 *input* 和 *target* 广播之后的结果相同。否则, 输出为 Scalar 的 Tensor。

异常:

- **TypeError** - *input* 或 *target* 不是 Tensor。
- **TypeError** - *input* 或 *target* 的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - *target* 的数据类型与 *input* 不同。
- **ValueError** - *reduction* 不为 'none'、'mean'、'sum' 或 'batchmean' 之一。
- **ValueError** - *target* 的 shape 和 *input* 的 shape 不能广播。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> import numpy as np
>>> input = ms.Tensor(np.array([[0.5, 0.5], [0.4, 0.6]]), ms.float32)
>>> target = ms.Tensor(np.array([[0., 1.], [1., 0.]]), ms.float32)
```

(下页继续)

(续上页)

```
>>> loss = mint.nn.KLDivLoss(reduction='mean', log_target=False)
>>> output = loss(input, target)
>>> print(output)
-0.225
```

mindspore.mint.nn.L1Loss

class mindspore.mint.nn.L1Loss(reduction='mean')

L1Loss 用于计算预测值和目标值之间的平均绝对误差。

假设 x 和 y 为一维 Tensor，长度 N ，则计算 x 和 y 的 loss 而不进行降维操作（即 reduction 参数设置为 'none'）。

公式如下：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad \text{with } l_n = |x_n - y_n|,$$

其中， N 为 batch size。如果 reduction 不是 'none'，则：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = 'mean'}; \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

输入：

- **logits** (Tensor) - 预测值，任意维度的 Tensor。
- **labels** (Tensor) - 目标值，通常情况下与 logits 的 shape 相同。但是如果 logits 和 labels 的 shape 不同，需要保证他们之间可以互相广播。

输出：

Tensor，类型为 float。

异常：

- **ValueError** - reduction 不为 'mean'、'sum' 或 'none'。
- **ValueError** - logits 和 labels 有不同的 shape，且不能互相广播。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> loss = mint.nn.L1Loss()
```

(下页继续)

(续上页)

```

>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
0.33333334
>>> # Case 2: logits.shape = (3,), labels.shape = (2, 3)
>>> loss = mint.nn.L1Loss(reduction='none')
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
[[0. 1. 2.]
 [0. 0. 1.]]

```

mindspore.mint.nn.MSELoss

class mindspore.mint.nn.MSELoss (*reduction='mean'*)

用于计算预测值与标签值之间的均方误差。

假设 x 和 y 为一维 Tensor，长度 N ，则计算 x 和 y 的 loss（即 *reduction* 参数设置为 'none'）的公式如下：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad \text{with } l_n = (x_n - y_n)^2.$$

其中， N 为 batch size。如果 *reduction* 不是 'none'，则：

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction = 'mean'}; \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - 'none'：不应用规约方法。
 - 'mean'：计算输出元素的平均值。
 - 'sum'：计算输出元素的总和。

输入：

- **logits** (Tensor) - 输入预测值，任意维度的 Tensor。数据类型和 *labels* 需要保持一致。
- **labels** (Tensor) - 输入标签，任意维度的 Tensor。数据类型和 *logits* 需要保持一致。支持在 *logits* 和 *labels* shape 不相同的情况下，通过广播保持一致。

输出：

- Tensor。如果 *reduction* 为 'mean' 或 'sum' 时，输出的 shape 为 *Tensor Scalar*。
- 如果 *reduction* 为 'none'，输出的 shape 则是 *logits* 和 *labels* 广播之后的 shape。

异常：

- **ValueError** - *reduction* 不为 'mean'，'sum' 或 'none' 中的一个。
- **ValueError** - *logits* 和 *labels* 的 shape 不能广播。
- **TypeError** - 如果 *logits* 和 *labels* 数据类型不一致。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> # Case 1: logits.shape = labels.shape = (3,)
>>> loss = mint.nn.MSELoss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 1, 1]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
1.6666667
>>> # Case 2: logits.shape = (3,), labels.shape = (2, 3)
>>> loss = mint.nn.MSELoss(reduction='none')
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
[[0. 1. 4.]
 [0. 0. 1.]]
```

`mindspore.mint.nn.NLLLoss`

class `mindspore.mint.nn.NLLLoss` (*weight=None, ignore_index=-100, reduction='mean'*)

获取预测值和目标值之间的负对数似然损失。

`reduction` 为 'none' 时, 负对数似然损失公式如下:

$$\ell(x, t) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{t_n} x_{n, t_n}, \quad w_c = \text{weight}[c] \cdot \mathbb{1}\{c \neq \text{ignore\_index}\},$$

其中, x 表示预测值, t 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 `reduction` 不为 'none' (默认为 'mean'), 则

$$\ell(x, t) = \begin{cases} \sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{t_n}} l_n, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'} \end{cases}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **weight** (Tensor, 可选) - 指定各类别的权重。若值不为 None, 则 `shape` 为 $(C,)$ 。数据类型仅支持 float16、float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。要求与 `input` 的数据类型保持一致。默认值: None。
- **ignore_index** (int, 可选) - 指定 `target` 中需要忽略的值 (一般为填充值), 使其不对梯度产生影响。仅在目标值为类别索引下生效, 在类别概率下请设置为负数。默认值: -100。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。

- 'sum': 计算输出元素的总和。

输入:

- **input** (Tensor) - 输入预测值, shape 为 (N) 或 (N, C) , C 表示类的数量, N 表示 batch size, 或 $(N, C, d_1, d_2, \dots, d_K)$ (针对高维数据)。input 需为对数概率。数据类型仅支持 float16、float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 输入目标值, shape 为 $()$ 或 (N) , 限定范围为 $[0, C - 1]$, 或 $(N, d_1, d_2, \dots, d_K)$ (针对高维数据)。数据类型支持 int32、int64、uint8。

输出:

Tensor, 数据类型与 input 相同。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> inputs = mindspore.Tensor(np.random.randn(3, 5), mindspore.float32)
>>> target = mindspore.Tensor(np.array([1, 0, 4]), mindspore.int32)
>>> op = mindspore.mint.nn.NLLLoss()
>>> output = op(inputs, target)
```

mindspore.mint.nn.SmoothL1Loss

class mindspore.mint.nn.SmoothL1Loss (reduction='mean', beta=1.0)

计算平滑 L1 损失, 该 L1 损失函数有稳健性。

警告: 这是一个实验性 API, 后续可能修改或删除。

参见 `mindspore.mint.nn.functional.smooth_l1_loss()` 获取更多详情。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([2, 2, 3]), mindspore.float32)
>>> target = Tensor(np.array([2, 2, 2]), mindspore.float32)
>>> beta = 1.0
>>> reduction_1 = 'none'
>>> loss1 = mint.nn.SmoothL1Loss(reduction=reduction_1, beta=beta)
>>> output = loss1(input, target)
>>> print(output)
[0.  0.  0.5]
>>> reduction_2 = 'mean'
>>> loss2 = mint.nn.SmoothL1Loss(reduction=reduction_2, beta=beta)
```

(下页继续)

(续上页)

```
>>> output = loss2(input, target)
>>> print(output)
0.16666667
>>> reduction_3 = 'sum'
>>> loss3 = mint.nn.SmoothL1Loss(reduction=reduction_3, beta=beta)
>>> output = loss3(input, target)
>>> print(output)
0.5
```

4.4.10 Vision 层

接口名	概述	支持平台
<code>mindspore.mint.nn.PixelShuffle</code>	根据上采样系数重新排列 Tensor 中的元素。	Ascend
<code>mindspore.mint.nn.Upsample</code>	按照给定的 <i>size</i> 或 <i>scale_factor</i> 根据 <i>mode</i> 设置的插值方式，对输入 <i>input</i> 进行插值。	Ascend

mindspore.mint.nn.PixelShuffle

class mindspore.mint.nn.PixelShuffle(*upscale_factor*)
根据上采样系数重新排列 Tensor 中的元素。
更多参考详见`mindspore.mint.nn.functional.pixel_shuffle()`。
支持平台：
Ascend
样例：

```
>>> from mindspore import mint
>>> pixel_shuffle = mint.nn.PixelShuffle(3)
>>> input = mint.randn(1, 9, 4, 4)
>>> output = pixel_shuffle(input)
>>> print(output.shape())
[1, 1, 12, 12]
```

mindspore.mint.nn.Upsample

class mindspore.mint.nn.Upsample(*size=None, scale_factor=None, mode='nearest, align_corners=None, recompute_scale_factor=None*)
按照给定的 *size* 或 *scale_factor* 根据 *mode* 设置的插值方式，对输入 *input* 进行插值。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见`mindspore.mint.nn.functional.interpolate()`。
支持平台：
Ascend
样例：

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> x = ms.Tensor([[[[1.0, 2.0, 3.0, 4.0], [5.0, 6.0, 7.0, 8.0]]]])
>>> upsample = mint.nn.Upsample(size=(5, 5))
>>> out = upsample(x)
>>> print(x.asnumpy())
[[[1. 2. 3. 4.]
  [5. 6. 7. 8.]]]
>>> print(out.asnumpy())
[[[1. 1. 2. 3. 4.]
  [1. 1. 2. 3. 4.]
  [1. 1. 2. 3. 4.]
  [5. 5. 6. 7. 8.]
  [5. 5. 6. 7. 8.]]]
>>> print(out.shape)
(1, 1, 5, 5)
```

4.4.11 工具

接口名	概述	支持平台
<code><i>mindspore.mint.nn.Identity</i></code>	网络占位符，返回与输入完全一致。	Ascend

`mindspore.mint.nn.Identity`

class `mindspore.mint.nn.Identity(*args, **kwargs)`

网络占位符，返回与输入完全一致。

参数：

- **args** (Any) - 任意参数。
- **kwargs** (Any) - 任意关键字参数。

输入：

- **input** (Any) - Identity 的输入。

输出：

与 *input* 完全相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([1, 2, 3, 4]), mindspore.int64)
>>> net = mint.nn.Identity()
>>> output = net(input)
>>> print(output)
[1 2 3 4]
```

4.5 mindspore.mint.nn.functional

4.5.1 卷积函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.conv2d</code>	对输入 Tensor 计算二维卷积。	Ascend
<code>mindspore.mint.nn.functional.conv3d</code>	对输入 Tensor 计算三维卷积。	Ascend
<code>mindspore.mint.nn.functional.conv_transpose2d</code>	将 2D 转置卷积运算应用于由多个输入平面组成的输入图像，有时也称为反卷积（尽管它不是实际的反卷积）。	Ascend
<code>mindspore.mint.nn.functional.fold</code>	将提取出的滑动局部区域块还原成更大的输出 Tensor。	Ascend
<code>mindspore.mint.nn.functional.unfold</code>	从一个 batch 的输入 Tensor 中提取滑动局部块。	Ascend

mindspore.mint.nn.functional.conv2d

`mindspore.mint.nn.functional.conv2d` (*input*, *weight*, *bias*=None, *stride*=1, *padding*=0, *dilation*=1, *groups*=1)

对输入 Tensor 计算二维卷积。通常输入的 shape 为 $(N, C_{in}, H_{in}, W_{in})$ 或 (C_{in}, H_{in}, W_{in}) ，其中 N 为 batch size， C 为通道数， H 为特征图的高度， W 为特征图的宽度。

根据以下公式计算输出：

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} ccor(weight(C_{out_j}, k), X(N_i, k))$$

其中，*bias* 为输出偏置，*ccor* 为 [cross-correlation](#) 操作，*weight* 为卷积核的值，*X* 为输入的特征图。

i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。

j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。

k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中，*bias*(C_{out_j}) 为第 *j* 个输出通道的偏置，*weight*(C_{out_j}, k) 表示第 *j* 个卷积核在第 *k* 个输入通道的卷积核切片，*X*(N_i, k) 为特征图第 *i* 个 batch 第 *k* 个输入通道的切片。卷积核 shape 为 (kernel_size[0], kernel_size[1])，其中 kernel_size[0] 和 kernel_size[1] 是卷积核的高度和宽度。若考虑到输入输出通道以及 *groups*，则完整卷积核的 shape 为 $(C_{out}, C_{in}/groups, kernel\_size[0], kernel\_size[1])$ ，其中 *groups* 是分组卷积时在通道上分割输入 *x* 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#) 以及 [ConvNets](#)。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- input** (Tensor) - shape 为 $(N, C_{in}, H_{in}, W_{in})$ 或 (C_{in}, H_{in}, W_{in}) 的 Tensor。
- weight** (Tensor) - shape 为 $(C_{out}, C_{in}/groups, kernel\_size[0], kernel\_size[1])$ ，则卷积核的大小为 (kernel_size[0], kernel_size[1])。
- bias** (Tensor, 可选) - 偏置 Tensor，shape 为 (C_{out}) 的 Tensor。如果 *bias* 是 None，将不会添加偏置。默认值：None。

- **stride** (Union(int, tuple[int], list[int]), 可选) - 卷积核移动的步长, 数据类型为 int, 或者由 2 个 int 组成的 tuple。一个 int 表示在高度和宽度方向的移动步长均为该值。两个 int 组成的 tuple 分别表示在高度和宽度方向的移动步长。默认值: 1。
- **padding** (Union[int, tuple[int], list[int], str], 可选) - 输入的高度和宽度方向上填充的数量。数据类型为 int 或包含 2 个整数的 tuple 或 string { "valid", "same" }。如果 padding 是一个整数, 那么 padding_{H} 和 padding_{W} 都等于 padding。如果 padding 是一个有 2 个整数的 tuple, 那么 padding_{H} 和 padding_{W} 分别等于 padding[0] 和 padding[1]。值应该要大于等于 0。默认值: 0。
 - "same": 在输入的四周填充, 使得当 stride 为 1 时, 输入和输出的 shape 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在底部/右侧。若设置该模式, stride 的值必须为 1。
 - "valid": 不对输入进行填充, 返回输出可能的最大高度和宽度, 不能构成一个完整 stride 的额外的像素将被丢弃。
- **dilation** (Union(int, tuple[int], list[int]), 可选) - 卷积核元素间的间隔。数据类型为 int 或由 2 个 int 组成的 tuple。若 $k > 1$, 则卷积核间隔 k 个元素进行采样。垂直和水平方向上的 k , 其取值范围分别为 [1, H] 和 [1, W]。默认值: 1。
- **groups** (int, 可选) - 将过滤器拆分为组。默认值: 1。
 - $(C_{in} \% groups == 0)$, $(C_{out} \% groups == 0)$, $(C_{out} \geq groups)$, $(kernel\_size[1] = C_{in}/groups)$ 。

返回:

Tensor, 卷积后的值。shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。要了解不同的填充模式如何影响输出 shape, 请参考 `mindspore.mint.nn.Conv2d` 以获取更多详细信息。

异常:

- **ValueError** - 输入特征图的大小与参数应满足输出公式, 以确保输出特征图大小为正, 否则会报错。输出公式请参考 `mindspore.mint.nn.Conv2d` 以获取更多详细信息。
- **RuntimeError** - Ascend 上受不同型号 NPU 芯片上 L1 缓存大小限制, 用例尺寸或 Kernel Size 不能过大。
- **TypeError** - 如果 `in_channels`, `out_channels` 或者 `groups` 不是 int。
- **TypeError** - 如果 `kernel_size`, `stride` 或 `dilation` 既不是 int 也不是 tuple。
- **TypeError** - 如果 `bias` 不是 Tensor。
- **ValueError** - 如果 `bias` 的 shape 不是 (C_{out}) 。
- **ValueError** - 如果 `stride` 或 `dilation` 小于 1。
- **ValueError** - 如果 `padding` 是 "same", `stride` 不等于 1。
- **ValueError** - 输入参数不满足卷积输出公式。
- **ValueError** - `kernel_size` 不能超过输入特征图的大小。
- **ValueError** - `padding` 值不能导致计算区域超出输入大小。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint, mint
>>> x = Tensor(np.ones([10, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> output = mint.nn.functional.conv2d(x, weight)
>>> print(output.shape)
(10, 32, 30, 30)
```

mindspore.mint.nn.functional.conv3d

`mindspore.mint.nn.functional.conv3d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)`

对输入 Tensor 计算三维卷积。通常，输入 Tensor 的 shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ 或 $(C_{in}, D_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C 为通道数， D, H, W 分别为特征图的深度、高度和宽度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中， bias 为输出偏置， ccor 为 `cross-correlation` 操作， weight 为卷积核的值， X 为输入的特征图。

以下是索引的含义：

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中， $\text{bias}(C_{out_j})$ 为第 j 个输出通道的偏置， $\text{weight}(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。

卷积核 shape 为 (kd, kh, kw) ，其中 kd 、 kh 和 kw 分别是卷积核的深度、高度和宽度。若考虑到输入输出通道以及 groups ，则完整卷积核的 shape 为 $(C_{out}, C_{in}/\text{groups}, kd, kh, kw)$ ，其中 groups 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

以下罗列参数的一些限制条件。

- `input-conv3d` 的输入。输入每个维度大小必须在 $[1, \text{int32\_max}]$ 范围内。
- `weight-shape` 为 $(C_{out}, C_{in}/\text{groups}, kd, kh, kw)$ 的滤波器。 kh 和 kw 的值在 $[1, 511]$ 范围内。其余值在 $[1, \text{int32\_max}]$ 范围内。并且 $kh * kw * k0$ 必须小于 65536（其中 $k0$ 为 16。如果数据类型是 `float32`，则 $k0$ 为 8）。
- `bias`—形状为 (C_{out}) 的偏置张量。其形状必须与权重的第一维相等。
- `stride`—内核移动的步幅。它可以是一个整数或一个元组（表示为 $(\text{stride}_d, \text{stride}_h, \text{stride}_w)$ ）。其中， stride_h 和 stride_w 的范围是 $[1, 63]$ ， stride_d 的范围是 $[1, 255]$ 。
- `padding`—如果 `padding` 是一个整数，则其范围为 $[0, 255]$ 。
- `dilation`—该值的范围是 $[1, 255]$ 。
- `groups`—该值的范围是 $[1, 65535]$ 。
- $C_{in} \% \text{groups} == 0$ and $C_{out} \% \text{groups} == 0$ 。
- $\text{weight}[1] == C_{in} / \text{groups}$ 。
- $H_{in} + \text{PadUp} + \text{PadDown} \geq (kh - 1) * \text{DilationH} + 1$ 。
- $W_{in} + \text{PadLeft} + \text{PadRight} \geq (kw - 1) * \text{DilationW} + 1$ 。
- $D_{in} + \text{PadFront} + \text{PadBack} \geq (kd - 1) * \text{DilationD} + 1$ 。

- $H_{out} = (H_{in} + PadUp + PadDown - ((kh - 1) * DilationH + 1)) / StrideH + 1$ 。
- $W_{out} = (W_{in} + PadLeft + PadRight - ((kw - 1) * DilationW + 1)) / StrideW + 1$ 。
- $D_{out} = (D_{in} + PadFront + PadBack - ((kd - 1) * DilationD + 1)) / StrideD + 1$ 。
- $(D_{in} + PadFront + PadBack - ((kd - 1) * DilationD + 1)) \% StrideD \leq PadBack$ 。
- $(H_{in} + PadUp + PadDown - ((kh - 1) * Dilationh + 1)) \% StrideH \leq PadDown$ 。
- $stride_d \leq kernel_d$ 。
- $PadUp < kh$ 且 $PadDown < kh$ 。当 $padding = 'valid'$ 时, $PadUp$ 和 $PadDown$ 取值是 0。当 $padding = 'same'$ 时, 对于 $high$ 维度的 $PadUp$ 能通过 $\text{floor}(((H_{out} - 1) * strideH + (kh - 1) * DilationH + 1 - H_{in}) / 2)$ 计算得到。用类似的方法能计算得到关于 $depth$ 和 $width$ 维度的 $padding$ 值。且 $depth$ 和 $width$ 维度也有相同的约束条件。
- $((kh - 1) * DilationH - PadUp)$ 取值区间为 $[0, 255]$ 。深度和宽度维度具有相同的约束。
- 如果 $padding$ 为 $'same'$, $stride$ 必须为 1。

警告: 该 API 不支持 Atlas 系列产品。这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor。
- **weight** (Tensor) - shape 为 $(C_{out}, C_{in}/groups, kd, kh, kw)$, 则卷积核的大小为 (kd, kh, kw) 。
- **bias** (Tensor, 可选) - 偏置 Tensor, shape 为 (C_{out}) 的 Tensor。如果 $bias$ 是 None, 将不会添加偏置。默认值: None。
- **stride** (Union(int, tuple[int], list[int]), 可选) - 卷积核移动的步长, 可以为单个 int 或三个 int 组成的 tuple。一个 int 表示在深度、高度和宽度方向的移动步长均为该值。三个 int 组成的 tuple 分别表示在深度、高度和宽度方向的移动步长。默认值: 1。
- **padding** (Union(int, tuple[int], list[int], str), 可选) - 输入 x 两侧的隐式填充。可以是字符串、一个整数或包含 3 个整数的元组/列表。如果 $padding$ 是一个字符串, 则可选值为 $same$ 、 $valid$ 。
 - $"same"$: 采用完成方式。输出的高度和宽度将等于输入 x 除以步幅。填充将尽可能在顶部和底部、左侧和右侧均匀计算。否则, 最后一个额外的填充将从底部和右侧计算。如果设置了此模式, 则 $stride$ 必须为 1。
 - $"valid"$: 采用丢弃的方式。输出的可能最大高度和宽度将在没有填充的情况下返回。多余的像素将被丢弃。

如果 $padding$ 是一个整数, 则 top 、 $bottom$ 、 $left$ 和 $right$ 的 $padding$ 是相同的, 等于 $padding$ 。如果 $padding$ 是一个包含 3 个整数的元组/列表, 则 $head$ 、 $tail$ 、 top 、 $bottom$ 、 $left$ 和 $right$ 的填充分别等于 $pad[0]$ 、 $pad[0]$ 、 $pad[1]$ 、 $pad[1]$ 、 $pad[2]$ 和 $pad[2]$ 。默认值: 0。
- **dilation** (Union[int, tuple[int], list[int]], 可选) - 控制内核点之间的空间。默认值: 1。
- **groups** (int, 可选) - 将 $input$ 拆分的组数。默认值: 1。

返回:

Tensor, dtype 与 $input$ 相同, shape 为 $(N, C_{out}, D_{out}, H_{out}, W_{out})$ 或 $(C_{out}, D_{out}, H_{out}, W_{out})$ 。

异常:

- **TypeError** - $stride$ 、 $padding$ 或 $dilation$ 既不是 int 也不是 tuple。
- **TypeError** - $groups$ 不是 int。
- **TypeError** - $bias$ 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import mint
>>> x = mindspore.Tensor(np.random.randn(12, 1, 60, 50, 8), mindspore.float16)
>>> w = mindspore.Tensor(np.random.randn(26, 1, 2, 4, 4), mindspore.float16)
>>> out = mint.nn.functional.conv3d(x, w)
>>> print(out.shape)
(12, 26, 59, 47, 5)
```

mindspore.mint.nn.functional.conv_transpose2d

`mindspore.mint.nn.functional.conv_transpose2d` (*input*, *weight*, *bias=None*, *stride=1*, *padding=0*, *output_padding=0*, *groups=1*, *dilation=1*)

将 2D 转置卷积运算应用于由多个输入平面组成的输入图像, 有时也称为反卷积 (尽管它不是实际的反卷积)。

更多参考详见 `mindspore.mint.nn.ConvTranspose2d`。

警告:

- 这是一个实验性 API, 后续可能修改或删除。
- 在输入非连续场景下, `output_padding` 必须小于 `stride`。
- 在 Atlas 训练系列产品上, float32 类型输入时, 仅支持 `groups` 为 1。

参数:

- **input** (Tensor) - 输入 Tensor, shape 为 (*minibatch*, *in_channels*, *iH*, *iW*) 或 (*in_channels*, *iH*, *iW*)。
- **weight** (Tensor) - 卷积核, shape 为 (*in_channels*, $\frac{\text{out\_channels}}{\text{groups}}$, *kH*, *kW*)。
- **bias** (Tensor, 可选) - 偏置, shape 为 (*out_channels*)。默认值: None。
- **stride** (Union[int, tuple(int), list[int]], 可选) - 卷积的步长。可以为 1 个整数或 1 个元组 (*sH*, *sW*)。默认值: 1。
- **padding** (Union[int, tuple(int), list[int]], 可选) - $\text{dilation} * (\text{kernel\_size} - 1) - \text{padding}$ 零填充将添加到输入中每个维度的两侧。可以为 1 个整数或 1 个元组 (*padH*, *padW*)。默认值: 0。
- **output_padding** (Union[int, tuple(int), list[int]], 可选) - 在输出形状中每个维度的一侧增加额外的尺寸。可以为 1 个整数或 1 个元组 (*out_padH*, *out_padW*)。 `output_padding` 的值必须小于 `stride` 或 `dilation`。默认值: 0。
- **groups** (int, 可选) - 将输入分成 `groups` 组。 `in_channels` 应能被 `groups` 整除。默认值: 1。
- **dilation** (Union[int, tuple(int), list[int]], 可选) - 内核元素之间的间距。可以为 1 个整数或 1 个元组 (*dH*, *dW*)。默认值: 1。

返回:

Tensor, shape 为 (*minibatch*, *out_channels*, *oH*, *oW*) 或 (*out_channels*, *oH*, *oW*)。其中:

$$oH = (iH - 1) \times sH - 2 \times padH + dH \times (kH - 1) + out\_padH + 1$$

$$oW = (iW - 1) \times sW - 2 \times padW + dW \times (kW - 1) + out\_padW + 1$$

异常:

- **TypeError** - *stride*, *padding*, *output_padding* 或 *dilation* 既不是 int 也不是 tuple 或 list。
- **TypeError** - *groups* 不是 int。
- **ValueError** - *bias* 的 shape 不是 (*out_channels*)。
- **ValueError** - *stride* 或 *dilation* 小于 1。
- **ValueError** - *padding* 或 *output_padding* 小于 0。
- **ValueError** - *stride*, *padding*, *output_padding* 或 *dilation* 是 tuple 且其长度不等于 2。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.ones([1, 4, 5, 5]), mindspore.float32)
>>> weight = Tensor(np.ones([4, 8, 3, 3]), mindspore.float32)
>>> output = mint.nn.functional.conv_transpose2d(x, weight)
>>> print(output.shape)
(1, 8, 7, 7)
```

mindspore.mint.nn.functional.fold

`mindspore.mint.nn.functional.fold(input, output_size, kernel_size, dilation=1, padding=0, stride=1)`

将提取出的滑动局部区域还原成更大的输出 Tensor。

考虑一个 batch 的输入 Tensor，其 shape 为 $(N, C \times \prod(\text{kernel\_size}), L)$ ，其中 N 是批次维度， $C \times \prod(\text{kernel\_size})$ 为每个滑块内值的总数量（一个滑块有 $\prod(\text{kernel\_size})$ 个空间位置，每个位置都包含一个 C 通道的向量），共有 L 个这样的滑块：

$$L = \prod_d \left\lfloor \frac{\text{output\_size}[d] + 2 \times \text{padding}[d] - \text{dilation}[d] \times (\text{kernel\_size}[d] - 1) - 1}{\text{stride}[d]} + 1 \right\rfloor,$$

其中， d 遍历所有的空间维度。

因此，在最后一个维度（列维度）上 *output* 包含特定块内的所有值。

dilation, *padding* 和 *stride* 决定了滑块如何被取出。

警告:

- 目前，只支持 3-D 或 4-D（包含批维度）的类图像输出 Tensor。

参数:

- **input** (Tensor) - 二维或三维 Tensor。
- **output_size** (Union[int, tuple[int], list[int]]) - 表示输出 Tensor 的空间维度（如 `output.shape[2:]`）。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 滑动窗口的大小。如果类型为 tuple 或者 list，则存在两个 int 元素分别对应 kernel 的高度与宽度；如果类型为 int，则 kernel 的高度与宽度均为该值。

- **dilation** (Union[int, tuple[int], list[int]], 可选) - 窗口的空洞卷积的扩充率。如果类型为 **tuple** 或者 **list**, 则存在两个 **int** 元素分别作用于滑窗的高度和宽度; 如果类型为 **int**, 则这个值作用于滑窗的高度和宽度。默认值: 1。
- **padding** (Union[int, tuple[int], list[int]], 可选) - 滑窗的隐式零填充量。如果类型为 **tuple** 或者 **list**, 则存在的两个 **int** 元素分别为滑窗的高度和宽度方向的填充量; 如果类型为 **int**, 则高度和宽度方向的填充量均为这个 **int** 值。默认值: 0。
- **stride** (Union[int, tuple[int], list[int]], 可选) - 空间维度上滑动的步长。如果类型为 **tuple** 或者 **list**, 则存在的两个 **int** 元素分别为滑窗的高度和宽度方向上的步长; 如果类型为 **int**, 则高度和宽度方向上的步长均为这个 **int** 值。默认值: 1。

返回:

Tensor, 数据类型与 *input* 相同。

Shape:

- **input** - $(N, C \times \prod(\text{kernel\_size}), L)$ 或 $(C \times \prod(\text{kernel\_size}), L)$
- **output** - $(N, C, \text{output\_size}[0], \text{output\_size}[1], \dots)$ 或 $(C, \text{output\_size}[0], \text{output\_size}[1], \dots)$

异常:

- **TypeError** - 如果 *output_size*、*kernel_size*、*stride*、*dilation*、*padding* 的数据类型不是 **int**、**tuple** 或者 **list**。
- **ValueError** - 如果 *output_size*、*kernel_size*、*dilation*、*stride* 包含元素的值小于等于 0 或者元素个数不合法。
- **ValueError** - 如果 *padding* 包含元素的值小于零或者元素个数不合法。
- **ValueError** - 如果 *input.shape[-2]* 无法被 *kernel_size* 的乘积整除。
- **ValueError** - 如果 *input.shape[-1]* 不等于计算出的滑块数量 *L*。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.random.rand(16, 64, 25).astype(np.float32))
>>> output = mint.nn.functional.fold(x, (8, 8), [2, 2], [2, 2], [2, 2], [2, 2])
>>> print(output.shape)
(16, 16, 8, 8)
```

mindspore.mint.nn.functional.unfold

`mindspore.mint.nn.functional.unfold(input, kernel_size, dilation=1, padding=0, stride=1)`

从一个 batch 的输入 Tensor 中提取滑动局部块。

考虑一个 batch 的输入 Tensor, 其 shape 为 $(N, C, *)$, 其中 *N* 是 batch 维度, *C* 是 channel 维度, * 表示任意的空间维度。此操作将输入空间维度中每个大小为 *kernel_size* 的滑块展平为 3-D 输出 Tensor 的一列 (如最后一维), 输出 Tensor 的 shape 为 $(N, C \times \prod(\text{kernel\_size}), L)$, 其中 $C \times \prod(\text{kernel\_size})$ 为每个滑块内值的总数量 (一个滑块有 $\prod(\text{kernel\_size})$ 个空间位置, 每个位置都包含一个 *C* 通道的向量), 共有 *L* 个这样的滑块:

$$L = \prod_d \left\lfloor \frac{\text{spatial\_size}[d] + 2 \times \text{padding}[d] - \text{dilation}[d] \times (\text{kernel\_size}[d] - 1) - 1}{\text{stride}[d]} + 1 \right\rfloor,$$

其中, *spatial_size* 由 *input* 的空间维度 (上面的 *) 决定, *d* 遍历所有的空间维度。

因此，在最后一个维度（列维度）上 *output* 包含特定块内的所有值。

dilation, *padding* 和 *stride* 决定了滑块如何被取出。

警告：

- 目前，只支持 4-D（包含批维度）的类图像 Tensor。
- 对于 Ascend，仅 Atlas A2 以上平台支持。

参数：

- **input** (Tensor) - 四维 Tensor。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 滑窗大小。应该是两个 int，分别为滑窗的高度和宽度；如果 *kernel_size* 是 int，则高度和宽度均为这个 int 值。是一个必要参数。
- **dilation** (Union[int, tuple[int], list[int]])，可选 - 窗口的空洞卷积的扩充率，应该是两个 int，分别作用于滑窗的高度和宽度；如果 *dilation* 是 int，则这个值作用于滑窗的高度和宽度。默认值：1。
- **padding** (Union[int, tuple[int], list[int]])，可选 - 滑窗的隐式零填充量，应该是两个 int，分别作用于滑窗的高度和宽度；如果是 *padding* 是 int，则这个值作用于滑窗的高度和宽度。默认值：0。
- **stride** (Union[int, tuple[int], list[int]])，可选 - 空间维度上滑动的步长，应该是两个 int，分别为滑窗的高和宽方向上的步长；如果 *stride* 是 int，则高和宽方向上的步长均为这个 int 值。默认值：1。

返回：

Tensor，数据类型与 *input* 相同。

Shape:

- **input** - ($N, C, *$)
- **output** - ($N, C \times \prod(\text{kernel\_size}), L$)

异常：

- **TypeError** - 如果 *kernel_size*、*stride*、*dilation*、*padding* 的数据类型不是 int、tuple 或者 list。
- **ValueError** - 如果 *kernel_size*、*dilation*、*stride* 包含元素的值小于等于 0 或者元素个数不合法。
- **ValueError** - 如果 *padding* 包含元素的值小于零。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.random.rand(4, 4, 32, 32), mindspore.float32)
>>> output = mint.nn.functional.unfold(x, kernel_size=3, dilation=1, stride=1)
>>> print(output.shape)
(4, 36, 900)
```

4.5.2 池化函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.adaptive_avg_pool1d</code>	对一个多平面输入信号执行一维自适应平均池化。	Ascend
<code>mindspore.mint.nn.functional.adaptive_avg_pool2d</code>	对一个多平面输入信号执行二维自适应平均池化。	Ascend
<code>mindspore.mint.nn.functional.adaptive_avg_pool3d</code>	对一个多平面输入信号执行三维自适应平均池化。	Ascend
<code>mindspore.mint.nn.functional.adaptive_max_pool1d</code>	对一个多平面输入信号执行一维自适应最大池化。	Ascend
<code>mindspore.mint.nn.functional.avg_pool1d</code>	在输入 Tensor 上应用 1D 平均池化, 输入 Tensor 可以看作是由一系列 1D 平面组成的。	Ascend
<code>mindspore.mint.nn.functional.avg_pool2d</code>	在输入 Tensor 上应用 2D 平均池化, 输入 Tensor 可以看作是由一系列 2D 平面组成的。	Ascend
<code>mindspore.mint.nn.functional.avg_pool3d</code>	在输入 Tensor 上应用 3d 平均池化, 输入 Tensor 可以看作是由一系列 3d 平面组成的。	Ascend
<code>mindspore.mint.nn.functional.max_pool2d</code>	二维最大值池化。	Ascend
<code>mindspore.mint.nn.functional.max_unpool2d</code>	<code>max_pool2d</code> 的逆过程。	Ascend

mindspore.mint.nn.functional.adaptive_avg_pool1d

`mindspore.mint.nn.functional.adaptive_avg_pool1d(input, output_size)`
 对一个多平面输入信号执行一维自适应平均池化。也就是说，对于输入任何尺寸，指定输出的尺寸都为 L。但是输入和输出特征的数目不会变化。

警告：这是一个实验性 API，后续可能修改或删除。

- 参数：
- input** (Tensor) - `adaptive_avg_pool1d` 的输入，为二维或三维的 Tensor，数据类型为 float16 或者 float32。
 - output_size** (int) - 输出特征图的 size。`output_size` 是一个整数。

返回：
 Tensor，数据类型与 `input` 相同。
 输出的 shape 为 `input_shape[:len(input_shape) - 1] + [output_size]`。

- 异常：
- ValueError** - 如果 `output_size` 不是整数。
 - TypeError** - 如果 `input` 不是 Tensor。
 - TypeError** - 如果 `input` 的数据类型不是 float16、float32 或者 float64。

支持平台：
 Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([[2, 3], [3, 4]], dtype=mindspore.float16)
>>> output = mint.nn.functional.adaptive_avg_pool1d(input, 3)
>>> print(output)
[[2.  2.5 3. ]
 [3.  3.5 4. ]]
```

mindspore.mint.nn.functional.adaptive_avg_pool2d

`mindspore.mint.nn.functional.adaptive_avg_pool2d(input, output_size)`

对一个多平面输入信号执行二维自适应平均池化。也就是说，对于输入任何尺寸，指定输出的尺寸都为 $H * W$ 。但是输入和输出特征的数目不会变化。

输入和输出数据格式可以是“NCHW”和“CHW”。 N 是批处理大小， C 是通道数， H 是特征高度， W 是特征宽度。

对于二维的自适应平均池化操作，有如下公式：

$$\begin{aligned} h_{start} &= \text{floor}(i * H_{in}/H_{out}) \\ h_{end} &= \text{ceil}((i + 1) * H_{in}/H_{out}) \\ w_{start} &= \text{floor}(j * W_{in}/W_{out}) \\ w_{end} &= \text{ceil}((j + 1) * W_{in}/W_{out}) \\ \text{Output}(i, j) &= \frac{\sum \text{Input}[h_{start}:h_{end}, w_{start}:w_{end}]}{(h_{end} - h_{start}) * (w_{end} - w_{start})} \end{aligned}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - `adaptive_avg_pool2d` 的输入，为三维或四位的 Tensor，数据类型为 float16 或者 float32。
- **output_size** (Union[int, tuple]) - 输出特征图的 size。`output_size` 可以为二元 tuple 表示 (H, W) 。或者是单个 int 表示 (H, H) 。 H 和 W 可以是 int 或 None，如果是 None，则意味着输出的 size 与输入相同。

返回：

Tensor，数据类型与 `input` 相同。

输出的 shape 为 `input_shape[:len(input_shape) - len(out_shape)] + out_shape`。

$$\text{out\_shape} = \begin{cases} \text{input\_shape}[-2] + \text{output\_size}[1], & \text{if } \text{output\_size} \text{ is } (\text{None}, w); \\ \text{output\_size}[0] + \text{input\_shape}[-1], & \text{if } \text{output\_size} \text{ is } (h, \text{None}); \\ \text{input\_shape}[-2:], & \text{if } \text{output\_size} \text{ is } (\text{None}, \text{None}); \\ (h, h), & \text{if } \text{output\_size} \text{ is } h; \\ (h, w), & \text{if } \text{output\_size} \text{ is } (h, w) \end{cases}$$

异常：

- **ValueError** - 如果 `output_size` 是 tuple，并且 `output_size` 的长度不是 2。
- **ValueError** - 如果 `input` 的维度小于或等于 `output_size` 的维度。
- **TypeError** - 如果 `input` 不是 Tensor。
- **TypeError** - 如果 `input` 的数据类型不是 float16、float32 或者 float64。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> # case 1: output_size=(3, 2)
>>> input = Tensor(np.array([[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                           [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                           [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]]),
...mindspore.float32)
>>> output = mint.nn.functional.adaptive_avg_pool2d(input, (3, 2))
>>> print(output)
[[[1.5 2.5]
  [4.5 5.5]
  [7.5 8.5]]
 [[1.5 2.5]
  [4.5 5.5]
  [7.5 8.5]]
 [[1.5 2.5]
  [4.5 5.5]
  [7.5 8.5]]]
```

`mindspore.mint.nn.functional.adaptive_avg_pool3d`

`mindspore.mint.nn.functional.adaptive_avg_pool3d(input, output_size)`

对一个多平面输入信号执行三维自适应平均池化。对于任何输入尺寸，指定输出的尺寸都为 (D, H, W) ，但是输入和输出特征的数目不会变化。

假设输入 *input* 最后三维大小分别为 (inD, inH, inW) ，则输出的最后三维大小分别为 $(outD, outH, outW)$ ，运算如下：

$$\begin{aligned} \forall \quad & od \in [0, outD - 1], oh \in [0, outH - 1], ow \in [0, outW - 1] \\ & output[od, oh, ow] = \\ & \quad mean(x[istartD : iendD + 1, istartH : iendH + 1, istartW : iendW + 1]) \\ & \text{where,} \\ & \quad istartD = \left\lceil \frac{od * inD}{outD} \right\rceil \\ & \quad iendD = \left\lfloor \frac{(od+1) * inD}{outD} \right\rfloor \\ & \quad istartH = \left\lceil \frac{oh * inH}{outH} \right\rceil \\ & \quad iendH = \left\lfloor \frac{(oh+1) * inH}{outH} \right\rfloor \\ & \quad istartW = \left\lceil \frac{ow * inW}{outW} \right\rceil \\ & \quad iendW = \left\lfloor \frac{(ow+1) * inW}{outW} \right\rfloor \end{aligned}$$

警告： 对于 Ascend，仅支持 Atlas A2 训练系列产品。这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - `adaptive_avg_pool3d` 的输入，是 4D 或者 5D 的 Tensor。
- **output_size** (Union[int, tuple]) - 指定输出特征图的尺寸，可以是 tuple (D, H, W) ，也可以是一个 int 值 D 来表示输出尺寸为 (D, D, D) 。 D ， H 和 W 可以是 int 值或者 None，其中 None 表示输出大小与对应的输入的大小相同。

返回：

Tensor，与输入 *input* 的数据类型相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *input* 维度不是 4D 或者 5D。
- **ValueError** - *output_size* 的值不是正数。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> # case 1: output_size=(3, 3, 4)
>>> output_size=(3, 3, 4)
>>> input_val = np.random.randn(4, 3, 5, 6, 7)
>>> input = Tensor(input_val, mindspore.float32)
>>> output = mint.nn.functional.adaptive_avg_pool3d(input, output_size)
>>> print(output.shape)
(4, 3, 3, 3, 4)
>>> # case 2: output_size=4
>>> output_size=5
>>> input_val = np.random.randn(2, 3, 8, 6, 12)
>>> input = Tensor(input_val, mindspore.float32)
>>> output = mint.nn.functional.adaptive_avg_pool3d(input, output_size)
>>> print(output.shape)
(2, 3, 5, 5, 5)
>>> # case 3: output_size=(None, 4, 5)
>>> output_size=(None, 4, 5)
>>> input_val = np.random.randn(4, 1, 9, 10, 8)
>>> input = Tensor(input_val, mindspore.float32)
>>> output = mint.nn.functional.adaptive_avg_pool3d(input, output_size)
>>> print(output.shape)
(4, 1, 9, 4, 5)
```

mindspore.mint.nn.functional.adaptive_max_pool1d

`mindspore.mint.nn.functional.adaptive_max_pool1d(input, output_size, return_indices=False)`

对一个多平面输入信号执行一维自适应最大池化。也就是说，对于输入任何尺寸，指定输出的尺寸都为 L。但是输入和输出特征的数目不会变化。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - `adaptive_max_pool1d` 的输入，为二维或三维的 Tensor，数据类型为 float16、float32 或者 float64。
- **output_size** (int) - 输出特征图的 size。*output_size* 是一个整数。

- **return_indices** (bool, 可选) - 如果为 True，输出最大值的索引，默认值为 False。

返回：

Union(Tensor, tuple(Tensor, Tensor))。

- 如果 **return_indices** 为 False，返回 Tensor，其 shape 为 (N, C_{in}, L_{out}) ，数据类型与 **input** 相同。
- 如果 **return_indices** 为 True，则是一个包含了两个 Tensor 的 Tuple，表示计算结果以及生成 max 值的位置。

异常：

- **TypeError** - **input** 不是 Tensor。
- **TypeError** - **input** 中的数据不是 float16、float32、float64。
- **TypeError** - **output_size** 不是 int 或者 tuple。
- **TypeError** - **return_indices** 不是 bool。
- **ValueError** - **output_size** 是 tuple，但大小不是 1。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([[2, 3], [3, 4]], dtype=mindspore.float16)
>>> output = mint.nn.functional.adaptive_max_pool1d(input, 3)
>>> print(output)
[[2.  3.  3. ]
 [3.  4.  4. ]]
```

mindspore.mint.nn.functional.avg_pool1d

`mindspore.mint.nn.functional.avg_pool1d(input, kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True)`

在输入 Tensor 上应用 1D 平均池化，输入 Tensor 可以看作是由一系列 1D 平面组成的。

一般地，输入的 shape 为 (N_{in}, C_{in}, L_{in}) ，输出 (L_{in}) 维度的区域平均值。给定 **kernel_size** 为 $ks = l_{ker}$ 和 **stride** 为 $s = s_0$ ，运算如下：

$$\text{output}(N_i, C_j, l) = \frac{1}{l_{ker}} \sum_{n=0}^{l_{ker}-1} \text{input}(N_i, C_j, s_0 \times l + n)$$

警告：

- 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 shape 为 (N, C_{in}, L_{in}) 的 Tensor。
- **kernel_size** (Union(int, tuple[int])) - 指定池化核尺寸大小。
- **stride** (Union(int, tuple[int]), 可选) - 池化操作的移动步长。可以是一个整数或者包含一个整数的 tuple。默认值：None，表示与 **kernel_size** 值相同。

- **padding** (Union(int, tuple[int]), 可选) - 池化填充长度。可以是一个整数或者包含一个整数的 tuple。默认值: 0。
- **ceil_mode** (bool, 可选) - 如果为 True, 用 ceil 代替 floor 来计算输出的 shape。默认值: False。
- **count_include_pad** (bool, 可选) - 如果为 True, 平均计算将包括零填充。默认值: True。

返回:

Tensor, shape 为 (N, C_{in}, L_{out}) 。

异常:

- **TypeError** - *input* 不是一个 Tensor。
- **TypeError** - *kernel_size* 或 *stride* 不是 int。
- **TypeError** - *ceil_mode* 或 *count_include_pad* 不是 bool。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *kernel_size* 或 *stride* 或 *padding* 不是 int 或者 tuple 的长度大于 1。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input_x = Tensor(np.random.randint(0, 10, [1, 3, 6]), mindspore.float32)
>>> output = mint.nn.functional.avg_pool1d(input_x, kernel_size=6, stride=1)
>>> print(output.shape)
(1, 3, 1)
```

mindspore.mint.nn.functional.avg_pool2d

`mindspore.mint.nn.functional.avg_pool2d(input, kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)`

在输入 Tensor 上应用 2D 平均池化, 输入 Tensor 可以看作是由一系列 2D 平面组成的。

一般地, 输入的 shape 为 (N, C, H_{in}, W_{in}) , 输出 (H_{in}, W_{in}) 维度的区域平均值。给定 *kernel_size* 为 (kH, kW) 和 *stride*, 运算如下:

$$\text{output}(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

说明: 在 Atlas 平台上, 计算输入时, 精度会从 float32 降到 float16。

参数:

- **input** (Tensor) - shape 为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) 的 Tensor。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 指定池化核尺寸大小, 可以是单个整数或一个元组 (kH, kW) 。
- **stride** (Union[int, tuple[int], list[int]], 可选) - 池化操作的移动步长, 可以是单个整数或一个元组 (sH, sW) 。默认值: None, 此时其值等于 *kernel_size*。

- **padding** (Union[int, tuple[int], list[int]], 可选) - 池化填充长度, 可以是单个整数或一个元组 ($padH, padW$)。默认值: 0。
- **ceil_mode** (bool, 可选) - 如果为 True, 用 ceil 代替 floor 来计算输出的 shape。默认值: False。
- **count_include_pad** (bool, 可选) - 如果为 True, 平均计算将包括零填充。默认值: True。
- **divisor_override** (int, 可选) - 如果指定了该值, 它将在平均计算中用作除数, 否则, 将使用池化区域的大小。默认值: None。

返回:

Tensor, 其 shape 为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) 。

$$H_{out} = \frac{H_{in} + 2 \times padding[0] - kernel\_size[0]}{stride[0]} + 1$$

$$W_{out} = \frac{W_{in} + 2 \times padding[1] - kernel\_size[1]}{stride[1]} + 1$$

异常:

- **TypeError** - input 不是一个 Tensor。
- **TypeError** - kernel_size 或 stride 既不是 int 也不是 tuple。
- **TypeError** - ceil_mode 或 count_include_pad 不是 bool。
- **TypeError** - divisor_override 不是 int 或 None。
- **ValueError** - input 的维度不等于 3 或 4。
- **ValueError** - kernel_size 或 stride 小于 1。
- **ValueError** - padding 的值小于 0。
- **ValueError** - kernel_size、padding 或 stride 是 tuple 且其长度不等于 1 或 2。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.arange(1 * 3 * 3 * 4).reshape(1, 3, 3, 4), mindspore.float32)
>>> output = mint.nn.functional.avg_pool2d(x, kernel_size=2, stride=1)
>>> print(output)
[[[ [ 2.5   3.5   4.5]
      [ 6.5   7.5   8.5]]
  [[14.5  15.5  16.5]
    [18.5  19.5  20.5]]
  [[26.5  27.5  28.5]
    [30.5  31.5  32.5]]]]]
```

mindspore.mint.nn.functional.avg_pool3d

`mindspore.mint.nn.functional.avg_pool3d(input, kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)`

在输入 Tensor 上应用 3d 平均池化, 输入 Tensor 可以看作是由一系列 3d 平面组成的。

一般地, 输入的 shape 为 $(N, C, D_{in}, H_{in}, W_{in})$, 输出 $(D_{out}, H_{out}, W_{out})$ 维度的区域平均值。给定 `kernel_size` 为 (kD, kH, kW) 和 `stride`, 运算如下:

$$\text{output}(N_i, C_j, d, h, w) = \frac{1}{kD * kH * kW} \sum_{l=0}^{kD-1} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times d + l, \text{stride}[1] \times h + m, \text{stride}[2] \times w + n)$$

警告: 这是一个实验性 API, 后续可能修改或删除。

说明: 该接口暂不支持 Atlas A2 训练系列产品。

参数:

- **input** (Tensor) - shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 或 $(C, D_{in}, H_{in}, W_{in})$ 的 Tensor。
- **kernel_size** (Union[int, tuple[int], list[int]]) - 指定池化核尺寸大小, 可以是单个整数或一个元组 (kD, kH, kW) 。
- **stride** (Union[int, tuple[int], list[int]], 可选) - 池化操作的移动步长, 可以是单个整数或一个元组 (sD, sH, sW) 。默认值: None, 此时其值等于 `kernel_size`。
- **padding** (Union[int, tuple[int], list[int]], 可选) - 池化填充长度, 可以是单个整数或一个元组 $(padD, padH, padW)$ 。默认值: 0。
- **ceil_mode** (bool, 可选) - 如果为 True, 用 ceil 代替 floor 来计算输出的 shape。默认值: False。
- **count_include_pad** (bool, 可选) - 如果为 True, 平均计算将包括零填充。默认值: True。
- **divisor_override** (int, 可选) - 如果指定了该值, 它将在平均计算中用作除数, 否则, 将使用池化区域的大小。默认值: None。

返回:

Tensor, 其 shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 或 $(C, D_{out}, H_{out}, W_{out})$ 。

$$\begin{aligned} D_{out} &= \frac{D_{in} + 2 \times padding[0] - kernel\_size[0]}{stride[0]} + 1 \\ H_{out} &= \frac{H_{in} + 2 \times padding[1] - kernel\_size[1]}{stride[1]} + 1 \\ W_{out} &= \frac{W_{in} + 2 \times padding[2] - kernel\_size[2]}{stride[2]} + 1 \end{aligned}$$

异常:

- **TypeError** - `input` 不是一个 Tensor。
- **TypeError** - `kernel_size` 或 `stride` 既不是 int 也不是 tuple。
- **TypeError** - `ceil_mode` 或 `count_include_pad` 不是 bool。
- **TypeError** - `divisor_override` 不是 int 或 None。

- **ValueError** - *input* 的维度不等于 4 或 5。
- **ValueError** - *kernel_size* 或 *stride* 小于 1。
- **ValueError** - *padding* 的值小于 0。
- **ValueError** - *kernel_size*、*padding* 或 *stride* 是 tuple 且其长度不等于 1 或 3。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input_x = Tensor(np.arange(1 * 2 * 2 * 2 * 3).reshape((1, 2, 2, 2, 3)), mindspore.
    ↳float16)
>>> output = mint.nn.functional.avg_pool3d(input_x, kernel_size=2, stride=1)
>>> print(output)
[[[[[ 5.  6.]]]
  [[17. 18.]]]]
```

mindspore.mint.nn.functional.max_pool2d

`mindspore.mint.nn.functional.max_pool2d(input, kernel_size, stride=None, padding=0, dilation=1, ceil_mode=False, return_indices=False)`

二维最大值池化。

输入是 shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$ 的 Tensor，输出 (H_{in}, W_{in}) 维度中的最大值。给定 *kernel_size* $ks = (h_{ker}, w_{ker})$ ，和 *stride* $s = (s_0, s_1)$ ，运算如下：

$$\text{output}(N_i, C_j, h, w) = \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} \text{input}(N_i, C_j, s_0 \times h + m, s_1 \times w + n)$$

警告：只支持 Atlas A2 训练系列产品。

参数：

- **input** (Tensor) - shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$ 的 Tensor。在 Ascend 上，数据类型仅支持 float32。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。可以是一个整数，同时表示池化核的高度和宽度。或者包含两个整数的 tuple，分别表示池化核的高度和宽度。
- **stride** (Union[int, tuple[int], None], 可选) - 池化操作的移动步长。可以是一个整数，同时表示在高度和宽度方向的移动步长。或者包含两个整数的 tuple，分别表示在高度和宽度方向的移动步长。默认值：None，表示移动步长为 *kernel_size*。
- **padding** (Union[int, tuple[int]], 可选) - 池化填充长度。可以是一个整数，同时表示在高度和宽度方向的填充长度。或者包含两个整数的 tuple，分别表示在高度和宽度方向的填充长度。默认值：0。
- **dilation** (Union[int, tuple[int]], 可选) - 控制池化核内元素的间距。默认值：1。
- **ceil_mode** (bool, 可选) - 是否是用 ceil 代替 floor 来计算输出的 shape。默认值：False。
- **return_indices** (bool, 可选) - 是否输出最大值的索引。默认值：False。

返回：

当 `return_indices` 是 `False` 时, 输出单个 `output` Tensor, 否则输出一个包含 `output` Tensor 和 `argmax` Tensor 的元组。

- **output** (Tensor) - 输出的池化后的最大值, `shape` 为 $(N_{out}, C_{out}, H_{out}, W_{out})$ 。其数据类型与 `input` 相同。

$$H_{out} = \left\lfloor \frac{H_{in} + 2 * padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 * padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

- **argmax** (Tensor) - 输出的最大值对应的索引。在 Ascend 上, 其数据类型为 `int32`。仅当 `return_indices` 为 `True` 时, 才返回该值。

异常:

- **TypeError** - `input` 不是 Tensor。
- **ValueError** - `input` 的维度不是 4D。
- **TypeError** - `kernel_size`、`stride`、`padding`、`dilation` 不是 `int` 或者 `tuple`。
- **ValueError** - `kernel_size`、`stride` 或者 `dilation` 的元素值小于 1。
- **ValueError** - `dilation` 不是全为 1。
- **ValueError** - `padding` 的元素值小于 0。
- **ValueError** - `padding` 的元素值大于 `kernel_size` 的一半。
- **TypeError** - `ceil_mode` 不是 `bool` 类型。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.arange(20 * 16 * 50 * 32).reshape((20, 16, 50, 32)), mindspore.float32)
>>> output_tensor, argmax = mint.nn.functional.max_pool2d(input, kernel_size=(3, 2),
↳ stride=(2, 1),
...
                                  ceil_mode=False, return_
↳ indices=True)
>>> print(output_tensor.shape)
(20, 16, 24, 31)
>>> print(argmax.shape)
(20, 16, 24, 31)
```

`mindspore.mint.nn.functional.max_unpool2d`

`mindspore.mint.nn.functional.max_unpool2d` (`input`, `indices`, `kernel_size`, `stride=None`, `padding=0`, `output_size=None`)

`max_pool2d` 的逆过程。

`max_unpool2d` 在计算过程中, 保留最大值位置的元素, 并将非最大值位置元素设置为 0。支持的输入数据格式为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) , 输出数据的格式为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) , 计算公式如下:

$$H_{out} = (H_{in} - 1) \times stride[0] - 2 \times padding[0] + kernel\_size[0]$$

$$W_{out} = (W_{in} - 1) \times stride[1] - 2 \times padding[1] + kernel\_size[1]$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 待求逆的 Tensor。shape 为 (N, C, H_{in}, W_{in}) 或 (C, H_{in}, W_{in}) 。
- **indices** (Tensor) - 最大值的索引。shape 必须与输入 *input* 相同。取值范围需满足 $[0, H_{in} \times W_{in} - 1]$ 。数据类型必须是 int32 或 int64。
- **kernel_size** (Union[int, tuple[int]]) - 池化核尺寸大小。int 类型表示池化核的长宽相同。tuple 类型中的两个值分别代表池化核的长和宽。
- **stride** (Union[int, tuple[int]], 可选) - 池化操作的移动步长，int 类型表示长宽方向的移动步长相同。tuple 中的两个值分别代表长宽方向移动的步长。默认值：None，表示移动步长为 *kernel_size*。
- **padding** (Union[int, tuple[int]], 可选) - 填充值。默认值：0。若为 int 类型，则长宽方向的填充大小相同，均为 *padding*。若为 tuple 类型，则 tuple 中的两个值分别代表长宽方向填充的大小。
- **output_size** (tuple[int], 可选) - 输出 shape。默认值：None。如果 *output_size* 为 ()，那么输出 shape 根据 *kernel_size*、*stride* 和 *padding* 计算得出。如果 *output_size* 不为 ()，那么 *output_size* 必须满足格式 (N, C, H, W) ， (C, H, W) 或 (H, W) ，取值范围需满足： $[(N, C, H_{out} - stride[0], W_{out} - stride[1]), (N, C, H_{out} + stride[0], W_{out} + stride[1])]$ 。

返回：

shape 为 (N, C, H_{out}, W_{out}) 或 (C, H_{out}, W_{out}) 的 Tensor，数据类型与输入 *input* 相同。

异常：

- **TypeError** - *input* 或 *indices* 的数据类型不支持。
- **TypeError** - *kernel_size*、*stride* 或 *padding* 既不是整数也不是 tuple。
- **ValueError** - *stride*、*padding* 或 *kernel_size* 的值不是非负的。
- **ValueError** - *input* 和 *indices* 的 shape 不一致。
- **ValueError** - *input* 的长度不为 3 或 4。
- **ValueError** - *output_size* 的类型不是 tuple。
- **ValueError** - *output_size* 的取值与根据 *kernel_size*、*stride*、*padding* 计算得到的结果差距太大。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> indices = Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> output = mint.nn.functional.max_unpool2d(input, indices, 1, stride=1, padding=0)
>>> print(output.asnumpy())
[[[0. 1.]
  [8. 9.]]]]
```

4.5.3 非线性激活函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.batch_norm</code>	对输入数据进行批量归一化和更新参数。	Ascend
<code>mindspore.mint.nn.functional.elu</code>	指数线性单元激活函数。	Ascend
<code>mindspore.mint.nn.functional.elu_</code>	指数线性单元激活函数。	Ascend
<code>mindspore.mint.nn.functional.gelu</code>	高斯误差线性单元激活函数。	Ascend
<code>mindspore.mint.nn.functional.glu</code>	计算输入 Tensor 的门线性单元激活函数 (Gated Linear Unit activation function) 值。	Ascend
<code>mindspore.mint.nn.functional.group_norm</code>	在 mini-batch 输入上进行组归一化。	Ascend
<code>mindspore.mint.nn.functional.hardshrink</code>	Hard Shrink 激活函数。	Ascend
<code>mindspore.mint.nn.functional.hardsigmoid</code>	Hard Sigmoid 激活函数。	Ascend
<code>mindspore.mint.nn.functional.hardswish</code>	Hard Swish 激活函数。	Ascend
<code>mindspore.mint.nn.functional.layer_norm</code>	在 mini-batch 输入上应用层归一化 (Layer Normalization)。	Ascend
<code>mindspore.mint.nn.functional.leaky_relu</code>	leaky_relu 激活函数。	Ascend
<code>mindspore.mint.nn.functional.log_softmax</code>	在指定轴上对输入 Tensor 应用 LogSoftmax 函数。	Ascend
<code>mindspore.mint.nn.functional.logsigmoid</code>	按元素计算 LogSigmoid 激活函数。	Ascend
<code>mindspore.mint.nn.functional.mish</code>	逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。	Ascend
<code>mindspore.mint.nn.functional.prelu</code>	带参数的线性修正单元激活函数 (Parametric Rectified Linear Unit activation function)。	Ascend
<code>mindspore.mint.nn.functional.relu</code>	对输入 Tensor 逐元素计算线性修正单元激活函数 (Rectified Linear Unit) 值。	Ascend
<code>mindspore.mint.nn.functional.relu6</code>	逐元素计算输入 Tensor 的 ReLU (修正线性单元), 其上限为 6。	Ascend
<code>mindspore.mint.nn.functional.relu_</code>	对输入 Tensor 逐元素原地计算线性修正单元激活函数 (Rectified Linear Unit) 值。	Ascend
<code>mindspore.mint.nn.functional.selu</code>	激活函数 selu (Scaled exponential Linear Unit)。	Ascend
<code>mindspore.mint.nn.functional.sigmoid</code>	逐元素计算 Sigmoid 激活函数。	Ascend
<code>mindspore.mint.nn.functional.silu</code>	逐元素计算激活函数 SiLU (Sigmoid Linear Unit)。	Ascend
<code>mindspore.mint.nn.functional.softmax</code>	在指定轴上对输入 Tensor 执行 Softmax 激活函数做归一化操作。	Ascend
<code>mindspore.mint.nn.functional.softplus</code>	将 softplus 函数作用于 <i>input</i> 的每个元素上。	Ascend
<code>mindspore.mint.nn.functional.softshrink</code>	Soft Shrink 激活函数。	Ascend
<code>mindspore.mint.nn.functional.tanh</code>	逐元素计算输入元素的双曲正切。	Ascend

mindspore.mint.nn.functional.batch_norm

`mindspore.mint.nn.functional.batch_norm` (*input*, *running_mean*, *running_var*, *weight=None*, *bias=None*, *training=False*, *momentum=0.1*, *eps=1e-5*)

对输入数据进行批量归一化和更新参数。

批量归一化广泛应用于卷积神经网络中。此运算对输入应用归一化，避免内部协变量偏移，详见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。使用 mini-batch 数据和学习参数进行训练，学习的参数见如下公式中，

$$y = \frac{x - \text{mean}}{\sqrt{\text{variance} + \epsilon}} * \gamma + \beta$$

其中， γ 为 *weight*， β 为 *bias*， ϵ 为 *eps*，*mean* 为 *x* 的均值，*variance* 为 *x* 的方差。

参数：

- **input** (Tensor) - 数据输入，shape 为 $(N, C, *)$ 的 Tensor，其中 $*$ 表示任意的附加维度。对于 Atlas 训练系列产品，当前仅支持 2-4 维。支持数据类型为 bfloat16，float16 或 float32。
- **running_mean** (Tensor) - shape 为 $(C,)$ ，数据类型为 bfloat16，float16 或 float32。
- **running_var** (Tensor) - shape 为 $(C,)$ ，数据类型为 bfloat16，float16 或 float32。
- **weight** (Tensor, 可选) - shape 为 $(C,)$ ，数据类型为 bfloat16，float16 或 float32，默认值: None。为 None 时，初始化为 1。
- **bias** (Tensor, 可选) - shape 为 $(C,)$ ，数据类型为 bfloat16，float16 或 float32，默认值: None。为 None 时，初始化为 0。
- **training** (bool, 可选) - 如果 *training* 为 *True*，*running_mean* 和 *running_var* 会在训练过程中进行计算。如果 *training* 为 *False*，它们会在推理阶段从 *checkpoint* 中加载。默认值: *False*。
- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。(例如 $\text{new\_running\_mean} = (1 - \text{momentum}) * \text{running\_mean} + \text{momentum} * \text{current\_mean}$)。默认值: 0.1。
- **eps** (float, 可选) - 添加到分母上的值，以确保数值稳定性。默认值: $1e-5$ 。

返回：

Tensor，数据类型与 shape 大小与 *input* 相同，其中，shape 大小为 $(N, C, *)$ 。

异常：

- **TypeError** - *training* 不是 bool。
- **TypeError** - *eps* 或 *momentum* 的数据类型不是 float。
- **TypeError** - *input*、*weight*、*bias*、*running_mean* 或 *running_var* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input_x = Tensor([[1.0, 2.0], [3.0, 4.0]], mindspore.float32)
>>> running_mean = Tensor([0.5, 1.5], mindspore.float32)
>>> running_var = Tensor([0.1, 0.2], mindspore.float32)
>>> weight = Tensor([2.0, 2.0], mindspore.float32)
>>> bias = Tensor([-1.0, -1.0], mindspore.float32)
>>> output = mint.nn.functional.batch_norm(input_x, running_mean, running_var, weight, bias)
```

(下页继续)

(续上页)

```
>>> print(output)
[[ 2.1621194  1.2360122]
 [14.810596 10.180061 ]]
```

`mindspore.mint.nn.functional.elu`

`mindspore.mint.nn.functional.elu(input, alpha=1.0, inplace=False)`

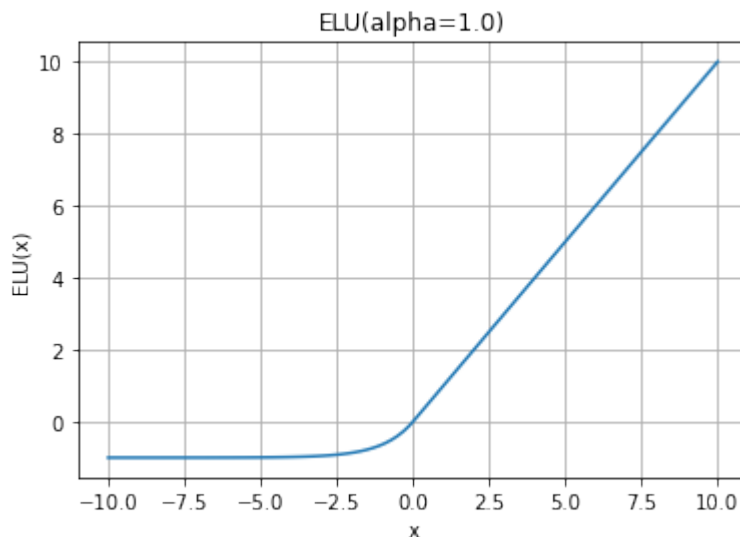
指数线性单元激活函数。

对输入的每个元素计算 ELU。该激活函数定义如下：

$$ELU_i = \begin{cases} x_i, & \text{if } x_i \geq 0; \\ \alpha * (\exp(x_i) - 1), & \text{otherwise.} \end{cases}$$

其中, x_i 表示输入的元素, α 表示 *alpha* 参数, *alpha* 决定 ELU 的平滑度。

ELU 函数图：



警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - ELU 的输入，为任意维度的 Tensor。
- **alpha** (float, 可选) - ELU 的 alpha 值，数据类型为 float。默认值：1.0。
- **inplace** (bool, 可选) - 是否使用原地更新模式，数据类型为 bool。默认值：False。

返回：

Tensor，输出的 shape 和数据类型与 *input* 相同。

异常：

- **RuntimeError** - 如果 *input* 的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - 如果 *alpha* 不是 float。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float32)
>>> output = mint.nn.functional.elu(input)
>>> print(output)
[-0.63212055 -0.86466473  0.  2.  1.]
```

`mindspore.mint.nn.functional.elu_`

`mindspore.mint.nn.functional.elu_(input, alpha=1.0)`

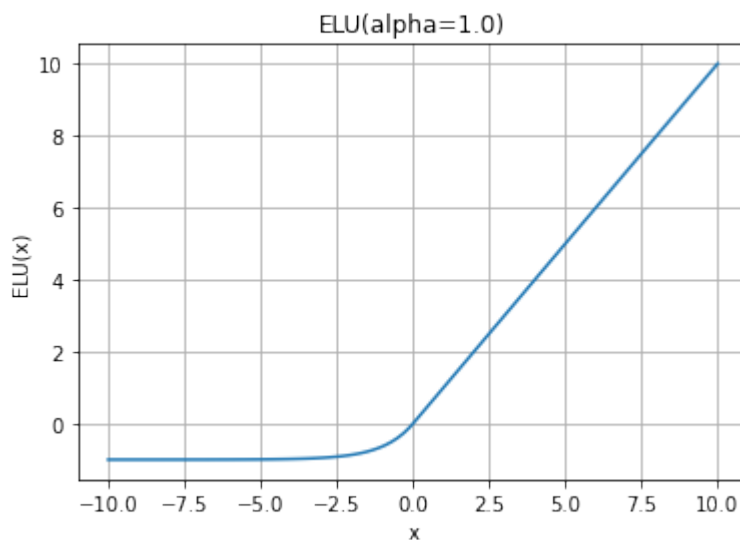
指数线性单元激活函数。

对输入的每个元素原地计算 ELU。该激活函数定义如下:

$$ELU_i = \begin{cases} x_i, & \text{if } x_i \geq 0; \\ \alpha * (\exp(x_i) - 1), & \text{otherwise.} \end{cases}$$

其中, x_i 表示输入的元素, α 表示 *alpha* 参数, *alpha* 决定 ELU 的平滑度。

ELU 函数图:



警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - ELU 的输入, 为任意维度的 Tensor。
- **alpha** (float, 可选) - ELU 的 alpha 值, 数据类型为 float 且必须大于 0。默认值: 1.0。

返回:

Tensor, 输出的 shape 和数据类型与 *input* 相同。

异常:

- **RuntimeError** - 如果 *input* 的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - 如果 *alpha* 的数据类型不是 float。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float32)
>>> mint.nn.functional.elu_(input)
>>> print(input)
[-0.63212055 -0.86466473  0.  2.  1.]
```

mindspore.mint.nn.functional.gelu

`mindspore.mint.nn.functional.gelu(input, *, approximate='none')`

高斯误差线性单元激活函数。

GeLU 的描述可以在 [Gaussian Error Linear Units \(GELUs\)](#) 这篇文章中找到。详情可查询 [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)。

当 *approximate* 为 *none*，GELU 的定义如下:

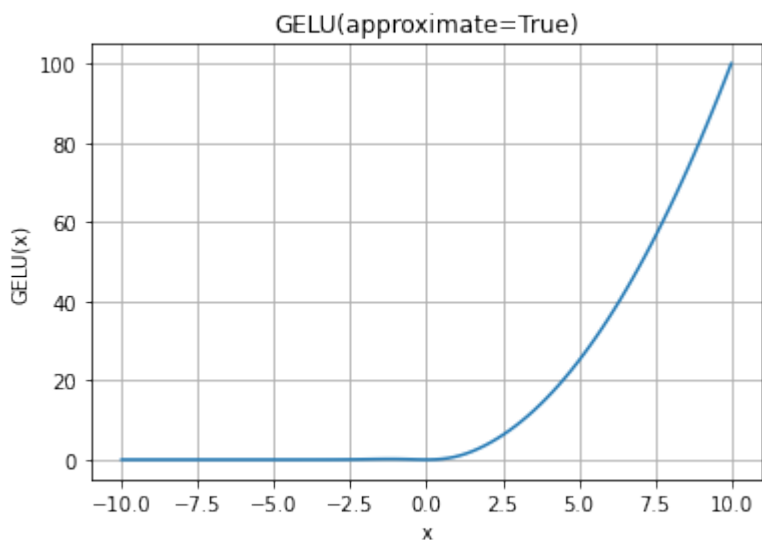
$$GELU(x_i) = x_i * P(X < x_i),$$

其中 P 是标准高斯分布的累积分布函数, x_i 是输入的元素。

当 *approximate* 为 *tanh*，GELU 的定义如下:

$$GELU(x_i) = 0.5 * x_i * (1 + \tanh(\sqrt{2/\pi} * (x_i + 0.044715 * x_i^3)))$$

GELU 函数图:



说明：在 Ascend 平台上，当 *input* 为 -inf 时，其梯度为 0，当 *input* 为 inf 时，其梯度为 *dout*。

参数：

- **input** (Tensor) - 用于计算 GELU 的 Tensor。数据类型是 float16、float32 或 float64。

关键字参数：

- **approximate** (str, 可选) - gelu 近似算法。有两种：'none' 和 'tanh'。默认值：'none'。

返回：

Tensor，具有与 *input* 相同的数据类型和 shape。

异常：

- **TypeError** - 如果 *input* 的数据类型不是 Tensor。
- **TypeError** - *input* 的数据类型既不是 bfloat16、float16、float32 或者 float64。
- **ValueError** - 如果 *approximate* 的值既不是 *none* 也不是 *tanh*。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> result = mint.nn.functional.gelu(input)
>>> print(result)
[[-1.58655241e-01  3.99987316e+00 -0.00000000e+00]
 [ 1.95449972e+00 -1.41860323e-06  9.00000000e+00]]
>>> result = mint.nn.functional.gelu(input, approximate="tanh")
>>> print(result)
[[-1.58808023e-01  3.99992990e+00 -3.10779147e-21]
 [ 1.95459759e+00 -2.29180174e-07  9.00000000e+00]]
```

mindspore.mint.nn.functional.glu

`mindspore.mint.nn.functional.glu(input, dim=-1)`

计算输入 Tensor 的门线性单元激活函数（Gated Linear Unit activation function）值。

$$GLU(a, b) = a \otimes \sigma(b)$$

其中，*a* 表示输入 Tensor *input* 拆分后的前半元素，*b* 表示 *input* 拆分后的另一半元素。其中 σ 是 sigmoid 函数， $\otimes$ 是 Hadamard 乘积。请参考 [Language Modeling with Gated Convluational Networks](#)。

参数：

- **input** (Tensor) - 待计算的 Tensor，数据类型为浮点型，shape 为 $(*_1, N, *_2)$ ，其中 $*$ 为任意额外维度，且要求 *N* 为偶数。*N* 为 *input* 在被 *dim* 选中的维度上的大小。
- **dim** (int, 可选) - 指定分割 *input* 的轴，取值范围 $[-r, r)$ ，其中 *r* 为 *input* 的维度数。默认值：-1，输入 *input* 的最后一维。

返回：

Tensor, 数据类型与输入 *input* 相同, shape 为 $(*_1, M, *_2)$, 其中 $M = N/2$ 。

异常:

- **TypeError** - *input* 不是 Tensor 或 *dim* 不是整型。
- **IndexError** - *dim* 的值超出了 $[-r, r)$ 的范围, 其中 r 为 *input* 的维度数。
- **RuntimeError** - *input* 数据类型不被支持。
- **RuntimeError** - *input* 在被 *dim* 选中的维度上的长度不为偶数。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, mint
>>> input = Tensor([[0.1, 0.2, 0.3, 0.4], [0.5, 0.6, 0.7, 0.8]])
>>> output = mint.nn.functional.glu(input)
>>> print(output)
[[0.05744425 0.11973753]
 [0.33409387 0.41398472]]
```

mindspore.mint.nn.functional.group_norm

`mindspore.mint.nn.functional.group_norm(input, num_groups, weight=None, bias=None, eps=1e-5)`

在 mini-batch 输入上进行组归一化。

Group Normalization 被广泛用于递归神经网络中。适用于单个训练用例的 mini-batch 输入归一化, 详见论文 [Group Normalization](#)。

Group Normalization 把通道划分为组, 然后计算每一组之内的均值和方差, 以进行归一化。其中 γ 是通过训练学习出的 scale 值, β 是通过训练学习出的 shift 值。

公式如下,

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中, γ 为 *weight*, β 为 *bias*, ϵ 为 *eps*。

参数:

- **input** (Tensor) - shape 为 $(N, C, *)$ 的特征输入, 其中 $*$ 表示任意的附加维度。
- **num_groups** (int) - 沿通道维度待划分的组数。
- **weight** (Tensor, 可选) - shape 为 $(C,)$, 默认值为: None, 具有与 *input* 相同的数据类型。
- **bias** (Tensor, 可选) - shape 为 $(C,)$, 默认值为: None, 具有与 *input* 相同的数据类型。
- **eps** (float, 可选) - 添加到分母中的值, 以确保数值稳定。默认 $1e-5$ 。

返回:

Tensor, 标准化和缩放的偏移 Tensor, 具有与 *input* 相同的 shape 和数据类型。

异常:

- **TypeError** - *num_groups* 不是 int。
- **TypeError** - *eps* 不是 float。

- **ValueError** - *num_groups* 小于 1。
- **ValueError** - *C* (*input* 的第二维) 未被 *num_groups* 整除。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import mint
>>> x = ms.Tensor(np.ones([1, 2, 4, 4], np.float32))
>>> output = mint.nn.functional.group_norm(x, 2)
>>> print(output)
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]]]
```

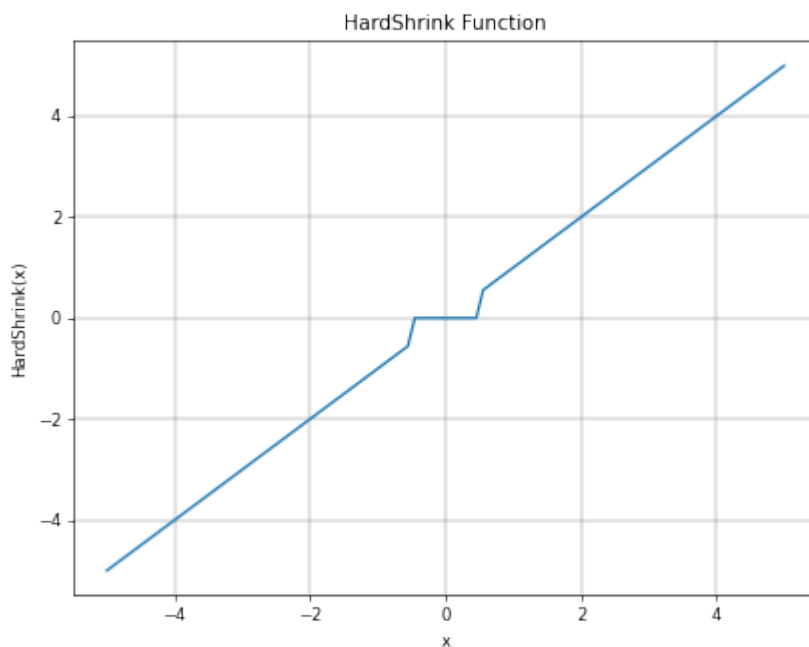
mindspore.mint.nn.functional.hardshrink

`mindspore.mint.nn.functional.hardshrink` (*input*, *lambd*=0.5)

Hard Shrink 激活函数。按输入元素计算输出。公式定义如下：

$$\text{HardShrink}(x) = \begin{cases} x, & \text{if } x > \lambda \\ x, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

HardShrink 激活函数图：



参数:

- **input** (Tensor) - Hard Shrink 的输入。支持数据类型：
 - Ascend: float16、float32、bfloat16。
- **lambd** (number, 可选) - Hard Shrink 公式定义的阈值 λ 。默认值: 0.5。

返回:

Tensor, shape 和数据类型与输入 *input* 相同。

异常:

- **TypeError** - *lambd* 不是 float, bool 或 int。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的 dtype 不是 float16、float32 或 bfloat16。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[0.5, 1, 2.0], [0.0533, 0.0776, -2.1233]]), mindspore.float32)
>>> output = mint.nn.functional.hardshrink(input)
>>> print(output)
[[ 0.      1.      2.      ]
 [ 0.      0.     -2.1233]]
```


mindspore.mint.nn.functional.hardsigmoid

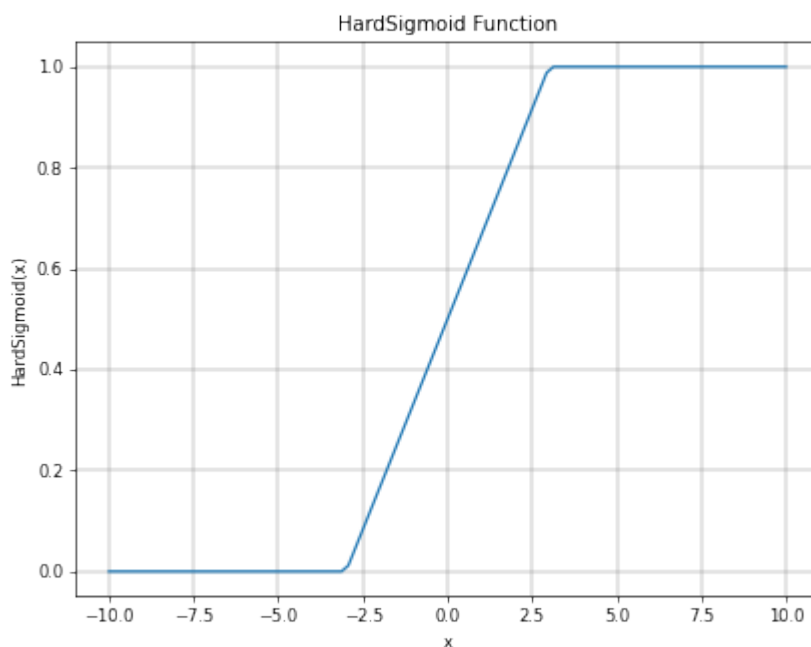
`mindspore.mint.nn.functional.hardsigmoid(input)`

Hard Sigmoid 激活函数。按元素计算输出。

Hard Sigmoid 定义为：

$$\text{HardSigmoid}(input) = \begin{cases} 0, & \text{if } input \leq -3, \\ 1, & \text{if } input \geq +3, \\ input/6 + 1/2, & \text{otherwise} \end{cases}$$

HardSigmoid 函数图：



参数：

- **input** (Tensor) - 输入 Tensor。

返回：

Tensor, shape 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
```

(下页继续)

(续上页)

```
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> output = mint.nn.functional.hardsigmoid(input)
>>> print(output)
[0.3333 0.1666 0.5      0.8335 0.6665]
```

`mindspore.mint.nn.functional.hardswish`

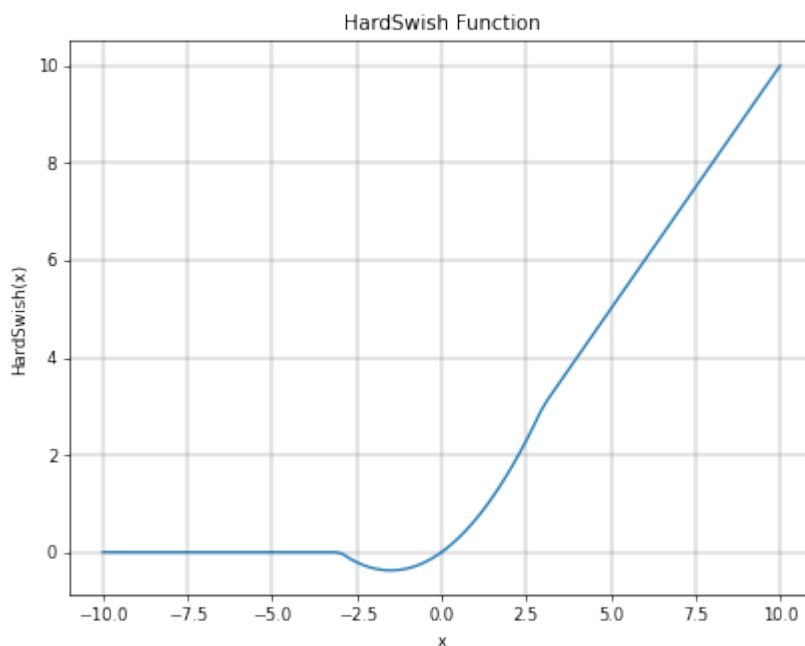
`mindspore.mint.nn.functional.hardswish(input)`

Hard Swish 激活函数。输入是一个 Tensor，具有任何有效的 shape。

Hard Swish 定义如下：

$$\text{HardSwish}(\text{input}) = \begin{cases} 0, & \text{if } \text{input} \leq -3, \\ \text{input}, & \text{if } \text{input} \geq +3, \\ \text{input} * (\text{input} + 3)/6, & \text{otherwise} \end{cases}$$

HardSwish 函数图：



参数：

- **input** (Tensor) - Hard Swish 的输入。

返回：

Tensor, shape 和数据类型与输入相同。

异常：

- **TypeError** - *input* 不是一个 Tensor。
- **TypeError** - *input* 不是 int 或者 float 类型。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([-1, -2, 0, 2, 1]), mindspore.float16)
>>> output = mint.nn.functional.hardswish(input)
>>> print(output)
[-0.3333 -0.3333  0  1.667  0.6665]
```

mindspore.mint.nn.functional.layer_norm

`mindspore.mint.nn.functional.layer_norm(input, normalized_shape, weight=None, bias=None, eps=1e-5)`

在 mini-batch 输入上应用层归一化 (Layer Normalization)。

层归一化在递归神经网络中被广泛的应用。适用单个训练用例的 mini-batch 输入上应用归一化, 详见论文 [Layer Normalization](#)。

与批归一化 (Batch Normalization) 不同, 层归一化在训练和测试时执行完全相同的计算。应用于所有通道和像素, 即使 `batch_size=1` 也适用。公式如下:

$$y = \frac{x - E[x]}{\sqrt{\text{Var}[x] + \epsilon}} * \gamma + \beta$$

其中 γ 是通过训练学习出的 `weight` 值, β 是通过训练学习出的 `bias` 值。

参数:

- **input** (Tensor) - `input` 的 shape 为 $(N, *)$, 其中 $*$ 表示任意的附加维度。
- **normalized_shape** (Union(int, tuple[int], list[int])) - 表示需要进行归一化的 shape, `normalized_shape` 等于 `input_shape[begin_norm_axis:]`, `begin_norm_axis` 代表归一化要开始的轴。
- **weight** (Tensor, 可选) - 可学习的权重值, shape 为 `normalized_shape`, 默认值: `None`。为 `None` 时, 初始化为 1。
- **bias** (Tensor, 可选) - 可学习的偏移值, shape 为 `normalized_shape`, 默认值: `None`。为 `None` 时, 初始化为 0。
- **eps** (float, 可选) - 添加到分母中的值 (ϵ), 以确保数值稳定。默认 `1e-5`。

返回:

Tensor, 归一化后的 Tensor, shape 和数据类型与 `input` 相同。

异常:

- **TypeError** - `input` 不是 Tensor。
- **TypeError** - `normalized_shape` 既不是 list 也不是 tuple。
- **TypeError** - `eps` 不是 float。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input_x = Tensor(np.array([[1, 2, 3], [1, 2, 3]]), mindspore.float32)
>>> normalized_shape = (3,)
>>> gamma = Tensor(np.ones(normalized_shape), mindspore.float32)
>>> beta = Tensor(np.zeros(normalized_shape), mindspore.float32)
>>> eps = 1e-7
>>> output = mint.nn.functional.layer_norm(input_x, normalized_shape, gamma, beta, eps)
>>> print(output)
[[-1.2247448 0. 1.2247448]
 [-1.2247448 0. 1.2247448]]
```

mindspore.mint.nn.functional.leaky_relu

`mindspore.mint.nn.functional.leaky_relu(input, negative_slope=0.01)`

`leaky_relu` 激活函数。`input` 中小于 0 的元素乘以 `negative_slope`。

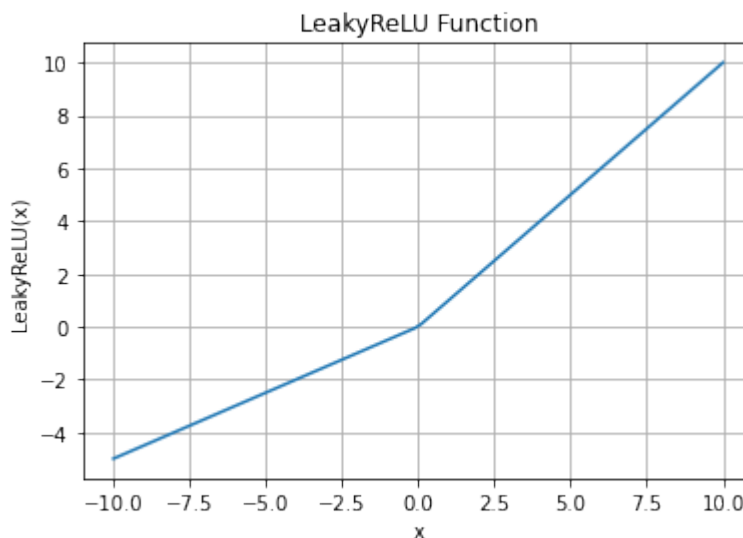
该激活函数定义如下：

$$\text{leaky\_relu}(\text{input}) = \begin{cases} \text{input}, & \text{if } \text{input} \geq 0; \\ \text{negative\_slope} * \text{input}, & \text{otherwise.} \end{cases}$$

其中，`negative_slope` 表示 `negative_slope` 参数。

更多细节详见 [Rectifier Nonlinearities Improve Neural Network Acoustic Models](#)。

LeakyReLU 函数图：



参数：

- **input** (Tensor) - 计算 `leaky_relu` 的任意维度的 Tensor。
- **negative_slope** (Union[int, float], 可选) - `input` 的元素小于 0 时激活函数的斜率，默认值：0.01。

返回：

Tensor，数据类型和 `shape` 与 `input` 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *negative_slope* 不是浮点数或整数。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> print(mint.nn.functional.leaky_relu(input, negative_slope=0.2))
[[-0.2  4.  -1.6]
 [ 2.  -1.   9. ]]
```

mindspore.mint.nn.functional.log_softmax

`mindspore.mint.nn.functional.log_softmax(input, dim=None, *, dtype=None)`

在指定轴上对输入 Tensor 应用 LogSoftmax 函数。假设在指定轴上， x 对应每个元素 x_i ，则 LogSoftmax 函数如下所示：

$$\text{output}(x_i) = \log \left(\frac{\exp(x_i)}{\sum_{j=0}^{N-1} \exp(x_j)} \right),$$

其中， N 为 Tensor 长度。

参数：

- **input** (Tensor) - 输入 Tensor。
- **dim** (int, 可选) - 指定进行 Log softmax 运算的轴。默认值：None。

关键字参数：

- **dtype** (*mindspore.dtype*, 可选) - 输出数据类型。如果不为 None，则输入会转化为 *dtype*。这有利于防止数值溢出。如果为 None，则输出和输入的数据类型一致。默认值：None。

返回：

Tensor，和输入 Tensor 的 shape 相同。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **ValueError** - 如果 *dim* 超出范围 $[-\text{len}(\text{input.shape}), \text{len}(\text{input.shape})$ 。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> logits = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> output = mint.nn.functional.log_softmax(logits, dim=-1)
>>> print(output)
[-4.4519143 -3.4519143 -2.4519143 -1.4519144 -0.4519144]
```

mindspore.mint.nn.functional.logsigmoid

`mindspore.mint.nn.functional.logsigmoid(input)`

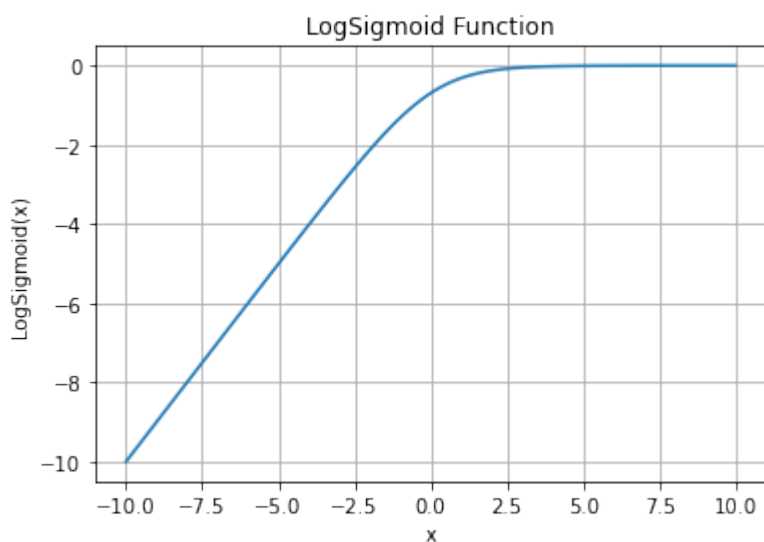
按元素计算 LogSigmoid 激活函数。输入是任意维度的 Tensor。

LogSigmoid 定义为：

$$\text{logsigmoid}(x_i) = \log\left(\frac{1}{1 + \exp(-x_i)}\right),$$

其中, x_i 是输入 Tensor 的一个元素。

LogSigmoid 函数图：



警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - logsigmoid 的输入，数据类型为 bfloat16、float16 或 float32。shape 为 (*)，其中 * 表示任意的附加维度。

返回：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型既不是 bfloat16、float16 也不是 float32。
- **TypeError** - *input* 不是一个 Tensor。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([1.0, 2.0, 3.0], mindspore.float32)
>>> output = mint.nn.functional.logsigmoid(input)
>>> print(output)
[-0.31326166 -0.12692806 -0.04858734]
```

`mindspore.mint.nn.functional.mish`

`mindspore.mint.nn.functional.mish(input)`

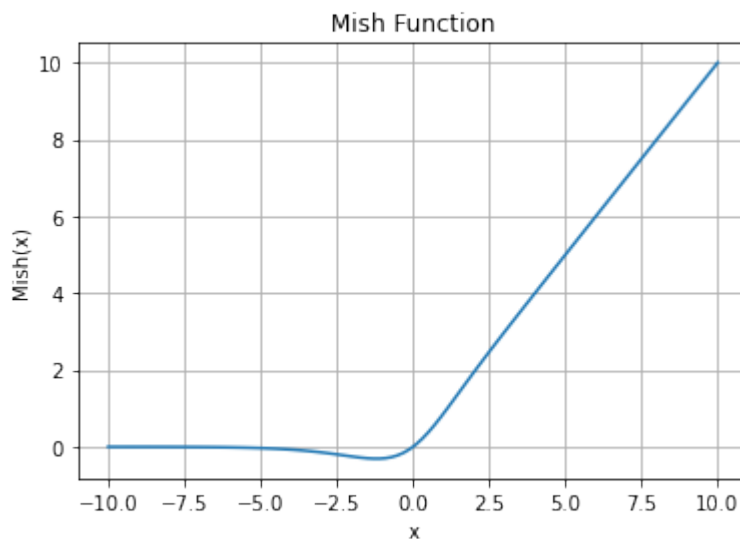
逐元素计算输入 Tensor 的 MISH (A Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。

公式如下:

$$\text{mish}(\text{input}) = \text{input} * \tanh(\text{softplus}(\text{input}))$$

更多详细信息请参见 [A Self Regularized Non-Monotonic Neural Activation Function](#)。

Mish 激活函数图:



参数:

- **input** (Tensor) - Mish 的输入。支持数据类型:
 - Ascend: float16、float32。

返回:

Tensor, 与 *input* 的 shape 和数据类型相同。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的数据类型不是 float16 或 float32。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> x = Tensor(np.array([[ -1.1, 4.0, -8.0], [2.0, -5.0, 9.0]]), mindspore.float32)
>>> output = mint.nn.functional.mish(x)
>>> print(output)
[[-3.0764845e-01  3.9974124e+00 -2.6832507e-03]
 [ 1.9439589e+00 -3.3576239e-02  8.9999990e+00]]
```

`mindspore.mint.nn.functional.prelu`

`mindspore.mint.nn.functional.prelu` (*input*, *weight*)

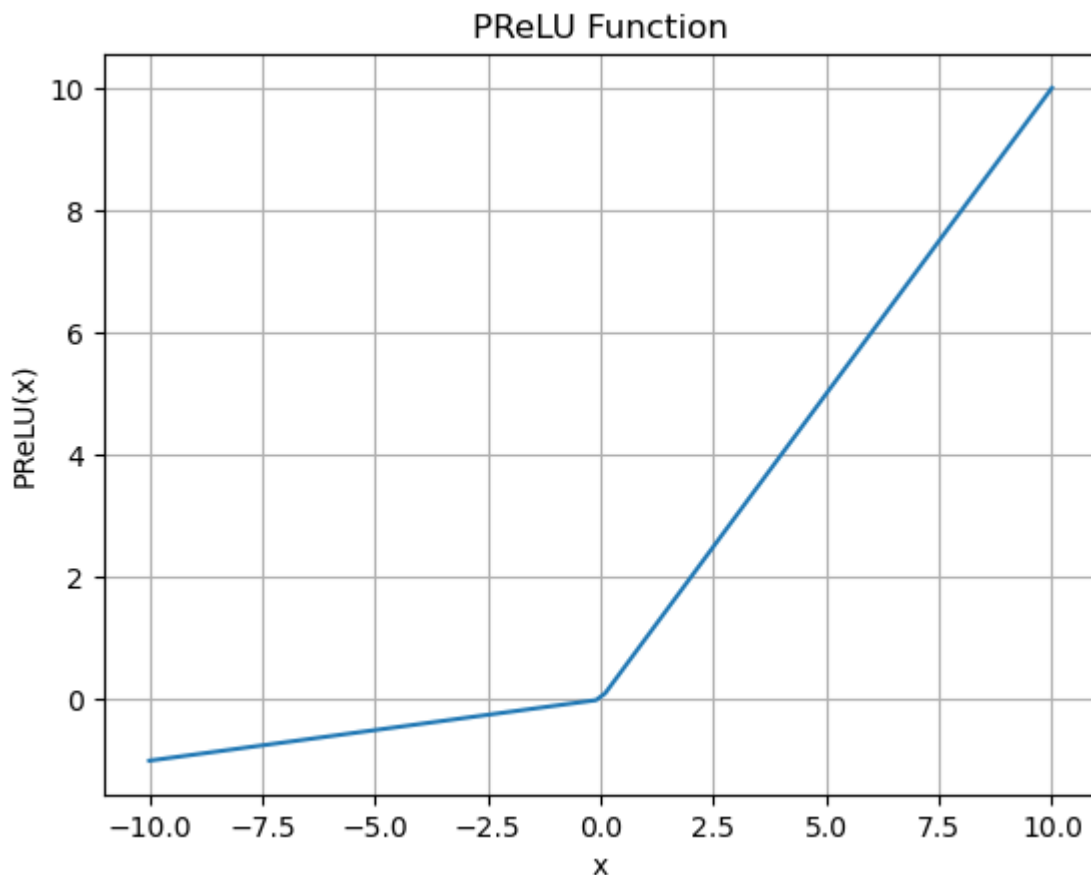
带参数的线性修正单元激活函数 (Parametric Rectified Linear Unit activation function)。

[Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification](#) 描述了 PReLU 激活函数。定义如下:

$$\text{prelu}(x_i) = \max(0, x_i) + \min(0, w * x_i),$$

其中 x_i 是输入的一个通道的一个元素, w 是通道权重。

PReLU 函数图:



说明：通道数是输入的第二个维度值。当输入的维度小于 2 时，则没有通道维度，此时通道数等于 1。

参数：

- **input** (Tensor) - 激活函数的输入 Tensor。
- **weight** (Tensor) - 权重 Tensor。其大小为 1 或者等于输入 Tensor *input* 的通道数。

返回：

Tensor，其 shape 和数据类型与 *input* 相同。有关详细信息，请参考 `mindspore.mint.nn.PReLU`。

异常：

- **TypeError** - *input* 或 *weight* 不是 Tensor。
- **ValueError** - *weight* 不是 0-D 或 1-D Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.arange(-6, 6).reshape((2, 3, 2)), mindspore.float32)
>>> weight = Tensor(np.array([0.1, 0.6, -0.3]), mindspore.float32)
>>> output = mint.nn.functional.prelu(x, weight)
>>> print(output)
[[[-0.60 -0.50]
  [-2.40 -1.80]
  [ 0.60  0.30]]
 [[ 0.00  1.00]
  [ 2.00  3.00]
  [ 4.0   5.00]]]
```

mindspore.mint.nn.functional.relu

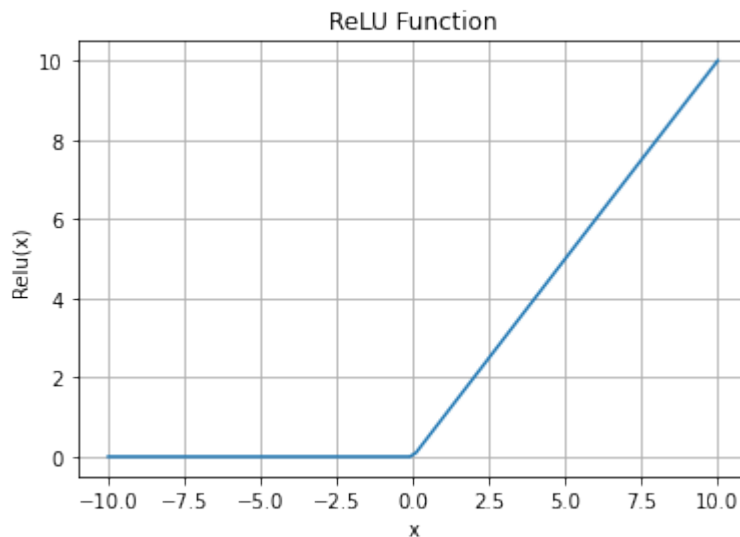
`mindspore.mint.nn.functional.relu(input, inplace=False)`

对输入 Tensor 逐元素计算线性修正单元激活函数（Rectified Linear Unit）值。

返回 $\max(input, 0)$ 的值。负值神经元将被设置为 0，正值神经元将保持不变。

$$ReLU(input) = (input)^+ = \max(0, input)$$

ReLU 激活函数图：



参数:

- **input** (Tensor) - 输入 Tensor。
- **inplace** (bool, 可选) - 是否采用原地更新模式, 默认值为 `False`。

返回:

Tensor, 其 `shape` 和数据类型与输入一致。

异常:

- **TypeError** - *input* 的数据类型不是数值型。
- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> output = mint.nn.functional.relu(input)
>>> print(output)
[[0.  4.  0.]
 [2.  0.  9.]]
```

mindspore.mint.nn.functional.relu6

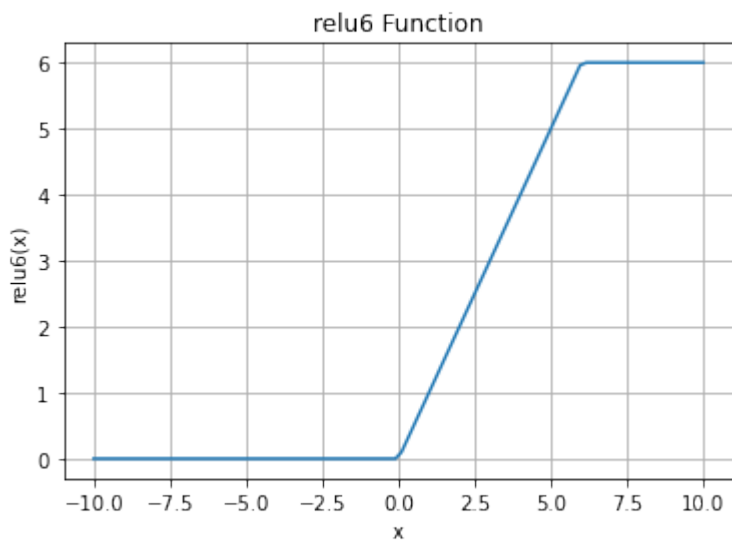
`mindspore.mint.nn.functional.relu6(input, inplace=False)`

逐元素计算输入 Tensor 的 ReLU (修正线性单元), 其上限为 6。

$$\text{ReLU6}(\text{input}) = \min(\max(0, \text{input}), 6)$$

返回 $\min(\max(0, \text{input}), 6)$ 元素的值。

ReLU6 函数图:



警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入 Tensor。数据类型为 int8、int16、int32、int64、uint8、float16、float32、bfloat16。
- **inplace** (bool, 可选) - 是否采用原地更新模式, 默认值为 False。

返回:

Tensor, 其 shape 和数据类型与 *input* 相同。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的数据类型不是 int8、int16、int32、int64、uint8、float16、float32、bfloat16。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> result = mint.nn.functional.relu6(x)
>>> print(result)
[[0. 4. 0.]
 [2. 0. 6.]]
```

`mindspore.mint.nn.functional.relu_`

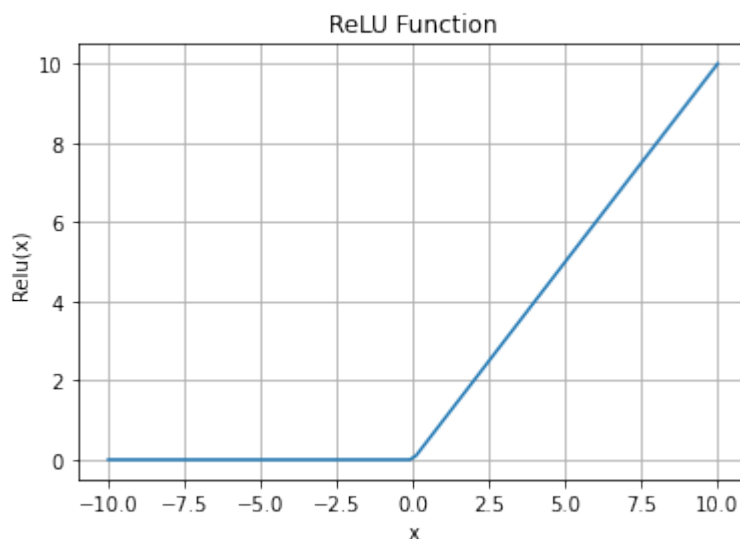
`mindspore.mint.nn.functional.relu_(input)`

对输入 **Tensor** 逐元素原地计算线性修正单元激活函数 (Rectified Linear Unit) 值。

返回 $\max(input, 0)$ 的值。负值神经元将被设置为 0，正值神经元将保持不变。

$$ReLU(input) = (input)^+ = \max(0, input)$$

ReLU 激活函数图：



警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入 Tensor。

返回：

Tensor，其 `shape` 和数据类型与输入一致。

异常：

- **TypeError** - *input* 的数据类型不是数值型。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> mint.nn.functional.relu_(input)
>>> print(input)
[[0. 4. 0.]
 [2. 0. 9.]]
```

mindspore.mint.nn.functional.selu

mindspore.mint.nn.functional.**selu**(input)

激活函数 selu (Scaled exponential Linear Unit)。

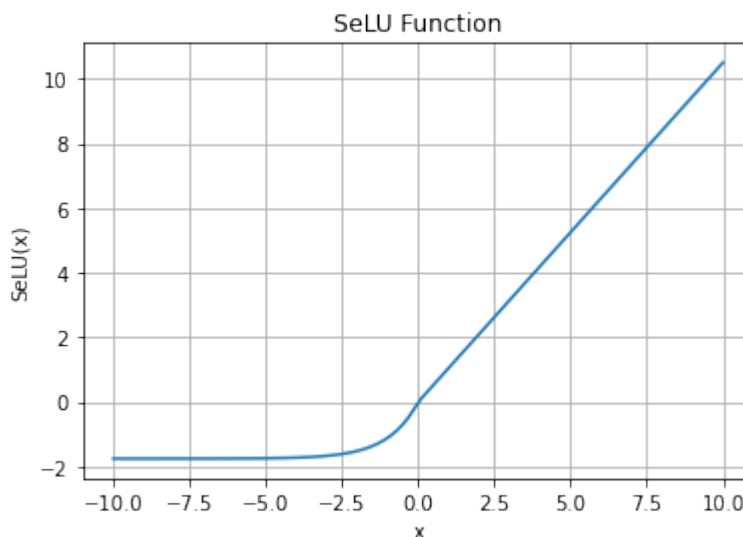
该激活函数定义为：

$$E_i = scale * \begin{cases} x_i, & \text{if } x_i \geq 0; \\ \alpha * (\exp(x_i) - 1), & \text{otherwise.} \end{cases}$$

其中, α 和 $scale$ 是预定义的常量 ($\alpha = 1.67326324$, $scale = 1.05070098$)。

更多详细信息, 请参见 [Self-Normalizing Neural Networks](#)。

SELU 函数图：



参数：

- **input** (Tensor) - 任意维度的 Tensor，数据类型为 float16、float32、bfloat16。

返回：

Tensor，数据类型和 shape 与 *input* 的相同。

异常：

- **TypeError** - *input* 的数据类型不是 float16、float32、bfloat16。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> output = mint.nn.functional.selu(input)
>>> print(output)
[[-1.1113307  4.202804 -1.7575096]
 [ 2.101402 -1.7462534  9.456309  ]]
```

`mindspore.mint.nn.functional.sigmoid`

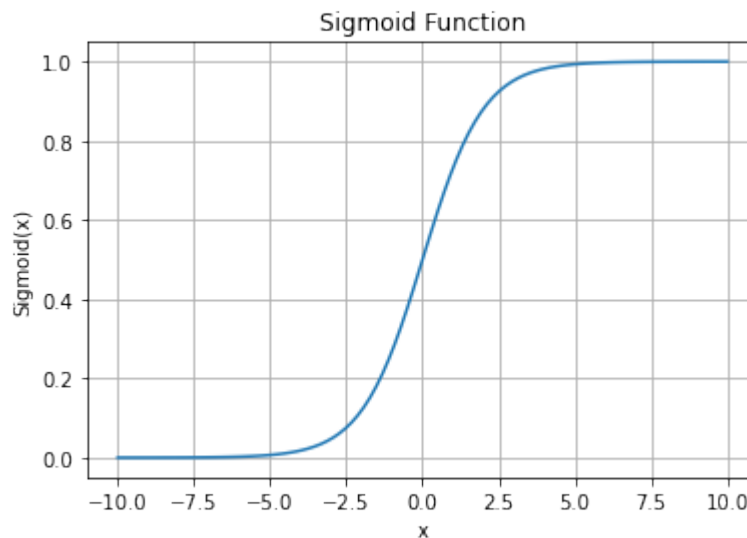
`mindspore.mint.nn.functional.sigmoid(input)`

逐元素计算 Sigmoid 激活函数。Sigmoid 函数定义为:

$$\text{sigmoid}(x_i) = \frac{1}{1 + \exp(-x_i)}$$

其中 x_i 是 x 的一个元素。

Sigmoid 函数图:



参数:

- **input** (Tensor) - *input* 即为上述公式中的 x 。任意维度的 Tensor, 数据类型为 float16、float32、float64、complex64 或 complex128。

返回:

Tensor, 数据类型和 shape 与 *input* 的相同。

异常:

- **TypeError** - *input* 的数据类型不是 float16、float32、float64、complex64 或 complex128。
- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = mint.nn.functional.sigmoid(input)
>>> print(output)
[0.7310586  0.880797  0.95257413 0.98201376 0.9933072 ]
```

`mindspore.mint.nn.functional.silu`

`mindspore.mint.nn.functional.silu(input)`

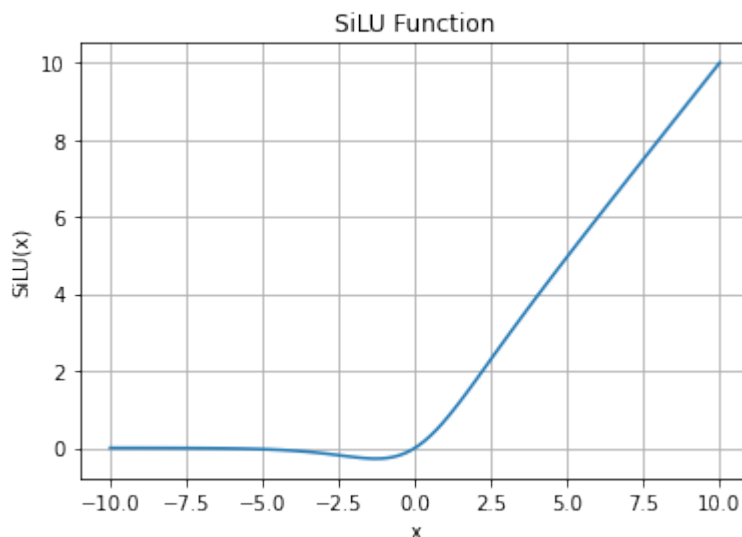
逐元素计算激活函数 SiLU (Sigmoid Linear Unit)。有时也被称作 Swish 函数。该激活函数定义为:

$$\text{SiLU}(x) = x * \sigma(x),$$

其中 x 是输入的元素, $\sigma(x)$ 是 Sigmoid 函数。

$$\sigma(x_i) = \frac{1}{1 + \exp(-x_i)},$$

SiLU 函数图:



参数:

- **input** (Tensor) - *input* 即为上述公式中的 x 。数据类型为 `float16` 或 `float32` 的输入。

返回:

Tensor, 数据类型和 `shape` 与 *input* 的相同。

异常:

- **TypeError** - *input* 的数据类型既不是 `float16` 也不是 `float32`。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> import numpy as np
>>> input = Tensor(np.array([-1, 2, -3, 2, -1]), mindspore.float16)
>>> output = mint.nn.functional.silu(input)
>>> print(output)
[-0.269  1.762 -0.1423  1.762 -0.269]
```

mindspore.mint.nn.functional.softmax

`mindspore.mint.nn.functional.softmax(input, dim=None, dtype=None)`

在指定轴上对输入 Tensor 执行 Softmax 激活函数做归一化操作。假设指定轴 *dim* 上有切片，那么每个元素 $input_i$ 所对应的 Softmax 函数如下所示：

$$output(input_i) = \frac{\exp(input_i)}{\sum_{j=0}^{N-1} \exp(input_j)},$$

其中 N 代表 Tensor 的长度。

参数：

- **input** (Tensor) - Tensor, shape 为 $(N,*)$ ，其中 $*$ 为任意额外维度。
- **dim** (int, 可选) - 指定 Softmax 操作的轴。默认值：None。
- **dtype** (`mindspore.dtype`, 可选) - 如果设置此参数，则会在执行之前将 *input* 转换为指定的类型，返回的 Tensor 类型也将为指定类型 *dtype*。默认值：None。

返回：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *dim* 不是 int。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = mint.nn.functional.softmax(input)
>>> print(output)
[0.01165623 0.03168492 0.08612854 0.23412167 0.6364086 ]
```


mindspore.mint.nn.functional.softplus

`mindspore.mint.nn.functional.softplus(input, beta=1, threshold=20)`

将 `softplus` 函数作用于 `input` 的每个元素上。

`softplus` 函数如下所示，其中 x 是 `input` 中的元素：

$$\text{output} = \frac{1}{\text{beta}} \log(1 + \exp(\text{beta} * x))$$

当 $\text{input} * \text{beta} > \text{threshold}$ 时，为保证数值稳定性，`softplus` 的实现被转换为线性函数。

参数：

- **input** (Tensor) - 任意维度的输入 Tensor。支持数据类型：
 - Ascend: float16、float32、bfloat16。
- **beta** (number.Number, 可选) - `softplus` 函数中的缩放参数。默认值：1。
- **threshold** (number.Number, 可选) - 为了数值稳定性，`softplus` 函数转换为线性函数的阈值参数。默认值：20。

返回：

Tensor，其数据类型和 `shape` 与 `input` 相同。

异常：

- **TypeError** - `input` 不是 Tensor。
- **TypeError** - `input` 的数据类型非 float16、float32、bfloat16。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0.1, 0.2, 30, 25]), mindspore.float32)
>>> output = mint.nn.functional.softplus(input)
>>> print(output)
[0.74439657 0.7981388 30. 25.]
```

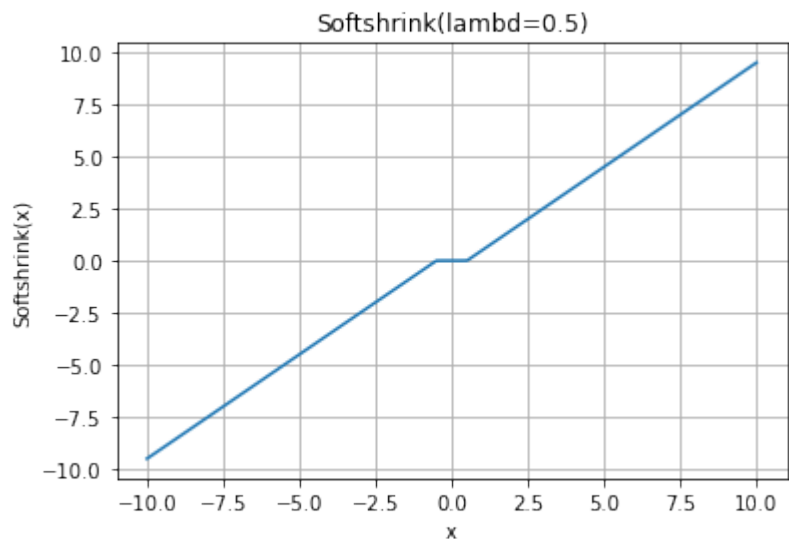
mindspore.mint.nn.functional.softshrink

`mindspore.mint.nn.functional.softshrink(input, lambd=0.5)`

Soft Shrink 激活函数。按输入元素计算输出。公式定义如下：

$$\text{SoftShrink}(x) = \begin{cases} x - \lambda, & \text{if } x > \lambda \\ x + \lambda, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}$$

SoftShrink 激活函数图：



参数：

- **input** (Tensor) - Soft Shrink 的输入。支持数据类型：
 - Ascend：float16、float32、bfloat16。
- **lambd** (number, 可选) - Soft Shrink 公式定义的阈值 λ 。*lambd* 应该大于等于 0。默认值：0.5。

返回：

Tensor, shape 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *lambd* 不是 float、bool 或 int。
- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *input* 的 dtype 不是 float16, float32 或 bfloat16。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore import mint
>>> import numpy as np
>>> x = Tensor(np.array([[ 0.5297,  0.7871,  1.1754], [ 0.7836,  0.6218, -1.1542]]),
↳mindspore.float32)
>>> output = mint.nn.functional.softshrink(x)
>>> print(output)
[[ 0.02979  0.287    0.676   ]
 [ 0.2837   0.1216  -0.6543  ]]
```

mindspore.mint.nn.functional.tanh

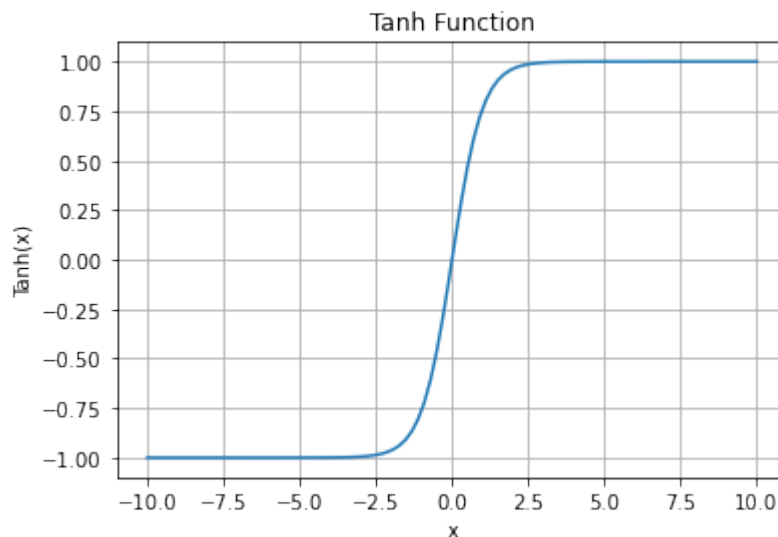
`mindspore.mint.nn.functional.tanh(input)`

逐元素计算输入元素的双曲正切。Tanh 函数定义为：

$$\tanh(x_i) = \frac{\exp(x_i) - \exp(-x_i)}{\exp(x_i) + \exp(-x_i)} = \frac{\exp(2x_i) - 1}{\exp(2x_i) + 1},$$

其中 x_i 是输入 Tensor 的元素。

Tanh 函数图：



参数：

- **input** (Tensor) - Tanh 的输入。

返回：

Tensor，数据类型和 shape 与 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = mint.nn.functional.tanh(input)
>>> print(output)
[0.7615941 0.9640276 0.9950547 0.9993293 0.9999092]
```

4.5.4 归一化函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.normalize</code>	将输入的张量按照指定维度进行归一化。	Ascend

mindspore.mint.nn.functional.normalize

`mindspore.mint.nn.functional.normalize(input, p=2.0, dim=1, eps=1e-12)`

将输入的张量按照指定维度进行归一化。

对于一个输入的张量，它的维度为 $(n_0, ..., n_{dim}, ..., n_k)$ ，对于第 n_{dim} 个向量 v ，它沿着维度 dim 按照如下公式进行转换

$$v = \frac{v}{\max(\|v\|_p, \epsilon)}$$

默认归一化计算方法为沿着第一个维度利用欧几里得范数进行归一化。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 输入的 Tensor。
- **p** (float) - 范数公式中的指数值。默认值为 2。
- **dim** (int) - 指定的维度。默认值为 1。
- **eps** (float) - 设置的最小值，以避免除法分母为 0。默认值为 1e-12。

返回：

- Tensor。shape 和数据类型与输入 input 相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> tensor = Tensor(np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]]), mindspore.float32)
>>> output = mint.nn.functional.normalize(tensor)
>>> print(output)
[[0.0000 0.4472 0.8944]
 [0.4243 0.5657 0.7071]
 [0.4915 0.5735 0.6554]]
```

4.5.5 线性函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.linear</code>	对输入 <i>input</i> 应用全连接操作。	Ascend

mindspore.mint.nn.functional.linear

`mindspore.mint.nn.functional.linear(input, weight, bias=None)`

对输入 *input* 应用全连接操作。全连接定义为：

$$output = input * weight^T + bias$$

警告：

- 这是一个实验性 API，后续可能修改或删除。
- 在 Ascend 硬件平台下，设置 PYNATIVE 或 KBK 模式时，如果 *bias* 不是 1D，*input* 不可以大于 6D。

参数：

- **input** (Tensor) - 输入 Tensor，shape 是 $(*, in\_channels)$ ，其中 * 表示任意的附加维度。
- **weight** (Tensor) - 输入 Tensor 的权重，shape 是 $(out\_channels, in\_channels)$ 或 $(in\_channels)$ 。
- **bias** (Tensor，可选) - 添加在输出结果的偏差，shape 是 $(out\_channels)$ 或 $()$ 。默认值：None，偏差为 0。

返回：

输出结果，shape 由 *input* 和 *weight* 的 shape 决定。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *weight* 不是 Tensor。
- **TypeError** - *bias* 不是 Tensor。
- **RuntimeError** - 在 Ascend 硬件平台下，设置 PYNATIVE 或 KBK 模式时，*bias* 不是 1D 且 *input* 大于 6D。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([[-1., 1., 2.], [-3., -3., 1.]], mindspore.float32)
>>> weight = Tensor([[-2., -2., -2.], [0., -1., 0.]], mindspore.float32)
>>> bias = Tensor([0., 1.], mindspore.float32)
>>> output = mint.nn.functional.linear(input, weight, bias)
>>> print(output)
[[-4.  0.]
 [10.  4.]]
```

4.5.6 Dropout 函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.dropout</code>	在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些值归零，起到减少神经元相关性的作用，避免过拟合。	Ascend
<code>mindspore.mint.nn.functional.dropout2d</code>	在训练期间，根据伯努利分布，以概率 p 随机将输入 Tensor 的某些通道置零（对于 shape 为 (N, C, H, W) 的四维 Tensor，其通道特征图指的是后两维 shape 为 (H, W) 的二维特征图）。	Ascend

mindspore.mint.nn.functional.dropout

`mindspore.mint.nn.functional.dropout` (*input*, $p=0.5$, *training=True*, *inplace=False*)

在训练期间，以服从伯努利分布的概率 p 随机将输入 Tensor 的某些值归零，起到减少神经元相关性的作用，避免过拟合。并且训练过程中返回值会乘以 $\frac{1}{1-p}$ 。在推理过程中，此层返回与 *input* 相同的 Tensor。

参数：

- **input** (Tensor) - 输入 Tensor，shape 为 $(*, N)$ 。
- **p** (float, 可选) - 输入神经元丢弃概率，数值范围在 0 到 1 之间。例如， $p=0.1$ ，删除 10% 的神经元。默认值：0.5。
- **training** (bool, 可选) - 若为 True 则启用 dropout 功能，若为 False 则直接返回输入，并且 p 无效。默认值：True。
- **inplace** (bool, 可选) - 若为 True 则启用原地更新功能。默认值：False。

返回：

- **output** (Tensor) - 归零后的 Tensor，shape 和数据类型与 *input* 相同。

异常：

- **TypeError** - p 不是 float。
- **TypeError** - *input* 不是 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor(((20, 16), (50, 50)), mindspore.float32)
>>> output = mint.nn.functional.dropout(input, p=0.5)
>>> print(output.shape)
(2, 2)
```

`mindspore.mint.nn.functional.dropout2d`

`mindspore.mint.nn.functional.dropout2d(input, p=0.5, training=True)`

在训练期间，根据伯努利分布，以概率 p 随机将输入 Tensor 的某些通道置零（对于 shape 为 (N, C, H, W) 的四维 Tensor，其通道特征图指的是后两维 shape 为 (H, W) 的二维特征图）。

例如，在批处理输入中， i_th 批的 j_th 通道有一个待处理的 2D Tensor $input[i, j]$ 。每个通道在每次前向传播时，将会独立地根据伯努利分布的概率 p 来确定是否被置零。论文 [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#) 中提出了该技术，并证明其能有效地减少过度拟合，防止神经元共适应。更多详细信息，请参见 [Improving neural networks by preventing co-adaptation of feature detectors](#)。

`dropout2d` 可以提高通道特征映射之间的独立性。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - 一个 shape 为 (N, C, H, W) 的四维 Tensor，其中 N 是批处理大小， C 是通道数， H 是特征高度， W 是特征宽度。
- **p** (float, 可选) - 通道的丢弃概率，介于 0 和 1 之间。例如 $p = 0.8$ ，意味着 80% 的清零概率。默认值：0.5。
- **training** (bool, 可选) - 如果 training 为 True，则执行对 *input* 的某些通道概率清零的操作。否则，不执行此操作。默认值：True。

返回：

Tensor，输出，具有与输入 *input* 相同的 shape 和数据类型。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *p* 的数据类型不是 float。
- **ValueError** - *p* 值不在 $[0.0, 1.0]$ 之间。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.ones([2, 1, 2, 3]), mindspore.float32)
>>> output = mint.nn.functional.dropout2d(input, 0.5)
>>> print(output.shape)
(2, 1, 2, 3)
```

4.5.7 稀疏函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.embedding</code>	以 <code>input</code> 中的值作为索引，从 <code>weight</code> 中查询对应的 <code>embedding</code> 向量。	Ascend
<code>mindspore.mint.nn.functional.one_hot</code>	返回一个 one-hot 类型的 Tensor。	Ascend

mindspore.mint.nn.functional.embedding

`mindspore.mint.nn.functional.embedding` (*input*, *weight*, *padding_idx=None*, *max_norm=None*, *norm_type=2.0*, *scale_grad_by_freq=False*)

以 *input* 中的值作为索引，从 *weight* 中查询对应的 `embedding` 向量。

警告：在 Ascend 后端，*input* 的值非法将导致不可预测的行为。

参数：

- **input** (Tensor) - 用于检索的索引输入。取值范围：`[0, weight.shape[0])`。
- **weight** (Union[Parameter, Tensor]) - 用于检索的数据。必须是 2D 输入。
- **padding_idx** (int, 可选) - 如果给定非 `None` 值，则 *weight* 中对应的行在反向计算中不会更新。取值范围：`[-weight.shape[0], weight.shape[0])`。默认值：`None`。
- **max_norm** (float, 可选) - 如果给定非 `None` 值，则先求出 *input* 指定位置的 *weight* 的 p-范数结果 *result* (p 的值通过 *norm_type* 指定)，然后对 *result > max_norm* 位置的 *weight* 进行原地更新，更新公式： $\frac{max\_norm}{result+1e^{-7}}$ 。默认值：`None`。
- **norm_type** (float, 可选) - 指定 p-范数计算中的 p 值。默认值：`2.0`。
- **scale_grad_by_freq** (bool, 可选) - 如果值为 `True`，则反向梯度值会按照 *input* 中索引值重复的次数进行缩放。默认值：`False`。

返回：

Tensor，数据类型与 *weight* 保持一致，shape 为 `(*input.shape, weight.shape[1])`。

异常：

- **ValueError** - *padding_idx* 的取值不在有效范围。
- **ValueError** - *weight.shape* 非法。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, Parameter, mint
>>> input = Tensor([[1, 0, 1, 1], [0, 0, 1, 0]])
>>> weight = Parameter(np.random.randn(3, 3).astype(np.float32))
>>> output = mint.nn.functional.embedding(input, weight, max_norm=0.4)
```

(下页继续)

(续上页)

```
>>> print(output)
[[ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01],
 [ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01],
 [ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01],
 [ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01]],
 [[ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01],
 [ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01],
 [ 5.49015924e-02,  3.47811311e-01, -1.89771220e-01],
 [ 2.09307984e-01, -2.24846993e-02,  3.40124398e-01]]]
```

mindspore.mint.nn.functional.one_hot

`mindspore.mint.nn.functional.one_hot(tensor, num_classes=-1)`

返回一个 one-hot 类型的 Tensor。

生成一个新的 Tensor，由索引 *tensor* 表示的位置取值为 1，而在其他所有位置取值为 0。

参数：

- **tensor** (Tensor) - 输入索引，shape 为 $(X_0, \dots, X_n)$ 的 Tensor。数据类型必须为 int32 或 int64。维度不能大于 7。
- **num_classes** (int, 可选) - 输入的 Scalar，定义 one-hot 的深度，默认值：-1。

返回：

Tensor，one-hot 类型的 Tensor。

异常：

- **TypeError** - *num_classes* 的数据类型不是 int。
- **TypeError** - *tensor* 的数据类型不是 int32 或者 int64。
- **ValueError** - *num_classes* 的输入值小于-1。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> tensor = Tensor(np.array([0, 1, 2]), mindspore.int32)
>>> num_classes = 3
>>> output = mint.nn.functional.one_hot(tensor, num_classes)
>>> print(output)
[[1 0 0]
 [0 1 0]
 [0 0 1]]
```

4.5.8 损失函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.cross_entropy</code>	获取预测值和目标值之间的交叉熵损失。	Ascend
<code>mindspore.mint.nn.functional.binary_cross_entropy</code>	计算预测值 <i>input</i> 和目标值 <i>target</i> 之间的二值交叉熵（度量两个概率分布间的差异性信息）损失。	Ascend
<code>mindspore.mint.nn.functional.binary_cross_entropy_with_logits</code>	输入经过 <code>sigmoid</code> 激活函数后作为预测值, <code>binary_cross_entropy_with_logits</code> 计算预测值和目标值之间的二值交叉熵损失。	Ascend
<code>mindspore.mint.nn.functional.kl_div</code>	计算输入 <i>input</i> 和 <i>target</i> 的 Kullback-Leibler 散度。	Ascend
<code>mindspore.mint.nn.functional.l1_loss</code>	用于计算预测值和目标值之间的平均绝对误差。	Ascend
<code>mindspore.mint.nn.functional.mse_loss</code>	计算预测值和标签值之间的均方误差。	Ascend
<code>mindspore.mint.nn.functional.nll_loss</code>	获取预测值和目标值之间的负对数似然损失。	Ascend
<code>mindspore.mint.nn.functional.smooth_l1_loss</code>	计算平滑 L1 损失, 该 L1 损失函数有稳健性。	Ascend

mindspore.mint.nn.functional.cross_entropy

`mindspore.mint.nn.functional.cross_entropy` (*input, target, weight=None, ignore_index=-100, reduction='mean', label_smoothing=0.0*)

获取预测值和目标值之间的交叉熵损失。

`cross_entropy` 方法支持两种不同的目标值 (*target*):

- 类别索引 (int), 取值范围为 $[0, C)$, 其中 C 为类别数, 当 `reduction` 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})} \cdot \mathbb{1}\{y_n \neq \text{ignore\_index}\}$$

其中, x 表示预测值, y 表示目标值, w 表示权重, N 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 `reduction` 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{1}{\sum_{n=1}^N w_{y_n} \cdot \mathbb{1}\{y_n \neq \text{ignore\_index}\}} \sum_{n=1}^N l_n, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

- 类别概率 (float), 用于目标值为多个类别标签的情况。当 `reduction` 为 'none' 时, 交叉熵损失公式如下:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad l_n = - \sum_{c=1}^C w_c \log \frac{\exp(x_{n,c})}{\sum_{i=1}^C \exp(x_{n,i})} y_{n,c}$$

其中, x 表示预测值, y 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 `reduction` 不为 'none' (默认为 'mean'), 则

$$\ell(x, y) = \begin{cases} \frac{\sum_{n=1}^N l_n}{N}, & \text{if reduction = 'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'}. \end{cases}$$

警告：这是一个实验性 API，后续可能修改或删除。

说明：动态 shape、动态 rank 和可变常量输入不支持在 严格图模式 (jit_syntax_level=mindspore.STRICT) 下执行。

参数：

- **input** (Tensor) - 输入预测值，shape 为 (N) 或 (N, C) 或 (N, C, H, W) (针对二维数据)，或 $(N, C, d_1, d_2, \dots, d_K)$ (针对高维数据)。input 需为对数概率。数据类型仅支持 float16 或 float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 输入目标值。若目标值为类别索引，则 shape 为 $()$ 、 (N) 或 $(N, d_1, d_2, \dots, d_K)$ ，数据类型仅支持 int32 或 int64。若目标值为类别概率，则 shape 为 $(N,)$ 、 (N, C) 或 $(N, C, d_1, d_2, \dots, d_K)$ ，数据类型仅支持 float16 或 float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。
- **weight** (Tensor, 可选) - 指定各类别的权重。若值不为 None，则 shape 为 $(C,)$ 。数据类型仅支持 float16 或 float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。默认值：None。
- **ignore_index** (int, 可选) - 指定 target 中需要忽略的值(一般为填充值)，使其不对梯度产生影响。仅在目标值为类别索引下生效，在类别概率下请设置为负数。默认值：-100。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。
- **label_smoothing** (float, 可选) - 标签平滑值，用于计算 Loss 时防止模型过拟合的正则化手段。取值范围为 $[0.0, 1.0]$ 。默认值：0.0。

返回：

Tensor，数据类型与 input 相同。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import mint, Tensor
>>> import numpy as np
>>> # Case 1: Indices labels
>>> inputs = Tensor(np.random.randn(3, 5), ms.float32)
>>> target = Tensor(np.array([1, 0, 4]), ms.int32)
>>> output = mint.nn.functional.cross_entropy(inputs, target)
>>> # Case 2: Probability labels
>>> inputs = Tensor(np.random.randn(3, 5), ms.float32)
>>> target = Tensor(np.random.randn(3, 5), ms.float32)
>>> output = mint.nn.functional.cross_entropy(inputs, target)
```

mindspore.mint.nn.functional.binary_cross_entropy

`mindspore.mint.nn.functional.binary_cross_entropy(input, target, weight=None, reduction='mean')`

计算预测值 *input* 和目标值 *target* 之间的二值交叉熵（度量两个概率分布间的差异性信息）损失。

将 *input* 设置为 x ，*target* 设置为 y ，输出设置为 $\ell(x, y)$ ，第 n 个 batch 二值交叉熵的权重为 w_n 。则，

$$L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

其中， L 表示所有 `batch_size` 的 loss 值， l 表示一个 `batch_size` 的 loss 值， n 表示在 $1 - N$ 范围内的一个 `batch_size`。

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

警告：

- *input* 的值必须要在 0-1 范围内。

参数：

- **input** (Tensor) - 输入预测值，其数据类型为 float16 或 float32。
- **target** (Tensor) - 输入目标值，shape 与 *input* 相同。数据类型为 float16 或 float32。
- **weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。支持广播，使其 shape 与 *input* 的 shape 保持一致。数据类型必须为 float16 或 float32。默认值：None。若为 None，损失函数将不会考虑任何样本的权重，每个样本在计算损失时被视为具有相同的重要性。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。

返回：

Tensor 或 Scalar，如果 *reduction* 为 'none'，则为 shape 和数据类型与输入 *input* 相同的 Tensor。否则，输出为 Scalar。

异常：

- **TypeError** - 输入 *input*，*target*，*weight* 不为 Tensor。
- **TypeError** - 输入 *input*，*target*，*weight* 的数据类型既不是 float16 也不是 float32。
- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。
- **ValueError** - *target* 的 shape 大小与 *input* 或者 *weight* 不相同。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0.2, 0.7, 0.1]), mindspore.float32)
>>> target = Tensor(np.array([0., 1., 0.]), mindspore.float32)
>>> weight = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = mint.nn.functional.binary_cross_entropy(input, target, weight)
>>> print(output)
0.38240486
```

mindspore.mint.nn.functional.binary_cross_entropy_with_logits

`mindspore.mint.nn.functional.binary_cross_entropy_with_logits` (*input*, *target*, *weight=None*, *reduction='mean'*, *pos_weight=None*)

输入经过 `sigmoid` 激活函数后作为预测值, `binary_cross_entropy_with_logits` 计算预测值和目标值之间的二值交叉熵损失。与 `mindspore.ops.binary_cross_entropy_with_logits()` 功能一致。

将输入 *input* 设置为 *X*, 输入 *target* 设置为 *Y*, 输入 *weight* 设置为 *W*, 输出设置为 *L*。则,

$$p_{ij} = \text{sigmoid}(X_{ij}) = \frac{1}{1+e^{-X_{ij}}}$$

$$L_{ij} = -[Y_{ij} \log(p_{ij}) + (1 - Y_{ij}) \log(1 - p_{ij})]$$

i 表示 i^{th} 样例, *j* 表示类别。则,

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

ℓ 表示计算损失的方法。有三种方法: 第一种方法是直接提供损失值, 第二种方法是计算所有损失的平均值, 第三种方法是计算所有损失的总和。

该算子会将输出乘以相应的权重。*weight* 表示一个 batch 中的每条数据分配不同的权重, *pos_weight* 为每个类别的正例子添加相应的权重。

此外, 它可以通过向正例添加权重来权衡召回率和精度。在多标签分类的情况下, 损失可以描述为:

$$p_{ij,c} = \text{sigmoid}(X_{ij,c}) = \frac{1}{1+e^{-X_{ij,c}}}$$

$$L_{ij,c} = -[P_c Y_{ij,c} * \log(p_{ij,c}) + (1 - Y_{ij,c}) \log(1 - p_{ij,c})]$$

其中 *c* 是类别数目 (*c*>1 表示多标签二元分类, *c*=1 表示单标签二元分类), *n* 是批次中样本的数量, P_c 是第 *c* 类正例的权重。 $P_c > 1$ 增大召回率, $P_c < 1$ 增大精度。

参数:

- **input** (Tensor) - 输入预测值, shape (*N*, *), 其中 * 代表任意数量的附加维度。其数据类型为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 输入目标值, shape 与 *input* 相同。数据类型为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。
- **weight** (Tensor, 可选) - 指定每个批次二值交叉熵的权重。支持广播, 使其 shape 与 *target* 的 shape 保持一致。数据类型必须为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。默认值: None, *weight* 是值为 1 的 Tensor。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。

- 'none': 不应用规约方法。
- 'mean': 计算输出元素的加权平均值。
- 'sum': 计算输出元素的总和。
- **pos_weight** (Tensor, 可选) - 指定正类的权重。是一个长度等于分类数的向量。支持广播, 使其 `shape` 与 `target` 的 `shape` 保持一致。数据类型必须为 `float16`、`float32` 或 `bfloat16` (仅 Atlas A2 训练系列产品支持)。默认值: `None`, 此时 `pos_weight` 等价于值为 1 的 Tensor。

返回:

Tensor 或 Scalar, 如果 `reduction` 为 'none', 则为 `shape` 和数据类型与输入 `target` 相同的 Tensor。否则, 输出为 Scalar。

异常:

- **TypeError** - 输入 `input`, `target`, `weight`, `pos_weight` 不为 Tensor。
- **TypeError** - `reduction` 输入数据类型不为 string。
- **ValueError** - `weight` 或 `pos_weight` 不能广播到 `shape` 为 `input` 的 Tensor。
- **ValueError** - `reduction` 不为 'none'、'mean' 或 'sum'。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([[[-0.8, 1.2, 0.7], [-0.1, -0.4, 0.7]]], mindspore.float32)
>>> target = Tensor(np.array([[0.3, 0.8, 1.2], [-0.6, 0.1, 2.2]]), mindspore.float32)
>>> weight = Tensor(np.array([1.0, 1.0, 1.0]), mindspore.float32)
>>> pos_weight = Tensor(np.array([1.0, 1.0, 1.0]), mindspore.float32)
>>> output = mint.nn.functional.binary_cross_entropy_with_logits(input, target, weight, 'mean', pos_weight)
>>> print(output)
0.3463612
```

`mindspore.mint.nn.functional.kl_div`

`mindspore.mint.nn.functional.kl_div(input, target, reduction='mean', log_target=False)`

计算输入 `input` 和 `target` 的 Kullback-Leibler 散度。

对于相同 `shape` 的 Tensor `x` 和 `y`, `KLDivLoss` 的计算公式如下:

$$L(x, y) = y \cdot (\log y - x)$$

则

$$\ell(x, y) = \begin{cases} L(x, y), & \text{if reduction = 'none';} \\ \text{mean}(L(x, y)), & \text{if reduction = 'mean';} \\ \text{sum}(L(x, y))/x.\text{shape}[0], & \text{if reduction = 'batchmean';} \\ \text{sum}(L(x, y)), & \text{if reduction = 'sum'}. \end{cases}$$

其中 `x` 表示 `input`, `y` 表示 `target`, $\ell(x, y)$ 表示输出。

说明：仅当 *reduction* 设置为 "batchmean" 时，输出才与 Kullback-Leibler 散度的数学定义一致。

参数：

- **input** (Tensor) - 输入 Tensor。数据类型必须为 float16、float32 或 bfloat16（仅 Atlas A2 训练系列产品支持）。
- **target** (Tensor) - 目标 Tensor，其数据类型与 *input* 相同。*target* 的 shape 和 *input* 的 shape 能广播。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式。默认值：'mean'。
- **log_target** (bool, 可选) - 指定输入 *target* 是否在对数空间内。默认值：False。

返回：

Tensor，数据类型与 *input* 相同。如果 *reduction* 为 'none'，则 shape 与 *input* 和 *target* 广播之后的结果相同。否则，输出为 Scalar 的 Tensor。

异常：

- **TypeError** - *input* 或 *target* 不是 Tensor。
- **TypeError** - *input* 或 *target* 的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - *target* 的数据类型与 *input* 不同。
- **ValueError** - *reduction* 不为 'none'、'mean'、'sum' 或 'batchmean' 之一。
- **ValueError** - *target* 的 shape 和 *input* 的 shape 不能广播。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> import numpy as np
>>> input = ms.Tensor(np.array([[0.5, 0.5], [0.4, 0.6]]), ms.float32)
>>> target = ms.Tensor(np.array([[0., 1.], [1., 0.]]), ms.float32)
>>> output = mint.nn.functional.kl_div(input, target, reduction='mean', log_target=False)
>>> print(output)
-0.225
```

mindspore.mint.nn.functional.l1_loss

mindspore.mint.nn.functional.l1_loss (*input*, *target*, *reduction*='mean')

用于计算预测值和目标值之间的平均绝对误差。

假设 *x* 和 *y* 为预测值和目标值，均为一维 Tensor，长度 *N*，*reduction* 设置为 'none'，则计算 *x* 和 *y* 的 loss 不进行降维操作。

公式如下：

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^T, \quad \text{with } l_n = |x_n - y_n|$$

其中，*N* 为 batch size。

如果 *reduction* 是 'mean' 或者 'sum' , 则:

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction} = \text{'mean'}; \\ \text{sum}(L), & \text{if reduction} = \text{'sum'}. \end{cases}$$

参数:

- **input** (Tensor) - 预测值, 任意维度的 Tensor。
- **target** (Tensor) - 目标值, 通常情况与 *input* 的 shape 相同。如果 *input* 和 *target* 的 shape 不同, 需要保证他们之间可以互相广播。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。注意: 当 *reduction* 为 'mean' 时, *input* 和 *target* 的数据类型需要至少有一个是 float。
 - 'sum': 计算输出元素的总和。

返回:

Tensor 或 Scalar, 如果 *reduction* 为 'none', 则返回与 *input* 具有相同 shape 和 dtype 的 Tensor。否则, 将返回 Scalar。

异常:

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *target* 不是 Tensor。
- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。

支持平台:

Ascend

样例:

```
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> x = Tensor([[1, 2, 3], [4, 5, 6]], mstype.float32)
>>> target = Tensor([[6, 5, 4], [3, 2, 1]], mstype.float32)
>>> output = mint.nn.functional.l1_loss(x, target, reduction="mean")
>>> print(output)
3.0
```

mindspore.mint.nn.functional.mse_loss

`mindspore.mint.nn.functional.mse_loss(input, target, reduction='mean')`

计算预测值和标签值之间的均方误差。

更多参考详见 `mindspore.mint.nn.MSELoss`。

参数:

- **input** (Tensor) - 任意维度的 Tensor。数据类型和 *target* 需要一致。
- **target** (Tensor) - 输入标签, 任意维度的 Tensor。数据类型和 *input* 需要一致。支持在 *input* 和 *target* shape 不相同的情况下, 通过广播保持一致。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。

- 'none': 不应用规约方法。
- 'mean': 计算输出元素的平均值。
- 'sum': 计算输出元素的总和。

返回:

- Tensor。如果 *reduction* 为 'mean' 或 'sum' 时, *shape* 为 *Tensor Scalar*。
- 如果 *reduction* 为 'none', 输出的 *shape* 则是 *input* 和 *target* 广播之后的 *shape*。

异常:

- **ValueError** - 如果 *reduction* 不为 'mean', 'sum' 或 'none' 中的一个。
- **ValueError** - 如果 *input* 和 *target* 无法广播。
- **TypeError** - 如果 *input* 和 *target* 数据类型不一致。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> logits = Tensor(np.array([[1, 2, 3]]), mindspore.float32)
>>> labels = Tensor(np.array([[1, 1, 1], [1, 2, 2]]), mindspore.float32)
>>> output = mint.nn.functional.mse_loss(logits, labels, reduction='none')
>>> print(output)
[[0. 1. 4.]
 [0. 0. 1.]]
```

mindspore.mint.nn.functional.nll_loss

`mindspore.mint.nn.functional.nll_loss (input, target, weight=None, ignore_index=-100, reduction='mean')`

获取预测值和目标值之间的负对数似然损失。

reduction 为 'none' 时, 负对数似然损失公式如下:

$$\ell(x, t) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{t_n} x_{n, t_n}, \quad w_c = \text{weight}[c] \cdot \mathbb{1}\{c \neq \text{ignore\_index}\},$$

其中, x 表示预测值, t 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

若 *reduction* 不为 'none' (默认为 'mean'), 则

$$\ell(x, t) = \begin{cases} \frac{\sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{t_n}} l_n, & \text{if reduction} = \text{'mean'}, \\ \sum_{n=1}^N l_n, & \text{if reduction} = \text{'sum'} \end{cases}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **input** (Tensor) - 输入预测值, *shape* 为 (N) 或 (N, C) , C 表示类的数量, N 表示 batch size, 或 $(N, C, d_1, d_2, \dots, d_K)$ (针对高维数据)。 *input* 需为对数概率。数据类型仅支持 float32 或 float16 或 bfloat16(仅 Atlas A2 训练系列产品支持)。

- **target** (Tensor) - 输入目标值, shape 为 () 或 (N), 限定范围为 $[0, C - 1]$, 或 $(N, d_1, d_2, \dots, d_K)$ (针对高维数据)。数据类型支持 int32、int64、uint8。
- **weight** (Tensor, 可选) - 指定各类别的权重。若值不为 None, 则 shape 为 (C,)。数据类型仅支持 float16 或 float32 或 bfloat16(仅 Atlas A2 训练系列产品支持)。要求与 *input* 的数据类型保持一致。默认值: None。
- **ignore_index** (int, 可选) - 指定 **target** 中需要忽略的值 (一般为填充值), 使其不对梯度产生影响。默认值: -100。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。

返回:

Tensor, 数据类型与 *input* 相同。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = mindspore.Tensor(np.random.randn(3, 5), mindspore.float32)
>>> target = mindspore.Tensor(np.array([1, 0, 4]), mindspore.int32)
>>> output = mint.nn.functional.nll_loss(input, target)
```

mindspore.mint.nn.functional.smooth_l1_loss

`mindspore.mint.nn.functional.smooth_l1_loss(input, target, reduction='mean', beta=1.0)`

计算平滑 L1 损失, 该 L1 损失函数有稳健性。

平滑 L1 损失是一种类似于 MSELoss 的损失函数, 但对异常值相对不敏感, 可以参阅论文 [Fast R-CNN](#)。

给定长度为 N 的两个输入 x, y , 平滑 L1 损失的计算如下:

$$L_i = \begin{cases} \frac{0.5(x_i - y_i)^2}{\beta}, & \text{if } |x_i - y_i| < \beta \\ |x_i - y_i| - 0.5 * \beta, & \text{otherwise.} \end{cases}$$

当 *reduction* 不是设定为 *none* 时, 计算如下:

$$L = \begin{cases} \text{mean}(L_i), & \text{if reduction = 'mean'}; \\ \text{sum}(L_i), & \text{if reduction = 'sum'}. \end{cases}$$

其中, β 控制损失函数在线性与二次间变换的阈值, $\beta \geq 0$, 默认值是 1.0。 N 为 batch size。

警告: 这是一个实验性 API, 后续可能修改或删除。

说明：参数 *input* 和 *target* 遵循隐式类型转换规则，使数据类型保持一致。如果两参数数据类型不一致，则低精度类型会被转换成较高精度类型。

参数：

- **input** (Tensor) - shape: $(N, *)$ ，其中 $*$ 表示任意数量的附加维度。支持数据类型：
 - Ascend: float16、float32、bfloat16。
- **target** (Tensor) - shape: $(N, *)$ ，与 *input* 的 shape 相同。支持数据类型：
 - Ascend: float16、float32、bfloat16。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。
 - 'sum': 计算输出元素的总和。
- **beta** (number, 可选) - 控制损失函数在 L1Loss 和 L2Loss 间变换的阈值，该值必须大于等于 0。默认值：1.0。

返回：

Tensor，数据类型与 *input* 相同。如果 *reduction* 为 'none'，则输出为 Tensor 且与 *input* 的 shape 相同。否则 shape 为 $()$ 。

异常：

- **TypeError** - *input* 或 *target* 不是 Tensor。
- **RuntimeError** - *input* 或 *target* 的数据类型不是 float16，float32 和 bfloat16 中的任一者。
- **ValueError** - *input* 与 *target* 的 shape 不同。
- **ValueError** - *reduction* 不是 'none'，'mean' 和 'sum' 中的任一者。
- **TypeError** - *beta* 不是 float，bool 或 int。
- **RuntimeError** - *beta* 小于 0。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([2, 2, 3]), mindspore.float32)
>>> target = Tensor(np.array([2, 2, 2]), mindspore.float32)
>>> beta = 1.0
>>> reduction_1 = 'none'
>>> output = mint.nn.functional.smooth_l1_loss(input, target, reduction_1, beta)
>>> print(output)
[0.  0.  0.5]
>>> reduction_2 = 'mean'
>>> output = mint.nn.functional.smooth_l1_loss(input, target, reduction_2, beta)
```

(下页继续)

(续上页)

```
>>> print(output)
0.16666667
>>> reduction_3 = 'sum'
>>> output = mint.nn.functional.smooth_l1_loss(input, target, reduction_3, beta)
>>> print(output)
0.5
```

4.5.9 Vision 函数

接口名	概述	支持平台
<code>mindspore.mint.nn.functional.interpolate</code>	按照给定的 <i>size</i> 或 <i>scale_factor</i> 根据 <i>mode</i> 设置的插值方式，对输入 <i>input</i> 进行插值。	Ascend
<code>mindspore.mint.nn.functional.grid_sample</code>	给定一个输入和一个网格，使用网格中的输入值和像素位置计算输出。	Ascend
<code>mindspore.mint.nn.functional.pad</code>	根据参数 <i>pad</i> 对输入进行填充。	Ascend
<code>mindspore.mint.nn.functional.pixel_shuffle</code>	根据上采样系数重排 Tensor 中的元素。	Ascend

mindspore.mint.nn.functional.interpolate

`mindspore.mint.nn.functional.interpolate` (*input*, *size=None*, *scale_factor=None*, *mode='nearest'*, *align_corners=None*, *recompute_scale_factor=None*)

按照给定的 *size* 或 *scale_factor* 根据 *mode* 设置的插值方式，对输入 *input* 进行插值。

警告：这是一个实验性 API，后续可能修改或删除。

说明：

- 在 *linear* 模式下，*align_corners* 为 *False* 时不支持 *scale_factor*。
- 在 *nearest* 模式下，在输入为 3-D/4-D **Tensor** 图像按 *scale_factor* 进行缩放的过程中可能存在精度问题。
- mode* 和 *recompute_scale_factor* 只能为常量。

参数：

- input** (**Tensor**) - 被调整大小的 **Tensor**。输入向量必须为三维，四维或五维，*shape* 为 (*N*, *C*, [*optionalD*], [*optionalH*], *W*)，数据类型为 *float*。
- size** (*Union[int, tuple[int], list[int]]*, 可选) - 目标大小。如果 *size* 为 *tuple* 或 *list*，那么其长度应该和 *input* 去掉 *N*, *C* 的维度相同。*size* 和 *scale_factor* 同时只能指定一个。默认值：*None*。
- scale_factor** (*Union[float, tuple[float], list[float]]*, 可选) - 每个维度的缩放系数。如果 *scale_factor* 为 *tuple* 或 *list*，那么其长度应该和 *input* 去掉 *N*, *C* 的维度相同。*size* 和 *scale_factor* 同时只能指定一个。默认值：*None*。
- mode** (*str*) - 采样算法。以下采样方式的一种，'nearest'(最近邻插值)，'linear'(线性插值，仅三维)，'bilinear'(双线性插值，仅四维)，'trilinear'(三线性插值，仅五维)，'bicubic'(双三次插值，仅四维)。默认值："nearest"。

- **align_corners** (bool, 可选) - 是否使用角对齐进行坐标映射。假设对输入 Tensor 沿 x 轴进行变换，具体计算公式如下：

```
ori_i = new_length != 1 ? new_i * (ori_length - 1) / (new_length - 1) : 0    # 'align_
↪corners' 为 True

ori_i = new_length > 1 ? (new_i + 0.5) * ori_length / new_length - 0.5 : 0    # 'align_
↪corners' 为 False
```

其中, *ori_length* 与 *new_length* 分别表示 Tensor 在 x 轴方向上转换前、后的长度, *new_i* 表示转换后沿 x 轴第 *i* 个元素的坐标, *ori_i* 表示沿 x 轴原始数据的对应坐标。

此选项只对 'linear'、'bilinear' 和 'bicubic' 模式有效，默认值：None。

- **recompute_scale_factor** (bool, 可选) - 重计算 *scale_factor*。如果为 True，会使用参数 *scale_factor* 计算参数 *size*，最终使用 *size* 的值进行缩放。如果为 False，将使用 *size* 或 *scale_factor* 直接进行插值。默认值：None。

参数支持列表和支持平台：

mode	input.dim	align_corners	scale_factor	device
nearest	3	-	√	Ascend
	4	-	√	Ascend
	5	-	√	Ascend
linear	3	√	√	Ascend
bilinear	4	√	×	Ascend
bicubic	4	√	×	Ascend
trilinear	5	√	√	Ascend

- - 表示无此参数。
- × 表示当前不支持此参数。
- √ 表示当前支持此参数。

返回：

采样后的 Tensor，维度和数据类型与 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **ValueError** - *size* 和 *scale_factor* 都不为空。
- **ValueError** - *size* 和 *scale_factor* 都为空。
- **ValueError** - *size* 为元组或列表类型时长度不等于 *input.ndim - 2*。
- **ValueError** - *scale_factor* 为元组或列表类型时长度不等于 *input.ndim - 2*。
- **ValueError** - *mode* 不在模式支持列表中。
- **ValueError** - *input.ndim* 不在模式对应维度的支持列表中。
- **ValueError** - *size* 不为空，*recompute_scale_factor* 不为空。
- **ValueError** - *scale_factor* 不在对应的支持列表中。
- **ValueError** - *align_corners* 不在对应的支持列表中。

支持平台：

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, mint
>>> input = Tensor([[[[1, 2, 3], [4, 5, 6]]], mindspore.float32)
>>> output = mint.nn.functional.interpolate(input, size=(6,), mode='nearest')
>>> print(output)
[[[1. 1. 2. 2. 3. 3.]
  [4. 4. 5. 5. 6. 6.]]]
```

mindspore.mint.nn.functional.grid_sample

`mindspore.mint.nn.functional.grid_sample(input, grid, mode='bilinear', padding_mode='zeros', align_corners=False)`

给定一个输入和一个网格，使用网格中的输入值和像素位置计算输出。*input* 只支持 4-D (GridSampler2D) 和 5-D (GridSampler3D)。

在 4-D 场景下, *input* 的 shape 为 (N, C, H_{in}, W_{in}) , *grid* 的 shape 为 $(N, H_{out}, W_{out}, 2)$, *output* 的 shape 为 (N, C, H_{out}, W_{out}) 。对于每个输出位置 $output[n, :, h, w]$, $grid[n, h, w]$ 指定 *input* 像素位置 x 和 y , 用于计算 $output[n, :, h, w]$ 的插值。以 5D 为例, $grid[n, d, h, w]$ 指定 x, y, z 像素位置的插值位置为 $[n, :, d, h, w]$ 。*mode* 参数指定 *nearest* 或 *bilinear* (bicubic 暂不支持) 插值法对输入像素进行采样。

grid 指定由 *input* 归一化的采样像素位置。因此, 它应该在 $[-1, 1]$ 范围内的值最多。

如果 *grid* 的值在 $[-1, 1]$ 范围之外, 则相应的输出将按照定义的 *padding_mode* 方式处理。如果 *padding_mode* 设置为 0, 则使用 0 来表示出界的网格位置。如果 *padding_mode* 设置为 *border*, 对于出界网格位置, 则使用 *border* 值。如果 *padding_mode* 设置为 *reflection*, 请使用边界所反映的位置的值用于指定出界网格位置。对于远离边界的位置, 它会一直被反射, 直到在边界内。

参数:

- **input** (Tensor) - 4-D 场景下, shape 为 (N, C, H_{in}, W_{in}) , 5-D 场景下, shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 。数据类型为 float32 或 float64。
- **grid** (Tensor) - 4-D 场景下, shape 为 $(N, H_{out}, W_{out}, 2)$, 5-D 场景下, shape 为 $(N, D_{out}, H_{out}, W_{out}, 3)$ 。数据类型与 *input* 保持一致。
- **mode** (str, 可选) - 插值方法。可选方法为 'bilinear', 'nearest'。默认值: 'bilinear'。注: 'bilinear' 还不支持。当 *mode* 为 'bilinear', 且输入为 5-D, 则 *mode* 为 'trilinear'。但是, 当输入为 4-D, 则 *mode* 为 'bilinear'。默认值: 'bilinear'。
 - 'nearest': 最近邻插值。每个输出像素的值为最近的输入像素的值。这种方法简单快速, 但可能导致块状或像素化的输出。
 - 'bilinear': 双线性插值。每个输出像素是最接近的四个输入像素的加权平均值, 使用双线性插值计算。与最近邻插值相比, 此方法产生更平滑的结果。
 - 'trilinear': 三线性插值。这是双线性插值在三维数据上的扩展。它在两个空间维度上执行双线性插值, 并沿第三个维度进行线性插值。通常用于体积或三维图像插值。
- **padding_mode** (str, 可选) - 填充方法。可选方法为 'zeros', 'border' 和 'reflection'。默认值: 'zeros'。
- **align_corners** (bool, 可选) - 如果设置成 True, -1 和 1 被视为引用输入角像素的中心点。如果设置为 False, 将被视为引用到输入角像素的角点, 使采样更不受分辨率影响。默认值为 False。

返回:

Tensor, 数据类型与 *input* 相同, 4-D 场景下, shape 为 (N, C, H_{out}, W_{out}) , 5-D 场景下, shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 。
异常:

- **TypeError** - 如果 *input* 或 *grid* 不是 Tensor 类型。
- **TypeError** - 如果 *input* 和 *grid* 的数据类型不一致。
- **TypeError** - 如果 *input* 或 *grid* 的数据类型无效。
- **TypeError** - 如果 *align_corners* 不是一个布尔值。
- **ValueError** - 如果 *input* 或 *grid* 的维度不是四维或五维。
- **ValueError** - 如果 *input* 的第一个维度不等于 *grid* 的第一个维度。
- **ValueError** - 如果 *grid* 最后一个维度不等于 2 (4-D 场景) 或者 3 (5-D 场景)。
- **ValueError** - 如果 *mode* 不是 *bilinear*, *nearest*, 数据类型不为 String。
- **ValueError** - 如果 *padding_mode* 不是 *zeros*, *border*, *reflection*, 数据类型不为 String。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input_x = Tensor(np.arange(16).reshape((2, 2, 2, 2)).astype(np.float32))
>>> grid = Tensor(np.arange(0.2, 1, 0.1).reshape((2, 2, 1, 2)).astype(np.float32))
>>> output = mint.nn.functional.grid_sample(input_x, grid, mode='bilinear', padding_mode=
↳ 'zeros',
...                                     align_corners=True)
>>> print(output)
[[[ [ 1.9      ]
    [ 2.1999998]]
  [ [ 5.9      ]
    [ 6.2      ]]]
 [[ [10.5     ]
    [10.8     ]]
  [ [14.5     ]
    [14.8     ]]]]
```

mindspore.mint.nn.functional.pad

`mindspore.mint.nn.functional.pad(input, pad, mode='constant', value=None)`

根据参数 *pad* 对输入进行填充。

警告: *circular* 模式性能较差, 不推荐使用。

参数:

- **input** (Tensor) - 输入 Tensor, shape 为 $(N, *)$, * 代表任意附加维度。
- **pad** (Union[tuple[int], list[int], Tensor]) - pad 的填充位置。 $\left\lfloor \frac{\text{len(pad)}}{2} \right\rfloor$ 维度的 *input* 将会被填充。可根据以下示例以此类推:
 - 示例: 若只需要填充输入 tensor 的最后一个维度, 则 *pad* 的填充方式为 (padding_left, padding_right);

- 示例: 若只需要填充输入 `tensor` 的最后两个维度, 则 `pad` 的填充方式为 (`padding_left`, `padding_right`, `padding_top`, `padding_bottom`);
- 示例: 若只需要填充输入 `tensor` 的最后三个维度, 则 `pad` 的填充方式为 (`padding_left`, `padding_right`, `padding_top`, `padding_bottom`, `padding_front`, `padding_back`);
- **mode** (str, 可选) - `pad` 的填充模式, 可选择 'constant'、'reflect'、'replicate' 或者 'circular'。默认值: 'constant'。
 - 对于 'constant' 模式, 请参考 [mindspore.nn.ConstantPad1d](#) 作为示例来理解这个填充模式, 并将这个模式扩展到 `n` 维。
 - 对于 'reflect' 模式, 请参考 [mindspore.nn.ReflectionPad1d](#) 作为示例来理解这个填充模式, `reflect` 模式用于填充四维或五维输入的最后三个维度、三维或四维输入的最后两个维度, 或者二维或三维输入的最后两个维度。
 - 对于 'replicate' 模式, 请参考 [mindspore.nn.ReplicationPad1d](#) 作为示例来理解这个填充模式, `replicate` 模式用于填充四维或五维输入的最后三个维度、三维或四维输入的最后两个维度, 或者二维或三维输入的最后两个维度。
 - 对于 'circular' 模式, `circular` 模式用于将图像的像素从一侧循环地填充到另一侧。例如, 右侧的像素将被替换为左侧的像素, 底部的像素将被替换为顶部的像素。`circular` 模式用于填充四维或五维输入的最后三个维度、三维或四维输入的最后两个维度, 或者二维或三维输入的最后两个维度。
- **value** (Union[int, float, None], 可选) - 仅在 'constant' 模式下生效, 设置在 'constant' 模式下的填充值, 如果值为 None, 则会使用 0 作为默认填充值。默认值: None。

返回:

填充后的 Tensor。

异常:

- **TypeError** - `pad` 不是全为 int 的 tuple 或者 list。
- **TypeError** - `input` 不是 Tensor。
- **ValueError** - `pad` 的长度不为偶数。
- **ValueError** - `pad` 的长度大于 6。
- **ValueError** - `mode` 不为 'constant' 并且 `value` 既不是 None 也不是 0。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import mint
>>> import numpy as np
>>> x = ms.Tensor(np.arange(1 * 2 * 2 * 2).reshape((1, 2, 2, 2)), dtype=ms.float64)
>>> output = mint.nn.functional.pad(x, [1, 0, 0, 1], mode='constant', value=6.0)
>>> print(output)
[[[6. 0. 1.]
  [6. 2. 3.]
  [6. 6. 6.]]
 [6. 4. 5.]
 [6. 6. 7.]
 [6. 6. 6.]]]
```


mindspore.mint.nn.functional.pixel_shuffle

`mindspore.mint.nn.functional.pixel_shuffle(input, upscale_factor)`

根据上采样系数重排 Tensor 中的元素。

将 shape 为 $(*, C \times r^2, H, W)$ 的 Tensor 重排 shape 为 $(*, C, H \times r, W \times r)$ 的 Tensor，其中 r 是放大因子。

这是步长为 $1/r$ 的子像素卷积。

关于 pixel_shuffle 算法详细介绍，请参考 [Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network](#)。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **input** (Tensor) - Tensor，shape 为 $(*, C \times r^2, H, W)$ 。input 的维度需要大于 2，并且倒数第三维 length 可以被 *upscale_factor* 的平方整除。
- **upscale_factor** (int) - 打乱输入 Tensor 的因子，是正整数。*upscale_factor* 是上面提到的 r 。

返回：

- **output** (Tensor) - Tensor，shape 为 $(*, C, H \times r, W \times r)$ 。

异常：

- **ValueError** - *upscale_factor* 不是正整数。
- **ValueError** - *input* 倒数第三维度的 length 不能被 *upscale_factor* 的平方整除。
- **ValueError** - *input* 维度小于 3。

支持平台：

Ascend

样例：

```
>>> from mindspore import mint
>>> input = mint.randn(1, 9, 4, 4)
>>> output = mint.nn.functional.pixel_shuffle(input, 3)
>>> print(output.shape)
(1, 1, 12, 12)
```

4.6 mindspore.mint.optim

接口名	概述	支持平台
<code>mindspore.mint.optim.Adam</code>	Adaptive Moment Estimation (Adam) 算法的实现。	Ascend
<code>mindspore.mint.optim.AdamW</code>	Adaptive Moment Estimation Weight Decay(AdamW) 算法的实现。	Ascend
<code>mindspore.mint.optim.SGD</code>	随机梯度下降算法。	Ascend

4.6.1 mindspore.mint.optim.Adam

class mindspore.mint.optim.**Adam** (*params*, *lr*=1e-3, *betas*=(0.9, 0.999), *eps*=1e-8, *weight_decay*=0.0, *amsgrad*=False, *, *maximize*=False)

Adaptive Moment Estimation (Adam) 算法的实现。

更新公式如下：

```

input :  $\gamma$  (lr),  $\beta_1, \beta_2$  (betas),  $\theta_0$  (params),  $f(\theta)$  (objective)
          $\lambda$  (weight decay), amsgrad, maximize
initialize :  $m_0 \leftarrow 0$  (first moment),  $v_0 \leftarrow 0$  (second moment),  $\hat{v}_0^{max} \leftarrow 0$ 
for  $t = 1$  to ... do
    if maximize :
         $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ 
    else
         $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ 
    if  $\lambda \neq 0$ 
         $g_t \leftarrow g_t + \lambda \theta_{t-1}$ 
     $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
     $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ 
    if amsgrad
         $\hat{v}_t^{max} \leftarrow \max(\hat{v}_t^{max}, \hat{v}_t)$ 
         $\theta_t \leftarrow \theta_{t-1} - \gamma \hat{m}_t / (\sqrt{\hat{v}_t^{max}} + \epsilon)$ 
    else
         $\theta_t \leftarrow \theta_{t-1} - \gamma \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ 
return  $\theta_t$ 

```

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-3。
- **betas** (Tuple[float, float], 可选) - 动量矩阵的指数衰减率。参数范围 (0.0, 1.0)。默认值：(0.9, 0.999)。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值：1e-8。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：0.0。
- **amsgrad** (bool, 可选) - 是否使用 AMSGrad 算法。默认值：False。

关键字参数：

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常：

- **ValueError** - *lr* 不是 int、float 或 Tensor。
- **ValueError** - *lr* 小于 0。
- **ValueError** - *eps* 小于 0。

- **ValueError** - *betas* 范围不在 [0, 1) 之间。
- **ValueError** - *weight_decay* 小于 0。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import mint
>>> from mindspore import mint
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = mint.optim.Adam(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```


4.6.2 mindspore.mint.optim.AdamW

class mindspore.mint.optim.**AdamW** (*params*, *lr*=1e-3, *betas*=(0.9, 0.999), *eps*=1e-8, *weight_decay*=1e-2, *amsgrad*=False, *, *maximize*=False)

Adaptive Moment Estimation Weight Decay(AdamW) 算法的实现。

input : γ (lr), β_1, β_2 (betas), θ_0 (params), $f(\theta)$ (objective), ϵ (epsilon)

λ (weight decay), *amsgrad*, *maximize*

initialize : $m_0 \leftarrow 0$ (first moment), $v_0 \leftarrow 0$ (second moment), $\widehat{v}_0^{max} \leftarrow 0$

for $t = 1$ **to** ... **do**

if *maximize* :

$g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$

else

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$

$\theta_t \leftarrow \theta_{t-1} - \gamma \lambda \theta_{t-1}$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

if *amsgrad*

$\widehat{v}_t^{max} \leftarrow \max(\widehat{v}_t^{max}, \widehat{v}_t)$

$\theta_t \leftarrow \theta_t - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t^{max}} + \epsilon)$

else

$\theta_t \leftarrow \theta_t - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$

return θ_t

警告:

- 这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。
- 对于 Ascend，仅 Atlas A2 以上平台支持。

参数:

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (float, 可选) - 学习率。默认值: `1e-3`。
- **betas** (Tuple[float, float], 可选) - 动量矩阵的指数衰减率。默认值: `(0.9, 0.999)`。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值: `1e-8`。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值: `1e-2`。
- **amsgrad** (bool, 可选) - 是否使用 AMSGrad 算法。默认值: `False`。

关键字参数:

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值: `False`。

输入:

- **gradients** (tuple[Tensor], 可选) - 网络权重的梯度。

异常:

- **ValueError** - 学习率不是 float。
- **ValueError** - 学习率小于 0。
- **ValueError** - `eps` 小于 0。
- **ValueError** - `betas` 范围不在 `[0, 1)` 之间。
- **ValueError** - `weight_decay` 小于 0。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> from mindspore import mint
>>> from mindspore.mint import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.AdamW(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

4.6.3 mindspore.mint.optim.SGD

class mindspore.mint.optim.SGD (*params, lr, momentum=0, dampening=0, weight_decay=0, nesterov=False, *, maximize=False*)

随机梯度下降算法。

$$v_{t+1} = u * v_t + gradient * (1 - dampening)$$

如果 `nesterov` 为 `True`:

$$p_{t+1} = p_t - lr * (gradient + u * v_{t+1})$$

如果 `nesterov` 为 `False`:

$$p_{t+1} = p_t - lr * v_{t+1}$$

需要注意的是, 对于训练的第一步 $v_{t+1} = gradient$ 。其中, `p`、`v` 和 `u` 分别表示 *parameters*、*accum* 和 *momentum*。

警告: 这是一个实验性的优化器接口, 后续可能修改或删除。需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数:

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[bool, int, float, Tensor]) - 学习率。
- **momentum** (Union[bool, int, float], 可选) - 动量值。默认值: 0。
- **weight_decay** (Union[bool, int, float], 可选) - 权重衰减 (L2 penalty), 必须大于等于 0。默认值: 0.。
- **dampening** (Union[bool, int, float], 可选) - 动量的阻尼值。默认值: 0。
- **nesterov** (bool, 可选) - 启用 Nesterov 动量。如果使用 Nesterov, 动量必须为正, 阻尼必须等于 0。默认值: `False`。

关键字参数:

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值: `False`。

输入:

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常:

- **ValueError** - 学习率不是 bool、int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *momentum* 和 *weight_decay* 值小于 0。
- **ValueError** - *momentum*、*dampening* 和 *weight_decay* 不是 bool、int 或 float。
- **ValueError** - *nesterov* 和 *maximize* 不是 bool 类型。
- **ValueError** - *nesterov* 为 `True` 时, *momentum* 不为正或 *dampening* 不为 0。

支持平台:

Ascend

样例:

```

>>> import mindspore
>>> from mindspore import mint
>>> from mindspore.mint import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.SGD(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss

```

4.7 mindspore.mint.linalg

4.7.1 逆数

接口名	概述	支持平台
<code>mindspore.mint.linalg.inv</code>	计算输入矩阵的逆。	Ascend
<code>mindspore.mint.linalg.matrix_norm</code>	返回给定 Tensor 在指定维度上的矩阵范数。	Ascend
<code>mindspore.mint.linalg.norm</code>	返回给定 Tensor 的矩阵范数或向量范数。	Ascend
<code>mindspore.mint.linalg.vector_norm</code>	返回给定 Tensor 在指定维度上的向量范数。	Ascend
<code>mindspore.mint.linalg.qr</code>	对输入矩阵进行正交分解： $A = QR$ 。	Ascend

mindspore.mint.linalg.inv

`mindspore.mint.linalg.inv(input)`

计算输入矩阵的逆。

参数：

- **input** (**Tensor**) - 计算的矩阵。*input* 至少是两维的，最后两个维度大小相同，并且矩阵需要可逆。

返回：

Tensor，其类型和 *shape* 与 *input* 相同。

异常：

- **ValueError** - *input* 最后两个维度的大小不相同。
- **ValueError** - *input* 不是空并且它的维度小于 2 维。
- **ValueError** - *input* 的维度大于 6

支持平台：

Ascend

样例：


```
>>> from mindspore import Tensor, mint
>>> from mindspore import dtype as mstype
>>> x = Tensor([[1., 2.], [3., 4.]], mstype.float32)
>>> print(mint.linalg.inv(x))
[[-2.    1. ]
 [ 1.5 -0.5]]
```

mindspore.mint.linalg.matrix_norm

mindspore.mint.linalg.matrix_norm(A, ord='fro', dim=(- 2, - 1), keepdim=False, *, dtype=None)

返回给定 Tensor 在指定维度上的矩阵范数。
ord 为范数的计算模式。支持下列范数模式。

ord	矩阵范数
'fro' (默认值)	Frobenius 范数
'nuc'	Nuclear 范数
inf	max(sum(abs(x), dim = 1))
-inf	min(sum(abs(x), dim = 1))
1	max(sum(abs(x), dim = 0))
-1	min(sum(abs(x), dim = 0))
2	最大奇异值
-2	最小奇异值

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **A** (Tensor) - shape 为 $(*, m, n)$ 的 Tensor，其中 $*$ 是零个或多个 batch 维度。
- **ord** (Union[int, inf, -inf, 'fro', 'nuc'], 可选) - 范数的计算模式。行为参考上表。默认值：'fro'。
- **dim** (Tuple(int, int), 可选) - 计算矩阵范数的维度。默认值：(-2, -1)。
- **keepdim** (bool) - 输出 Tensor 是否保留原有的维度。默认值：False。

关键字参数：

- **dtype** (mindspore.dtype, 可选) - 如果设置此参数，则会在执行之前将 **A** 转换为指定的类型，返回的 Tensor 类型也将为指定类型 **dtype**。如果 **dtype** 为 None，保持 **A** 的类型不变。默认值：None。

返回：

Tensor，在指定维度 **dim** 上进行范数计算的结果。

异常：

- **TypeError** - **dim** 不是由 int 组成的 tuple。
- **ValueError** - **dim** 的长度不是 2。
- **ValueError** - **ord** 不在 [2, -2, 1, -1, float('inf'), float('-inf'), 'fro', 'nuc'] 中。
- **ValueError** - **dim** 的两个元素在标准化过后取值相同。
- **ValueError** - **dim** 的任意元素超出索引。

说明：动态 shape、动态 rank 和可变输入不支持在 图模式 (mode=mindspore.GRAPH_MODE) 下执行。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> A = ms.ops.arange(0, 12, dtype=ms.float32).reshape(3, 4)
>>> print(ms.mint.linalg.matrix_norm(A, ord='fro'))
22.494444
>>> print(ms.mint.linalg.matrix_norm(A, ord='nuc'))
24.364643
>>> print(ms.mint.linalg.matrix_norm(A, ord=float('inf')))
38.0
>>> print(ms.mint.linalg.matrix_norm(A, ord=float('-inf')))
6.0
```

mindspore.mint.linalg.norm

`mindspore.mint.linalg.norm(A, ord=None, dim=None, keepdim=False, *, dtype=None)`

返回给定 Tensor 的矩阵范数或向量范数。

`ord` 为范数的计算模式。支持下列范数模式。

<i>ord</i>	矩阵范数	向量范数
<i>None</i> (默认值)	Frobenius 范数	2-范数 (参考最下方公式)
'fro'	Frobenius 范数	不支持
'nuc'	Nuclear 范数	不支持
<i>inf</i>	$\max(\text{sum}(\text{abs}(x), \text{dim} = 1))$	$\max(\text{abs}(x))$
<i>-inf</i>	$\min(\text{sum}(\text{abs}(x), \text{dim} = 1))$	$\min(\text{abs}(x))$
0	不支持	$\text{sum}(x \neq 0)$
1	$\max(\text{sum}(\text{abs}(x), \text{dim} = 0))$	参考最下方公式
-1	$\min(\text{sum}(\text{abs}(x), \text{dim} = 0))$	参考最下方公式
2	最大奇异值	参考最下方公式
-2	最小奇异值	参考最下方公式
其余 int 或 float 值	不支持	$\text{sum}(\text{abs}(x)^{\text{ord}})^{(1/\text{ord})}$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **A** (Tensor) - shape 为 $(*, n)$ 或者 $(*, m, n)$ 的 Tensor。其中 $*$ 是零个或多个 batch 维度。
- **ord** (Union[int, float, inf, -inf, 'fro', 'nuc'], 可选) - 范数的计算模式。行为模式参考上表。默认值：None。
- **dim** (Union[int, Tuple(int)], 可选) - 计算向量范数或矩阵范数的维度。默认值：None。
 - 当 *dim* 为 int 时，会按向量范数计算。
 - 当 *dim* 为一个二元组时，会按矩阵范数计算。

- 当 *dim* 为 None 且 *ord* 为 None, *A* 将会被展平为 1D 并计算向量的 2-范数。
- 当 *dim* 为 None 且 *ord* 不为 None, *A* 必须为 1D 或者 2D。

- **keepdim** (bool) - 输出 Tensor 是否保留原有的维度。默认值: False。

关键字参数:

- **dtype** (*mindspore.dtype*, 可选) - 如果设置此参数, 则会在执行之前将 *A* 转换为指定的类型, 返回的 Tensor 类型也将为指定类型 *dtype*。默认值: None。

返回:

Tensor, 在指定维度 *dim* 上进行范数计算的结果, 与输入 *A* 的数据类型相同。

异常:

- **ValueError** - *dim* 超出范围。
- **TypeError** - *dim* 既不是 int 也不是由 int 组成的 tuple。
- **TypeError** - *A* 是一个向量并且 *ord* 是 str 类型。
- **ValueError** - *A* 是一个矩阵并且 *ord* 不是有效的取值。
- **ValueError** - *dim* 的两个元素在标准化过后取值相同。
- **ValueError** - *dim* 的任意元素超出索引。

说明: 动态 shape、动态 rank 和可变输入不支持在 图模式 (*mode=mindspore.GRAPH_MODE*) 下执行。

支持平台:

Ascend

样例:

```
>>> import mindspore as ms
>>> from mindspore import mint, ops
>>> data_range = ops.arange(-13, 13, dtype=ms.float32)
>>> x = data_range[data_range != 0]
>>> print(mint.linalg.norm(x))
38.327538
>>> print(mint.linalg.norm(x, 1))
169.0
>>> n = ops.arange(27, dtype=ms.float32).reshape(3, 3, 3)
>>> print(mint.linalg.norm(n, dim=(1, 2)))
[14.282857 39.76179 66.45299 ]
>>> print(mint.linalg.norm(n[0, :, :]))
14.282857
>>> print(mint.linalg.norm(n[1, :, :]))
39.76179
```

mindspore.mint.linalg.vector_norm

`mindspore.mint.linalg.vector_norm(x, ord=2, dim=None, keepdim=False, *, dtype=None)`

返回给定 Tensor 在指定维度上的向量范数。

ord 为范数的计算模式。支持下列范数模式。

<i>ord</i>	向量范数
2 (默认值)	2 - 范数 (参考最下方公式)
inf	$\max(abs(x))$
-inf	$\min(abs(x))$
0	$\sum(x \neq 0)$
其余 int 或 float 值	$\sum(abs(x)^{ord})^{(1/ord)}$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **x** (Tensor) - shape 为 (*) 的 Tensor。其中 * 是零个或多个 batch 维度。
- **ord** (Union[bool, int, float, inf, -inf], 可选) - 范数的计算模式。行为模式参考上表。默认值：2。
- **dim** (Union[int, List(int), Tuple(int)], 可选) - 计算向量范数的维度。默认值：None。
 - 当 *dim* 是 int、list 或者 tuple 时，会在指定的维度上计算范数，而剩余的维度会被作为 batch 维度。
 - 当 *dim* 为 None 时，在计算范数之前，会将 Tensor *x* 展平。
- **keepdim** (bool) - 输出 Tensor 是否保留原有的维度。默认值：False。

关键字参数：

- **dtype** (`mindspore.dtype`, 可选) - 如果设置此参数，则会在执行之前将 *x* 转换为指定的类型，返回的 Tensor 类型也将为指定类型 *dtype*。如果 *dtype* 为 None，保持 *x* 的类型不变。默认值：None。

返回：

Tensor，在指定维度 *dim* 上进行范数计算的结果，与输入 *x* 的数据类型相同。

异常：

- **TypeError** - *x* 不是 tensor。
- **TypeError** - *dim* 既不是 int 也不是由 int 组成的 list 或 tuple。
- **ValueError** - *ord* 不在 [bool, int, float, inf, -inf] 中。
- **ValueError** - *dim* 中的元素有重复。
- **ValueError** - *dim* 中的任意元素超出了范围。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> x = ms.ops.arange(0, 12, dtype=ms.float32) - 6
>>> print(ms.mint.linalg.vector_norm(x, ord=2))
12.083046
>>> print(ms.mint.linalg.vector_norm(x, ord=float('inf')))
6.0
>>> print(ms.mint.linalg.vector_norm(x, ord=float('-inf')))
0.0
>>> print(ms.mint.linalg.vector_norm(x, ord=0))
11.0
>>> print(ms.mint.linalg.vector_norm(x, ord=4.5))
7.2243643
```

mindspore.mint.linalg.qr

`mindspore.mint.linalg.qr(A, mode='reduced')`

对输入矩阵进行正交分解: $A = QR$ 。

其中 A 为输入 Tensor, 维度至少为 2, A 可以表示为正交矩阵 Q 与上三角矩阵 R 的乘积形式。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **A** (Tensor) - 计算的矩阵, A 至少是两维的。
- **mode** (str, 可选) - 矩阵分解的模式, 可选 `reduced`、`complete`、`r`, 默认值: `reduced`。
 - "reduced": 对于输入 $A(*, m, n)$ 输出简化大小的 $Q(*, m, k)$, $R(*, k, n)$, 其中 k 为 m, n 的最小值。
 - "complete": 对于输入 $A(*, m, n)$ 输出完整大小的 $Q(*, m, m)$, $R(*, m, n)$ 。
 - "r": 仅计算 `reduced` 场景下的 $R(*, k, n)$, 其中 k 为 m 和 n 的最小值, 返回 Q 为空 tensor。

返回:

- **Q** (Tensor) - shape 为 $Q(*, m, k)$ 或 $(*, m, n)$, 与 A 具有相同的 dtype。
- **R** (Tensor) - shape 为 $Q(*, k, n)$ 或 $(*, m, n)$, 与 A 具有相同的 dtype。

异常:

- **TypeError** - 如果 A 不是 tensor。
- **TypeError** - 如果 A 的 dtype 不是 float32。
- **ValueError** - 如果 A 不为空并且它的维度小于 2 维。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([[1.0, 1.0, 2.0, 4.0], [1.0, 1.0, 2.0, 4.0]]), mindspore.float32)
>>> output = mint.linalg.qr(x)
```

(下页继续)

(续上页)

```
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[-7.07106829e-01, -7.07106769e-01],
[-7.07106769e-01,  7.07106829e-01]]),
Tensor(shape=[2, 4], dtype=Float32, value=
[[-1.41421354e+00, -1.41421354e+00, -2.82842731e+00, -5.65685463e+00],
[ 0.00000000e+00,  3.42285418e-08,  0.00000000e+00,  0.00000000e+00]]))
```

4.8 mindspore.mint.special

4.8.1 逐元素运算

接口名	概述	支持平台
<code>mindspore.mint.special.erfc</code>	逐元素计算输入 <code>tensor</code> 的互补误差。	Ascend
<code>mindspore.mint.special.exp2</code>	逐元素计算 <code>Tensor input</code> 以 2 为底的指数。	Ascend
<code>mindspore.mint.special.expm1</code>	逐元素计算输入 <code>tensor</code> 的指数，然后减 1。	Ascend
<code>mindspore.mint.special.log1p</code>	逐元素计算 (<code>tensor + 1</code>) 的自然对数。	Ascend
<code>mindspore.mint.special.log_softmax</code>	在指定轴上对输入 <code>Tensor</code> 应用 <code>LogSoftmax</code> 函数。	Ascend
<code>mindspore.mint.special.round</code>	对输入数据进行四舍五入到最接近的整数数值。	Ascend
<code>mindspore.mint.special.sinc</code>	计算输入的归一化正弦值。	Ascend

mindspore.mint.special.erfc

`mindspore.mint.special.erfc(input)`
逐元素计算输入 `tensor` 的互补误差。

$$\text{erfc}(x) = 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

参数：

- `input` (Tensor) - 输入 `tensor`。

返回：
`Tensor`
支持平台：
Ascend
样例：

```
>>> import mindspore
>>> # The datatype of output will be float32 when datatype of input is in [int64,
↳bool](Datatype only supported on Ascend).
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.int64)
>>> mindspore.mint.special.erfc(input)
Tensor(shape=[5], dtype=Float32, value= [ 1.84270079e+00,  1.00000000e+00,  1.57299207e-01,
↳4.67773498e-03,  2.20904970e-05])
```

(下页继续)

(续上页)

```
>>>
>>> # Otherwise output has the same dtype as the input.
>>> input = mindspore.tensor([-1, 0, 1, 2, 3], mindspore.float64)
>>> mindspore.mint.special.erfc(input)
Tensor(shape=[5], dtype=Float64, value= [ 1.84270079e+00,  1.00000000e+00,  1.57299207e-01,  4.67773498e-03,  2.20904970e-05])
```

mindspore.mint.special.exp2

`mindspore.mint.special.exp2(input)`

逐元素计算 Tensor *input* 以 2 为底的指数。

$$out_i = 2^{input_i}$$

参数:

- **input** (Tensor) - 输入 Tensor。

返回:

Tensor, shape 与 *input* 相同。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> x = Tensor(np.array([0.0, 1.0, 2.0, 4.0]), mindspore.float32)
>>> output = mint.special.exp2(x)
>>> print(output)
[ 1.  2.  4. 16.]
```

mindspore.mint.special.expm1

`mindspore.mint.special.expm1(input)`

逐元素计算输入 tensor 的指数，然后减 1。

$$out_i = e^{x_i} - 1$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.0, 1.0, 3.0], mindspore.float32)
>>> output = mindspore.mint.special.expm1(input)
>>> print(output)
[ 0.          1.7182817 19.085537 ]
```

mindspore.mint.special.log1p

`mindspore.mint.special.log1p(input)`

逐元素计算 $(\text{tensor} + 1)$ 的自然对数。

$$\text{out}_i = \log_e(\text{input}_i + 1)$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> x = mindspore.tensor([1.0, 2.0, 4.0], mindspore.float32)
>>> output = mindspore.mint.special.log1p(x)
>>> print(output)
[0.6931472 1.0986123 1.609438 ]
```

mindspore.mint.special.log_softmax

`mindspore.mint.special.log_softmax(input, dim=None, *, dtype=None)`

在指定轴上对输入 Tensor 应用 LogSoftmax 函数。假设在指定轴上, x 对应每个元素 x_i , 则 LogSoftmax 函数如下所示:

$$\text{output}(x_i) = \log \left(\frac{\exp(x_i)}{\sum_{j=0}^{N-1} \exp(x_j)} \right),$$

其中, N 为 Tensor 长度。

参数:

- **input** (Tensor) - 输入 Tensor。
- **dim** (int, 可选) - 指定进行 Log softmax 运算的轴。默认值: None。

关键字参数:

- **dtype** (`mindspore.dtype`, 可选) - 输出数据类型。如果不为 None, 则输入会转化为 `dtype`。这有利于防止数值溢出。如果为 None, 则输出和输入的数据类型一致。默认值: None。

返回:

Tensor, 和输入 Tensor 的 shape 相同。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。
- **ValueError** - 如果 *dim* 超出范围 $[-\text{len}(\text{input.shape}), \text{len}(\text{input.shape})]$ 。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> logits = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> output = mint.special.log_softmax(logits, dim=-1)
>>> print(output)
[-4.4519143 -3.4519143 -2.4519143 -1.4519144 -0.4519144]
```

mindspore.mint.special.round

`mindspore.mint.special.round(input)`

对输入数据进行四舍五入到最接近的整数数值。

$$out_i \approx input_i$$

说明: Ascend 平台支持的输入数据类型包括 bfloat16 (Atlas 训练系列产品不支持)、float16、float32、float64、int32、int64。

参数:

- **input** (Tensor) - 输入 Tensor。

返回:

Tensor, shape 和数据类型与输入 *input* 相同。

异常:

- **TypeError** - *input* 不是 Tensor。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, mint
>>> input = Tensor(np.array([0.8, 1.5, 2.3, 2.5, -4.5]), mindspore.float32)
>>> output = mint.special.round(input)
>>> print(output)
[ 1.  2.  2.  2. -4.]
```

mindspore.mint.special.sinc

mindspore.mint.special.**sinc** (input)
计算输入的归一化正弦值。

$$out_i = \begin{cases} \frac{\sin(\pi input_i)}{\pi input_i} & input_i \neq 0 \\ 1 & input_i = 0 \end{cases}$$

参数:

- **input** (Tensor) - 输入 tensor。

返回:

Tensor

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([0.62, 0.28, 0.43, 0.62], mindspore.float32)
>>> output = mindspore.mint.special.sinc(input)
>>> print(output)
[0.47735003 0.8759357 0.7224278 0.47735003]
```

4.9 mindspore.mint.distributed

接口名	概述	支持平台
<code>mindspore.mint.distributed.all_gather</code>	汇聚指定的通信组中的 Tensor，并返回汇聚后的 Tensor 列表。	Ascend
<code>mindspore.mint.distributed.all_gather_into_tensor</code>	汇聚指定的通信组中的 Tensor，并返回汇聚后的 Tensor。	Ascend
<code>mindspore.mint.distributed.all_gather_object</code>	汇聚指定的通信组中的 Python 对象。	Ascend
<code>mindspore.mint.distributed.all_reduce</code>	使用指定方式对通信组内的所有设备的 Tensor 数据进行规约操作，所有设备都得到相同的结果，返回规约操作后的张量。	Ascend
<code>mindspore.mint.distributed.all_to_all</code>	根据用户输入的张量列表，将对应的张量发送到远端设备，并从其他设备接收张量，返回一个接收的张量列表。	Ascend
<code>mindspore.mint.distributed.all_to_all_single</code>	根据用户输入的切分大小，把输入 tensor 切分后，发送到其他的设备上，并从其他设备接收切分块，然后合并到一个输出 tensor 中。	Ascend
<code>mindspore.mint.distributed.barrier</code>	同步通信域内的多个进程。	Ascend
<code>mindspore.mint.distributed.batch_isend_irecv</code>	异步地发送和接收张量。	Ascend
<code>mindspore.mint.distributed.broadcast</code>	对输入数据整组广播。	Ascend

下页继续

表 32 – 续上页

<code>mindspore.mint.distributed.broadcast_object_list</code>	对输入 Python 对象进行整组广播。	Ascend
<code>mindspore.mint.distributed.destroy_process_group</code>	销毁指定通讯 group。	Ascend
<code>mindspore.mint.distributed.gather</code>	对通信组的输入张量进行聚合。	Ascend
<code>mindspore.mint.distributed.gather_object</code>	对通信组的输入 Python 对象进行聚合。	Ascend
<code>mindspore.mint.distributed.get_backend</code>	在指定通信组中获取当前分布式后端的名称。	Ascend
<code>mindspore.mint.distributed.get_global_rank</code>	由指定通信组中的设备序号获取通信集群中的全局设备序号。	Ascend
<code>mindspore.mint.distributed.get_group_rank</code>	由通信集群中的全局设备序号获取指定用户通信组中的 rank ID。	Ascend
<code>mindspore.mint.distributed.get_process_group_ranks</code>	获取指定通信组中的进程，并将通信组中的进程编号以列表方式返回。	Ascend
<code>mindspore.mint.distributed.get_rank</code>	在指定通信组中获取当前的设备序号。	Ascend
<code>mindspore.mint.distributed.get_world_size</code>	获取指定通信组实例的 rank_size。	Ascend
<code>mindspore.mint.distributed.init_process_group</code>	初始化通信服务并创建默认通讯 group (group=GlobalComm.WORLD_COMM_GROUP)。	Ascend
<code>mindspore.mint.distributed.irecv</code>	异步接收 Tensor 到指定线程。	Ascend
<code>mindspore.mint.distributed.isend</code>	异步发送张量到指定线程。	Ascend
<code>mindspore.mint.distributed.is_available</code>	分布式模块是否可用。	Ascend
<code>mindspore.mint.distributed.is_initialized</code>	默认的通信组是否初始化。	Ascend
<code>mindspore.mint.distributed.new_group</code>	创建用户自定义的通信组实例。	Ascend
<code>mindspore.mint.distributed.P2POp</code>	用于存放关于 'isend'、'irecv' 相关的信息，并用于 <code>batch_isend_irecv</code> 接口的入参。	Ascend
<code>mindspore.mint.distributed.recv</code>	同步接收 Tensor 到指定线程。	Ascend
<code>mindspore.mint.distributed.reduce</code>	规约指定通信组中的 Tensor，并将规约结果发送到目标为 <code>dst</code> 的进程（全局的进程编号）中，返回发送到目标进程的 Tensor。	Ascend
<code>mindspore.mint.distributed.reduce_scatter</code>	规约并且分发指定通信组中的 Tensor，返回分发后的 Tensor。	Ascend
<code>mindspore.mint.distributed.reduce_scatter_tensor</code>	规约并且分发指定通信组中的 Tensor，返回分发后的 Tensor。	Ascend
<code>mindspore.mint.distributed.scatter</code>	将输入 Tensor 均匀散射到通信域的卡上。	Ascend
<code>mindspore.mint.distributed.scatter_object_list</code>	将输入 Python 对象列表均匀散射到通信域的卡上。	Ascend
<code>mindspore.mint.distributed.send</code>	同步发送张量到指定线程。	Ascend

4.9.1 mindspore.mint.distributed.all_gather

`mindspore.mint.distributed.all_gather` (*tensor_list*, *tensor*, *group=None*, *async_op=False*)
 汇聚指定的通信组中的 Tensor，并返回汇聚后的 Tensor 列表。

说明：集合中所有进程的 Tensor 必须具有相同的 shape 和格式。

参数：

- **tensor_list** (list[Tensor]) - 输出汇聚的 Tensor 列表。
- **tensor** (Tensor) - 输入待汇聚操作的 Tensor。
- **group** (str, 可选) - 通信组名称。如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：False。

返回：

CommHandle。若 *async_op* 是 True，则 CommHandle 是一个异步工作句柄；若 *async_op* 是 False，则 CommHandle 将返回 None。

异常：

- **TypeError** - 输入 *tensor* 的数据类型不为 Tensor，*tensor_list* 不为 Tensor 列表，*group* 不是 str，*async_op* 不是 bool。
- **TypeError** - *tensor_list* 的大小不为通信组大小。
- **TypeError** - *tensor* 的数据类型和 shape 与 *tensor_list* 中所有元素存在不一致。
- **RuntimeError** - 如果目标设备无效，或后端无效，或分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import all_gather
>>> from mindspore import Tensor
>>>
>>> init_process_group()
>>> input_tensor = Tensor(np.ones([2, 8]).astype(np.float32))
>>> out_tensors = [Tensor(np.zeros([2, 8]).astype(np.float32)), Tensor(np.zeros([2, 8]).
↪astype(np.float32))]
>>> output = all_gather(out_tensors, input_tensor)
>>> print(out_tensors)
```

(下页继续)

(续上页)

```
[Tensor(shape=[2, 8], dtype=Float32, value=
[[ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00 ...  1.00000000e+00,  1.00000000e+00,  1.
  ↪00000000e+00],
 [ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00 ...  1.00000000e+00,  1.00000000e+00,  1.
  ↪00000000e+00]])],
Tensor(shape=[2, 8], dtype=Float32, value=
[[ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00 ...  1.00000000e+00,  1.00000000e+00,  1.
  ↪00000000e+00],
 [ 1.00000000e+00,  1.00000000e+00,  1.00000000e+00 ...  1.00000000e+00,  1.00000000e+00,  1.
  ↪00000000e+00]])]
```

4.9.2 mindspore.mint.distributed.all_gather_into_tensor

`mindspore.mint.distributed.all_gather_into_tensor` (*output_tensor*, *input_tensor*, *group=None*,
async_op=False)

汇聚指定的通信组中的 Tensor，并返回汇聚后的 Tensor。

说明：集合中所有进程的 Tensor 必须具有相同的 shape 和格式。

参数：

- **output_tensor** (Tensor) - 输出待汇聚操作的 Tensor。如果组中的 device 数量为 N，则输出 Tensor 的 shape 为 $(N * x_1, x_2, \dots, x_R)$ 。
- **input_tensor** (Tensor) - 输入待汇聚操作的 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **group** (str, 可选) - 通信组名称。如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：False。

返回：

CommHandle。若 *async_op* 是 True，则 CommHandle 是一个异步工作句柄；若 *async_op* 是 False，则 CommHandle 将返回 None。

异常：

- **TypeError** - *output_tensor* 或 *input_tensor* 输入的数据类型不为 Tensor，*group* 不是 str，*async_op* 不是 bool。
- **RuntimeError** - 如果目标设备无效，或后端无效，或分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import mint
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import all_gather_into_tensor
>>> from mindspore import Tensor
>>>
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> input_tensor = Tensor(np.ones([2, 8]).astype(np.float32))
>>> out_tensor = Tensor(np.zeros([4, 8]).astype(np.float32))
>>> output = all_gather_into_tensor(out_tensor, input_tensor)
>>> print(out_tensor)
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

4.9.3 mindspore.mint.distributed.all_gather_object

`mindspore.mint.distributed.all_gather_object` (*object_list*, *obj*, *group=None*)

汇聚指定的通信组中的 Python 对象。

说明：类似 `mindspore.mint.distributed.all_gather()` 方法，传入的参数为 Python 对象。

参数：

- **object_list** (list[Any]) - 输出汇聚的 Python 对象列表。
- **obj** (Any) - 输入待汇聚操作的 Python 对象。
- **group** (str, 可选) - 通信组名称，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。

异常：

- **TypeError** - *group* 不是 str。
- **TypeError** - *object_list* 的大小不为通信组大小。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group, get_rank
>>> from mindspore.mint.distributed import all_gather_object
>>> init_process_group()
>>> rank = get_rank()
>>> obj = ["test", {1: 2}]
>>> object_gather_list=[None, None]
>>> all_gather_object(object_gather_list, obj[rank])
>>> print(object_gather_list)
# rank_0
['test', {1: 2}]
# rank_1
['test', {1: 2}]
```

4.9.4 mindspore.mint.distributed.all_reduce

`mindspore.mint.distributed.all_reduce` (*tensor*, *op*=*ReduceOp.SUM*, *group*=*None*, *async_op*=*False*)

使用指定方式对通信组内的所有设备的 Tensor 数据进行规约操作，所有设备都得到相同的结果，返回规约操作后的张量。

说明：集合中的所有进程的 Tensor 必须具有相同的 shape 和格式。

参数：

- **tensor** (Tensor) - 输入和输出待规约操作的 Tensor，Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ ，该函数输出直接覆盖输入。
- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max"、和 "min"。默认值：ReduceOp.SUM。
- **group** (str, 可选) - 通信组名称，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：False。

返回：

CommHandle，若 *async_op* 是 True，CommHandle 是一个异步工作句柄。若 *async_op* 是 False，CommHandle 将返回 None。

异常：

- **TypeError** - 首个输入的数据类型不为 Tensor，*op* 或 *group* 不是 str，*async_op* 不是 bool 或者 *op* 值非法。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 `msrun` 启动。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import all_reduce
>>> from mindspore import Tensor
>>>
>>> init_process_group()
>>> tensor = Tensor(np.ones([2, 8]).astype(np.float32))
>>> output = all_reduce(tensor)
>>> print(tensor)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]]
```

4.9.5 mindspore.mint.distributed.all_to_all

`mindspore.mint.distributed.all_to_all(output_tensor_list, input_tensor_list, group=None, async_op=False)`

根据用户输入的张量列表，将对应的张量发送到远端设备，并从其他设备接收张量，返回一个接收的张量列表。

说明：

- 各个设备之间发送和接收的张量形状需要互相匹配。
- 仅支持 PyNative 模式，目前不支持 Graph 模式。

参数：

- **output_tensor_list** (List[Tensor]) - 包含接收张量的列表。
- **input_tensor_list** (List[Tensor]) - 包含发送到其他设备张量的列表。
- **group** (str, 可选) - 通信组名称，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：False。

返回：

CommHandle，若 `async_op` 是 True，CommHandle 是一个异步工作句柄。若 `async_op` 是 False，CommHandle 将返回 None。

异常：

- **TypeError** - `input_tensor_list` 和 `output_tensor_list` 中不全是张量类型。
- **TypeError** - `input_tensor_list` 和 `output_tensor_list` 中张量的数据类型不全部一致。
- **TypeError** - `group` 不是 str，`async_op` 不是 bool。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。


```

>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, get_rank
>>> from mindspore.mint.distributed import all_to_all
>>> from mindspore import Tensor
>>>
>>> init_process_group()
>>> this_rank = get_rank()
>>> if this_rank == 0:
...     send_tensor_list = [Tensor(1.), Tensor([[2, 3], [4, 5.]])]
...     recv_tensor_list = [Tensor(0), dtype=ms.float32), Tensor([0, 0.])]
>>> if this_rank == 1:
...     send_tensor_list = [Tensor([2, 2.]), Tensor([4, 5, 6, 7.])]
...     recv_tensor_list = [Tensor([0, 0.], [0, 0.]), Tensor([0, 0, 0, 0.])]
>>> handle = all_to_all(recv_tensor_list, send_tensor_list)
>>> print(recv_tensor_list)
rank 0:
(Tensor(shape=[], dtype=Float32, value= 1),
Tensor(shape=[2], dtype=Float32, value= [2.00000000e+00, 2.00000000e+00]))
rank 1:
(Tensor(shape=[2, 2], dtype=Float32, value=
[[2.00000000e+00, 3.00000000e+00],
[4.00000000e+00, 5.00000000e+00]]),
Tensor(shape=[4], dtype=Float32, value=[4.00000000e+00, 5.00000000e+00, 6.00000000e+00, 7.
→00000000e+00]))

```

4.9.6 mindspore.mint.distributed.all_to_all_single

`mindspore.mint.distributed.all_to_all_single` (*output, input, output_split_sizes=None, input_split_sizes=None, group=None, async_op=False*)

根据用户输入的切分大小，把输入 `tensor` 切分后，发送到其他的设备上，并从其他设备接收切分块，然后合并到一个输出 `tensor` 中。

说明：

- 仅支持 PyNative 模式，目前不支持 Graph 模式。

参数：

- **output** (Tensor) - 表示从远端收集的 `tensor` 结果。
- **input** (Tensor) - 要发送到远端设备的 `tensor`。
- **output_split_sizes** (Union(Tuple(int), List(int)), 可选) - 接收 `tensor` 在 0 维的切分大小列表。默认值：None，表示均匀切分。
- **input_split_sizes** (Union(Tuple(int), List(int)), 可选) - 发送 `tensor` 在 0 维的切分大小列表。默认值：None，表示均匀切分。
- **group** (str, 可选) - 通信组名称。如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：False。

返回：

CommHandle，若 `async_op` 是 True，则 CommHandle 是一个异步工作句柄；若 `async_op` 是 False，则 CommHandle 将返回 None。

异常：

- **TypeError** - *input* 或者 *output* 不是 *tensor* 类型, *group* 不是 *str*, *async_op* 不是 *bool*。
- **ValueError** - 当 *input_split_sizes* 为空时, *input* 的第 0 维不能被通信组内卡数整除。
- **ValueError** - 当 *output_split_sizes* 为空时, *output* 的第 0 维不能被通信组内卡数整除。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore.mint.distributed import init_process_group, get_rank
>>> from mindspore.mint.distributed import all_to_all_single
>>> from mindspore import Tensor
>>> from mindspore.ops import zeros
>>>
>>> init_process_group()
>>> this_rank = get_rank()
>>> if this_rank == 0:
...     output = Tensor(np.zeros([3, 3]).astype(np.float32))
...     tensor = Tensor([[0, 1, 2.], [3, 4, 5], [6, 7, 8]])
...     result = all_to_all_single(output, tensor, [2, 1], [2, 1])
...     print(output)
>>> if this_rank == 1:
...     output = Tensor(np.zeros([2, 3]).astype(np.float32))
...     tensor = Tensor([[9, 10., 11], [12, 13, 14]])
...     result = all_to_all_single(output, tensor, [1, 1], [1, 1])
...     print(output)
rank 0:
[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 9. 10. 11.]]
rank 1:
[[ 6.  7.  8.]
 [12. 13. 14.]]
```

4.9.7 mindspore.mint.distributed.barrier

`mindspore.mint.distributed.barrier` (*group=None, async_op=False, device_ids=None*)

同步通信域内的多个进程。进程调用到该算子后进入阻塞状态，直到通信域内所有进程调用到该算子，进程被唤醒并继续执行。

参数：

- **group** (*str*, 可选) - 通信组名称，如果为 *None*，Ascend 平台表示为 `"hccl_world_group"`。默认值：*None*。

- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值: `False`。
- **device_ids** (list[int], 可选) - 当前为预留参数。默认值: `None`。

返回:

`CommHandle`, 若 `async_op` 是 `True`, `CommHandle` 是一个异步工作句柄。若 `async_op` 是 `False`, `CommHandle` 将返回 `None`。

异常:

- **TypeError** - `group` 不是 `str`, `async_op` 不是 `bool`。
- **RuntimeError** - 如果后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.communication.comm_func import barrier
>>> # Launch 2 processes.
>>> init_process_group()
>>> barrier()
>>> print("barrier finish!")
barrier finish!
```

4.9.8 mindspore.mint.distributed.batch_isend_irecv

`mindspore.mint.distributed.batch_isend_irecv(p2p_op_list)`

异步地发送和接收张量。

说明:

- 不同设备中, `p2p_op_list` 中的 `P2POp` 的 "isend" 和 "irecv" 应该互相匹配。
 - `p2p_op_list` 中的 `P2POp` 应该使用同一个通信组。
 - 暂不支持 `p2p_op_list` 中的 `P2POp` 含有 `tag` 入参。
 - `p2p_op_list` 中的 `P2POp` 的 `tensor` 的值不会被最后的结果原地修改。
 - 仅支持 `PyNative` 模式, 目前不支持 `Graph` 模式。
-

参数:

- **p2p_op_list** (list[P2POp]) - 包含 `P2POp` 类型对象的列表。`P2POp` 指的是 `mindspore.mint.distributed.P2POp`。

返回:

list[CommHandle], 当前 list 元素为 1, `CommHandle` 是一个异步工作句柄。

异常：

- **TypeError** - `p2p_op_list` 为空，或 `p2p_op_list` 中不全是 `P2POp` 类型。
- **TypeError** - 通信组名在 `p2p_op_list` 不一致。
- **TypeError** - `tensor` 在 `p2p_op_list` 中不为 `Tensor`。
- **TypeError** - `op` 在 `p2p_op_list` 中不是 `'isend'` 或 `'irecv'`。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore.mint.distributed import init_process_group, get_rank, get_world_size
>>> from mindspore.mint.distributed import batch_isend_irecv, P2POp
>>> from mindspore import Tensor
>>>
>>> init_process_group()
>>> this_rank = get_rank()
>>> world_size = get_world_size()
>>> next_rank = (this_rank + 1) % world_size
>>> prev_rank = (this_rank + world_size - 1) % world_size
>>>
>>> send_tensor = Tensor(this_rank + 1, dtype=mindspore.float32)
>>> recv_tensor = Tensor(0., dtype=mindspore.float32)
>>>
>>> send_op = P2POp('isend', send_tensor, next_rank)
>>> recv_op = P2POp('irecv', recv_tensor, prev_rank)
>>>
>>> p2p_op_list = [send_op, recv_op]
>>> output = batch_isend_irecv(p2p_op_list)
>>> print(recv_tensor)
rank 0:
2.0
rank 1:
1.0
```

4.9.9 mindspore.mint.distributed.broadcast

`mindspore.mint.distributed.broadcast` (*tensor*, *src*, *group=None*, *async_op=False*)
对输入数据整组广播。

说明:

- 集合中的所有进程的 Tensor 的 shape 和数据格式必须相同。
- 当前支持 PyNative 模式，不支持 Graph 模式。

参数:

- **tensor** (Tensor) - 待广播出去的或接收广播的 Tensor。
- **src** (int) - 表示发送源的进程编号。只有该进程会广播张量。
- **group** (str, 可选) - 通信组名称, 如果为 None, Ascend 平台表示为 "hccl_world_group"。默认值: None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值: False。

返回:

CommHandle, 若 *async_op* 是 True, CommHandle 是一个异步工作句柄。若 *async_op* 是 False, CommHandle 将返回 None。

异常:

- **TypeError** - *tensor* 的数据类型不为 Tensor, *src* 不是 int, *group* 不是 str 或 *async_op* 不是 bool。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, broadcast
>>> import numpy as np
>>> # Launch 2 processes.
>>>
>>> init_process_group()
>>> data = ms.Tensor(np.arange(8).reshape([2, 4]).astype(np.float32))
>>> handle = broadcast(tensor=data, src=0)
>>> print(data)
[[0. 1. 2. 3.]
 [4. 5. 6. 7.]]
```

4.9.10 mindspore.mint.distributed.broadcast_object_list

`mindspore.mint.distributed.broadcast_object_list(object_list, src=0, group=None, device=None)`
对输入 Python 对象进行整组广播。

说明:

- 类似 `mindspore.mint.distributed.broadcast()` 方法, 传入的参数为 Python 对象。
- 当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **object_list** (list[Any]) - 待广播出去的或接收广播的 Python 对象。
- **src** (int, 可选) - 表示发送源的进程编号。只有该进程会广播张量。默认值: 0。
- **group** (str, 可选) - 通信组名称, 如果为 None, Ascend 平台表示为 "hccl_world_group"。默认值: None。
- **device** (str, 可选) - 当前为预留参数。默认值: None。

异常:

- **TypeError** - `src` 不是 int 或 `group` 不是 str。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group, broadcast_object_list, get_
    rank
>>> init_process_group()
>>> rank = get_rank()
>>> obj = ["test", 12, {1: 2}]
>>> if rank == 1:
...     obj = [None, None, None]
>>> broadcast_object_list(obj)
>>> print(obj)
['test', 12, {1: 2}]
```

4.9.11 mindspore.mint.distributed.destroy_process_group

`mindspore.mint.distributed.destroy_process_group(group=None)`

销毁指定通讯 group。如果指定的通讯 group 为 None 或 “hccl_world_group”，则销毁全局通讯域并释放分布式资源，例如 *hccl* 服务。

说明：

- 当前接口不支持 GPU、CPU 版本的 MindSpore 调用。
- 此方法应该在 `mindspore.mint.distributed.init_process_group()` 方法之后使用。

参数：

- **group** (str, 可选) - 通信组名称，通常由 `mindspore.mint.distributed.new_group()` 方法创建，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。

异常：

- **TypeError** - 在参数 *group* 不是字符串时抛出。
- **RuntimeError** - HCCL 服务不可用时，或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, destroy_process_group
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> destroy_process_group()
```

4.9.12 mindspore.mint.distributed.gather

`mindspore.mint.distributed.gather(tensor, gather_list, dst=0, group=None, async_op=False)`

对通信组的输入张量进行聚合。操作会将每张卡的输入 Tensor 进行聚合，发送到对应卡上。

说明：

- 只有目标为 *dst* 的进程（全局的进程编号）才会收到聚合操作后的输出。其他进程需要传入张量但没有数学意义。
- 当前仅支持集合中所有进程的 Tensor 必须具有相同的 shape 和格式。
- 当前支持 PyNative 模式，不支持 Graph 模式。

参数：

- **tensor** (Tensor) - 输入待聚合的 Tensor。
- **gather_list** (list[Tensor]) - 聚合后的 Tensor 列表。
- **dst** (int, 可选) - 表示发送源的进程编号。只有该进程会接收聚合后的张量。默认值: 0。
- **group** (str, 可选) - 通信组名称, 如果为 None, Ascend 平台表示为 "hccl_world_group"。默认值: None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值: False。

返回:

CommHandle, 若 *async_op* 是 True, CommHandle 是一个异步工作句柄。若 *async_op* 是 False, CommHandle 将返回 None。

异常:

- **TypeError** - 输入 *tensor* 的数据类型不为 Tensor, *gather_list* 不为 Tensor 列表。
- **TypeError** - *dst* 不是 int, *group* 不是 str 或 *async_op* 不是 bool。
- **TypeError** - *gather_list* 的大小不为通信组大小。
- **TypeError** - *tensor* 的数据类型和 shape 与 *gather_list* 中所有元素存在不一致。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore.mint.distributed import init_process_group, gather
>>> from mindspore import Tensor
>>> # Launch 2 processes.
>>> init_process_group()
>>> input = Tensor(np.arange(4).reshape([2, 2]).astype(np.float32))
>>> outputs = [Tensor(np.zeros([2, 2]).astype(np.float32)), Tensor(np.zeros([2, 2]).astype(np.
↪float32))]
>>> gather(input, outputs, dst=0)
>>> print(outputs)
# rank_0
[Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  1.00000000e+00],
 [ 2.00000000e+00,  3.00000000e+00]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  1.00000000e+00], [ 2.00000000e+00,  3.00000000e+00]])]
[Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  1.00000000e+00],
 [ 2.00000000e+00,  3.00000000e+00]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  1.00000000e+00], [ 2.00000000e+00,  3.00000000e+00]])]
# rank_1
```

(下页继续)

(续上页)

```
[Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  0.00000000e+00], [ 0.00000000e+00,  0.00000000e+00]])]
[Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 0.00000000e+00,  0.00000000e+00], [ 0.00000000e+00,  0.00000000e+00]])]
```

4.9.13 mindspore.mint.distributed.gather_object

`mindspore.mint.distributed.gather_object(obj, object_gather_list=None, dst=0, group=None)`

对通信组的输入 Python 对象进行聚合。

说明:

- 类似 `mindspore.mint.distributed.gather()` 方法, 传入的参数为 Python 对象。
- 当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **obj** (Any) - 输入待聚合的 Python 对象。
- **object_gather_list** (list[Any], 可选) - 聚合后的 Python 对象列表, 在 *dst* 设备上需保证输出为通信组的大小。默认值: None。
- **dst** (int, 可选) - 表示发送源的进程编号。只有该进程会接收聚合后的张量。默认值: 0。
- **group** (str, 可选) - 通信组名称, 如果为 None, Ascend 平台表示为 "hccl_world_group"。默认值: None。

异常:

- **TypeError** - *dst* 不是 int 或 *group* 不是 str。
- **TypeError** - *object_gather_list* 的大小不为通信组大小。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group, gather_object, get_rank
>>> init_process_group()
>>> rank = get_rank()
```

(下页继续)

(续上页)

```
>>> obj = ["test", {1: 2}]
>>> object_gather_list=[None, None]
>>> gather_object(obj[rank], object_gather_list)
>>> print(object_gather_list)
# rank_0
['test', {1: 2}]
```

4.9.14 mindspore.mint.distributed.get_backend

`mindspore.mint.distributed.get_backend(group=None)`

在指定通信组中获取当前分布式后端的名称。

说明：当前仅支持一种后端类型。后端类型包含 "hccl"、"nccl" 和 "mccl"，当前仅支持 "hccl" 和 "mccl"。

参数：

- **group** (str, 可选) - 通信组名称，通常由 `mindspore.mint.distributed.new_group()` 方法创建，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。

返回：

str，分布式后端的名称。

异常：

- **TypeError** - 参数 `group` 不是字符串。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, get_backend
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> backend = get_backend()
>>> print("backend is: ", backend)
backend is: hccl
```

4.9.15 mindspore.mint.distributed.get_global_rank

`mindspore.mint.distributed.get_global_rank(group, group_rank)`

由指定通信组中的设备序号获取通信集群中的全局设备序号。

说明: `get_global_rank` 方法应该在 `mindspore.mint.distributed.init_process_group()` 方法之后使用。

参数:

- **group** (str) - 通信组名称, 通常由 `mindspore.mint.distributed.new_group()` 方法创建, 如果为 `None`, Ascend 平台表示为 "hccl_world_group"。
- **group_rank** (int) - 通信组内的设备序号。

返回:

int, 通信集群中的全局设备序号。

异常:

- **TypeError** - 参数 `group` 不是字符串。
- **TypeError** - 参数 `group_rank` 不是数字。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, get_global_rank, new_group, \
    get_rank
>>> ms.set_device(device_target="Ascend")
>>> # Launch 8 processes.
>>> init_process_group()
>>> rank_ids = [0, 4]
>>> if get_rank() in rank_ids:
...     group = new_group(rank_ids)
...     world_rank_id = get_global_rank(group, 1)
...     print("world_rank_id is: ", world_rank_id)
#rank 0 and 4:
world_rank_id is: 4
```

4.9.16 mindspore.mint.distributed.get_group_rank

`mindspore.mint.distributed.get_group_rank(group, global_rank)`
由通信集群中的全局设备序号获取指定用户通信组中的 rank ID。

说明: `get_group_rank` 方法应该在 `mindspore.mint.distributed.init_process_group()` 方法之后使用。

参数:

- **group** (str) - 通信组名称, 通常由 `mindspore.mint.distributed.new_group()` 方法创建, 如果为 None, Ascend 平台表示为 "hccl_world_group"。
- **global_rank** (int) - 通信集群内的全局 rank ID。

返回:

当前通信组内的 rank_ID, 数据类型为 int。

异常:

- **TypeError** - 在参数 `global_rank` 不是数字或参数 `group` 不是字符串时抛出。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, new_group, get_group_rank, \
    get_rank
>>> ms.set_device(device_target="Ascend")
>>> # Launch 8 processes.
>>> init_process_group()
>>> rank_ids = [0, 4]
>>> if get_rank() in rank_ids:
...     group = new_group(rank_ids)
...     group_rank_id = get_group_rank(group, 4)
...     print("group_rank_id is: ", group_rank_id)
#rank 0 and 4:
group_rank_id is: 1
```

4.9.17 mindspore.mint.distributed.get_process_group_ranks

`mindspore.mint.distributed.get_process_group_ranks (group=None)`

获取指定通信组中的进程，并将通信组中的进程编号以列表方式返回。

参数：

- **group** (str, 可选) - 通信组名称，通常由`mindspore.mint.distributed.new_group()`方法创建，如果为 `None`，Ascend 平台表示为 "hccl_world_group"。默认值：None。

返回：

List (List[int]) - 指定通信组中的进程编号列表。

异常：

- **TypeError** - `group` 不是字符串。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, get_process_group_ranks
>>> # Launch 4 processes.
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> output = get_process_group_ranks()
>>> print(output)
[0, 1, 2, 3]
```

4.9.18 mindspore.mint.distributed.get_rank

`mindspore.mint.distributed.get_rank (group=None)`

在指定通信组中获取当前的设备序号。

说明：`get_rank` 方法应该在`mindspore.mint.distributed.init_process_group()`方法之后使用。

参数：

- **group** (str, 可选) - 通信组名称，通常由`mindspore.mint.distributed.new_group()`方法创建，如果为 `None`，Ascend 平台表示为 "hccl_world_group"。默认值：None。

返回：

int, 调用该方法的进程对应的组内序号。

异常：

- **TypeError** - 在参数 `group` 不是字符串时抛出。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 msrun 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, get_rank
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> rank_id = get_rank()
>>> print(rank_id)
>>> # the result is the rank_id in world_group
#rank 0: 0
#rank 1: 1
```

4.9.19 mindspore.mint.distributed.get_world_size

`mindspore.mint.distributed.get_world_size(group=None)`

获取指定通信组实例的 `rank_size`。

说明: `get_world_size` 方法应该在 `mindspore.mint.distributed.init_process_group()` 方法之后使用。

参数:

- **group** (str, 可选) - 通信组名称, 通常由 `mindspore.mint.distributed.new_group()` 方法创建, 如果为 `None`, Ascend 平台表示为 `"hccl_world_group"`。默认值: `None`。

返回:

指定通信组实例的 `rank_size`, 数据类型为 `int`。

异常:

- **TypeError** - 在参数 `group` 不是字符串时抛出。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 msrun 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, get_world_size
>>> ms.set_device(device_target="Ascend")
```

(下页继续)

(续上页)

```
>>> init_process_group()
>>> group_size = get_world_size()
>>> print("group_size is: ", group_size)
group_size is: 8
```

4.9.20 mindspore.mint.distributed.init_process_group

`mindspore.mint.distributed.init_process_group` (*backend='hccl', init_method=None, timeout=None, world_size=-1, rank=-1, store=None, pg_options=None, device_id=None*)

初始化通信服务并创建默认通讯 group (group=GlobalComm.WORLD_COMM_GROUP)。

说明:

- 当前接口不支持 GPU、CPU 版本的 MindSpore 调用。
- 在 Ascend 硬件平台下, 这个接口的设置需要在创建 Tensor 和 Parameter 之前, 以及所有算子和网络的实例化和运行之前。

参数:

- **backend** (str, 可选) - 分布式后端的名称, 默认为 "hccl", 且目前只能设置为 hccl。
- **init_method** (str, 无效参数) - 初始化通讯域时的 URL 配置。这个参数主要对标 PT, 但实际无效且设置不生效。
- **timeout** (timedelta, 无效参数) - 设置接口的超时配置。这个参数主要对标 PT, 但实际无效且设置不生效。
- **world_size** (int, 可选) - 配置初始化时全局通讯的卡数。
- **rank** (int, 无效参数) - 设置当前设备卡的卡号。这个参数主要对标 PT, 但实际无效且设置不生效。
- **store** (Store, 无效参数) - key/value 数据在设备进程上存储以便于设备进程间通讯地址、连接信息的交换。这个参数主要对标 PT, 但实际无效且设置不生效。
- **pg_options** (ProcessGroupOptions, 无效参数) - 针对创建的通讯 group 设置特殊配置策略。这个参数主要对标 PT, 但实际无效且设置不生效。
- **device_id** (int, 无效参数) - 设置当前进程使用的 NPU 卡卡号。这个参数主要对标 PT, 但实际无效且设置不生效。

异常:

- **ValueError** - 参数 *backend* 的值不是 hccl。
- **ValueError** - 参数 *world_size* 的值非-1, 且与实际通讯集群数不一致。
- **RuntimeError** - 1) 硬件设备类型无效; 2) 后台服务无效; 3) 分布式计算初始化失败; 4) 后端是 HCCL 的情况下, 未设置环境变量 RANK_ID 或 MINDSPORE_HCCL_CONFIG_PATH 的情况下初始化 HCCL 服务。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, destroy_process_group
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> destroy_process_group()
```

4.9.21 mindspore.mint.distributed.irecv

`mindspore.mint.distributed.irecv` (*tensor*, *src=0*, *group=None*, *tag=0*)

异步接收 Tensor 到指定线程。

说明: 当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **tensor** (Tensor) - 接收发送方数据存入 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **src** (int, 可选) - 表示发送源的进程编号。只会接收来自源进程的 Tensor。默认值: 0。
- **group** (str, 可选) - 通信组名称, 如果为 None, Ascend 平台表示为 "hccl_world_group"。默认值: None。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将接收来自相同 tag 的 Send 发送的 Tensor。默认值: 0。当前为预留参数。

返回:

CommHandle, CommHandle 是一个异步工作句柄。

异常:

- **TypeError** - *tensor* 不是 Tensor, *src* 不是 int 或 *group* 不是 str。
- **ValueError** - 如果该线程的 rank id 大于通信组的 rank size。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import isend, irecv, get_rank
>>> from mindspore import Tensor
>>> import numpy as np
>>>
# Launch 2 processes, Process 0 sends the array to Process 1.
```

(下页继续)

(续上页)

```
>>> init_process_group()
>>> this_rank = get_rank()
>>> if this_rank == 0:
...     input_ = Tensor(np.ones([2, 8]).astype(np.float32))
...     handle = isend(input_, 1)
...     handle.wait()
>>> if this_rank == 1:
...     x = Tensor(np.zeros([2, 8]).astype(np.float32))
...     handle = irecv(x, src=0)
...     handle.wait()
...     print(x)
rank 1:
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

4.9.22 mindspore.mint.distributed.isend

`mindspore.mint.distributed.isend(tensor, dst=0, group=None, tag=0)`

异步发送张量到指定线程。

说明：当前支持 PyNative 模式，不支持 Graph 模式。

参数：

- **tensor** (Tensor) - 输入发送 Tensor。
- **dst** (int, 可选) - 表示发送目标的进程编号。只有目标进程会收到张量。默认值：0。
- **group** (str, 可选) - 通信组名称，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将被拥有相同 *tag* 的 Receive 接收。默认值：0。当前为预留参数。

返回：

CommHandle，CommHandle 是一个异步工作句柄。

异常：

- **TypeError** - *tensor* 不是 Tensor，*dst* 不是 int 或 *group* 不是 str。
- **ValueError** - 如果发送线程和目的线程号相同。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import isend, irecv, get_rank
>>> from mindspore import Tensor
>>> import numpy as np
>>>
# Launch 2 processes, Process 0 sends the array to Process 1.
>>> init_process_group()
>>> this_rank = get_rank()
>>> if this_rank == 0:
...     input_ = Tensor(np.ones([2, 8]).astype(np.float32))
...     handle = isend(input_, 1)
...     handle.wait()
>>> if this_rank == 1:
...     x = Tensor(np.zeros([2, 8]).astype(np.float32))
...     handle = irecv(x, src=0)
...     handle.wait()
...     print(x)
rank 1:
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

4.9.23 mindspore.mint.distributed.is_available

`mindspore.mint.distributed.is_available()`
分布式模块是否可用。

说明: MindSpore 在所有平台下都支持分布式, 因此该函数返回值恒为 `True`。

返回:

`bool`, 表示分布式模块是否可用。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import is_available
>>> ms.set_device(device_target="Ascend")
>>> is_available()
True
```

4.9.24 mindspore.mint.distributed.is_initialized

`mindspore.mint.distributed.is_initialized()`

默认的通信组是否初始化。

返回:

`bool`, 表示是否初始化默认通信组。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, is_initialized
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> print(is_initialized())
True
```

4.9.25 mindspore.mint.distributed.new_group

`mindspore.mint.distributed.new_group(ranks=None, timeout=None, backend=None, pg_options=None, use_local_synchronization=False, group_desc=None)`

创建用户自定义的通信组实例。

说明: `new_group` 方法应该在 `mindspore.mint.distributed.init_process_group()` 方法之后使用。

参数:

- **ranks** (`list[int]`, 可选) - 设备编号列表。如果为 `None`, 创建全局通信组。默认值为 `None`。
- **timeout** (`int`, 无效参数) - 当前为预留参数。
- **backend** (`str`, 无效参数) - 当前为预留参数。
- **pg_options** (`str`, 无效参数) - 当前为预留参数。
- **use_local_synchronization** (`bool`, 无效参数) - 当前为预留参数。
- **group_desc** (`str`, 无效参数) - 当前为预留参数。

返回:

`str`, 生成的通信组名称, 如果执行异常则返回空。

异常:

- **TypeError** - 在参数 `ranks` 不是列表时或有重复设备号。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.mint.distributed import init_process_group, new_group
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> group = new_group()
>>> print("group is: ", group)
group is: hccl_world_group
```

4.9.26 mindspore.mint.distributed.P2POp

class mindspore.mint.distributed.P2POp(*op, tensor, peer, group=None, tag=0*)

用于存放关于‘isend’、‘irecv’相关的信息, 并用于 `batch_isend_irecv` 接口的入参。

说明: *tensor* 当 *op* 入参为‘irecv’时, 入参最后的结果会原地修改。

参数:

- **op** (Union[str, function]) - 对于字符串类型, 只允许‘isend’和‘irecv’。对于函数类型, 只允许 `distributed.isend` 和 `distributed.irecv` 函数。
- **tensor** (Tensor) - 用于发送或接收的张量。
- **peer** (int) - 发送或接收的远程设备的全局编号。
- **group** (str, 可选) - 工作的通信组, 默认值: `None` (即 Ascend 平台为 `"hccl_world_group"`)。
- **tag** (int, 可选) - 当前暂不支持。默认值: `0`。

返回:

P2POp 对象。

异常:

- **TypeError** - 当 *op* 不是与‘isend’和‘irecv’相关的字符串或函数。
- **TypeError** - 当 *tensor* 不是张量, *peer* 不是 `int`。
- **NotImplementedError** - 当 *tag* 入参不为 `0`。

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore.mint.distributed import P2POp, isend, irecv
>>> from mindspore import Tensor
>>> # Launch 2 processes.
>>> send_tensor = Tensor(1.)
>>> send_op = P2POp('isend', send_tensor, 1)
>>> send_op = P2POp(isend, send_tensor, 1)
>>> recv_tensor = Tensor(0.)
>>> recv_op = P2POp('irecv', recv_tensor, 0)
>>> recv_op = P2POp(irecv, recv_tensor, 0)
```

4.9.27 mindspore.mint.distributed.recv

`mindspore.mint.distributed.recv` (*tensor*, *src=0*, *group=None*, *tag=0*)

同步接收 Tensor 到指定线程。

说明: 当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **tensor** (Tensor) - 输入 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。输入的 *tensor* 的 shape 和 dtype 将用于接收 Tensor, 但 *tensor* 的数据值不起作用。
- **src** (int, 可选) - 表示发送源的进程编号。只会接收来自源进程的 Tensor。默认值: 0。
- **group** (str, 可选) - 通信组名称, 如果为 None, Ascend 平台表示为 "hccl_world_group"。默认值: None。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将被接收来自相同 *tag* 的 Send 发送的张量。默认值: 0。当前为预留参数。

返回:

int, 如果成功接收, 返回值为 0。

异常:

- **TypeError** - *tensor* 不是 Tensor, *src* 不是 int, 或 *group* 不是 str。
- **ValueError** - 如果该线程的 rank id 大于通信组的 rank size。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```

>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import send, recv, get_rank
>>> from mindspore import Tensor
>>> import numpy as np
>>>
# Launch 2 processes, Process 0 sends the array to Process 1.
>>> init_process_group()
>>> this_rank = get_rank()
>>> if this_rank == 0:
...     input_ = Tensor(np.ones([2, 8]).astype(np.float32))
...     send(input_, 1)
>>> if this_rank == 1:
...     x = Tensor(np.zeros([2, 8]).astype(np.float32))
...     out = recv(x, src=0)
...     print(x)
rank 1:
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]

```

4.9.28 mindspore.mint.distributed.reduce

`mindspore.mint.distributed.reduce` (*tensor*, *dst*, *op*=*ReduceOp.SUM*, *group*=*None*, *async_op*=*False*)

规约指定通信组中的 Tensor，并将规约结果发送到目标为 *dst* 的进程（全局的进程编号）中，返回发送到目标进程的 Tensor。

说明：

- 只有目标为 *dst* 的进程（全局的进程编号）才会收到规约操作后的输出。
- 当前支持 PyNative 模式，不支持 Graph 模式。
- 其他进程需传入 Tensor，该 Tensor 没有数学意义。

参数：

- **tensor** (Tensor) - 输入和输出规约的 Tensor。输出会直接修改输入。
- **dst** (int) - 指定接收输出的目标进程编号。只有该进程会接收规约操作后的输出结果。
- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max"、和 "min"。默认值：ReduceOp.SUM。
- **group** (str, 可选) - 通信组名称。如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：False。

返回：

CommHandle，若 *async_op* 是 True，则 CommHandle 是一个异步工作句柄；若 *async_op* 是 False，则 CommHandle 将返回 None。

异常：

- **TypeError** - 首个输入的数据类型不为 Tensor，*op* 和 *group* 不是 str，*async_op* 不是 bool，*op* 值非法。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore import mint
>>> import mindspore.nn as nn
>>> from mindspore.mint.distributed import init_process_group, reduce
>>> from mindspore import Tensor
>>> import numpy as np
>>> # Launch 2 processes.
>>> init_process_group()
>>> dest_rank=1
>>> input_tensor = Tensor(np.ones([2, 8]).astype(np.float32))
>>> output = reduce(input_tensor, dest_rank)
>>> print(input_tensor)
Process with rank 0: [[1. 1. 1. 1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1. 1. 1. 1.]],
Process with rank 1: [[2. 2. 2. 2. 2. 2. 2. 2.]
[2. 2. 2. 2. 2. 2. 2. 2.]],
```

4.9.29 mindspore.mint.distributed.reduce_scatter

`mindspore.mint.distributed.reduce_scatter` (*output*, *input_list*, *op*=`ReduceOp.SUM`, *group*=`None`, *async_op*=`False`)
规约并且分发指定通信组中的 Tensor, 返回分发后的 Tensor。

说明: 在集合的所有过程中, 输入列表中的 Tensor 必须具有相同的 shape 和格式。

参数:

- **output** (Tensor) - 输出规约且分发的 Tensor。
- **input_list** (list[Tensor]) - 输入待规约且分发的 Tensor 列表。
- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max"、和 "min"。默认值: `ReduceOp.SUM`。
- **group** (str, 可选) - 通信组名称, 如果为 `None`, Ascend 平台表示为 "hccl_world_group"。默认值: `None`。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值: `False`。

返回:

CommHandle, 若 *async_op* 是 `True`, CommHandle 是一个异步工作句柄。若 *async_op* 是 `False`, CommHandle 将返回 `None`。

异常:

- **TypeError** - 输入 *output* 的数据类型不为 Tensor, *input_list* 不为 Tensor 列表。
- **TypeError** - *op* 和 *group* 不是字符串, *async_op* 不是 bool, *op* 值非法。

- **TypeError** - *input_list* 的大小不为通信组大小。
- **TypeError** - *output* 的数据类型和 *shape* 与 *input_list* 中所有元素存在不一致。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore import Tensor
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import reduce_scatter
>>> import numpy as np
>>>
>>> init_process_group()
>>> input_tensors = [Tensor(np.ones([4, 8]).astype(np.float32)), Tensor(np.ones([4, 8]).
→astype(np.float32))]
>>> output_tensor = Tensor(np.zeros([4, 8]).astype(np.float32))
>>> output = reduce_scatter(output_tensor, input_tensors)
>>> print(output_tensor)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]]
```

4.9.30 mindspore.mint.distributed.reduce_scatter_tensor

`mindspore.mint.distributed.reduce_scatter_tensor` (*output, input, op=ReduceOp.SUM, group=None, async_op=False*)

规约并且分发指定通信组中的 Tensor，返回分发后的 Tensor。

说明：在集合的所有过程中，Tensor 必须具有相同的 shape 和格式。

参数：

- **output** (Tensor) - 输出分发的 Tensor，其 shape 为 $(N/rank\_size,*)$ 。
- **input** (Tensor) - 输入待规约且分发的 Tensor，假设其 shape 为 $(N,*)$ ，其中 * 为任意数量的额外维度。N 必须能够被 *rank_size* 整除，*rank_size* 为当前通讯组里面的计算卡数量。
- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max"、和 "min"。默认值：ReduceOp.SUM。
- **group** (str, 可选) - 通信组名称，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：False。

返回：

CommHandle, 若 *async_op* 是 True, CommHandle 是一个异步工作句柄。若 *async_op* 是 False, CommHandle 将返回 None。

异常:

- **TypeError** - 首个输入的数据类型不为 Tensor, *op* 和 *group* 不是字符串, *async_op* 不是 bool, *op* 值非法。
- **ValueError** - 如果输入的的第一个维度不能被 rank size 整除。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 msrun 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import reduce_scatter_tensor
>>> import numpy as np
>>>
>>> ms.set_device(device_target="Ascend")
>>> init_process_group()
>>> input_tensor = Tensor(np.ones([8, 8]).astype(np.float32))
>>> output_tensor = Tensor(np.ones([4, 8]).astype(np.float32))
>>> output = reduce_scatter_tensor(output_tensor, input_tensor)
>>> print(output_tensor)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]]
```

4.9.31 mindspore.mint.distributed.scatter

`mindspore.mint.distributed.scatter` (*tensor*, *scatter_list*, *src=0*, *group=None*, *async_op=False*)

将输入 Tensor 均匀散射到通信域的卡上。

说明:

- 该接口支持 Tensor List 输入, 只支持均匀切分。
- 只有源为 *src* 的进程 (全局的进程编号) 才会将输入 Tensor 作为散射源。
- 当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **tensor** (Tensor) - 输出散射的 Tensor。

- **scatter_list** (list[*Tensor*]) - 输入待散射的 *Tensor* 列表。
- **src** (int, 可选) - 表示发送源的进程编号。只有该进程会发送散射源 *Tensor*。默认值：0。
- **group** (str, 可选) - 通信组名称, 如果为 *None*, *Ascend* 平台表示为 "hccl_world_group"。默认值：*None*。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：*False*。

返回：

CommHandle, 若 *async_op* 是 *True*, *CommHandle* 是一个异步工作句柄。若 *async_op* 是 *False*, *CommHandle* 将返回 *None*。

异常：

- **TypeError** - 输入 *tensor* 的数据类型不为 *Tensor*, *scatter_list* 不为 *Tensor* 列表。
- **TypeError** - *op* 或 *group* 不是 str, *async_op* 不是 bool, *op* 值非法。
- **TypeError** - *scatter_list* 的大小不为通信组大小。
- **TypeError** - *tensor* 的数据类型和 *shape* 与 *scatter_list* 中所有元素存在不一致。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 *Ascend* 设备，推荐使用 *msrun* 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore import Tensor
>>> from mindspore.mint.distributed import init_process_group, scatter
>>> import numpy as np
>>> # Launch 2 processes.
>>>
>>> init_process_group()
>>> inputs = [Tensor(np.ones([2, 2]).astype(np.float32)), Tensor(np.ones([2, 2]).astype(np.
↪float32))]
>>> output = Tensor(np.zeros([2, 2]).astype(np.float32))
>>> scatter(output, inputs, src=0)
>>> print(output)
# rank_0
[[1. 1.]
 [1. 1.]]
# rank_1
[[1. 1.]
 [1. 1.]]
```

4.9.32 mindspore.mint.distributed.scatter_object_list

`mindspore.mint.distributed.scatter_object_list` (*scatter_object_output_list, scatter_object_input_list, src=0, group=None*)

将输入 Python 对象列表均匀散射到通信域的卡上。

说明:

- 类似 `mindspore.mint.distributed.scatter()` 方法, 只支持 Python 对象列表输入。
- 只有源为 `src` 的进程 (全局的进程编号) 才会将输入 Tensor 作为散射源。
- 当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **scatter_object_output_list** (list[Any]) - 输出散射的 Python 对象列表。
- **scatter_object_input_list** (list[Any]) - 输入待散射的 Python 对象列表。
- **src** (int, 可选) - 表示发送源的进程编号。只有该进程会发送散射源 Tensor。默认值: 0。
- **group** (str, 可选) - 通信组名称, 如果为 None, Ascend 平台表示为 "hccl_world_group"。默认值: None。

异常:

- **TypeError** - `src` 不是 int 或 `group` 不是 str。
- **TypeError** - `scatter_object_input_list` 的大小不为通信组大小。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group, scatter_object_list
>>> init_process_group()
>>> obj = ["test", {1: 2}]
>>> scatter_object_output_list=[None]
>>> scatter_object_list(scatter_object_output_list, obj)
>>> print(scatter_object_output_list)
# rank_0
['test']
# rank_1
[{1: 2}]
```

4.9.33 mindspore.mint.distributed.send

`mindspore.mint.distributed.send(tensor, dst=0, group=None, tag=0)`
同步发送张量到指定线程。

说明：当前支持 PyNative 模式，不支持 Graph 模式。

参数：

- **tensor** (Tensor) - 输入 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **dst** (int, 可选) - 表示发送目标的进程编号。只有目标进程会收到张量。默认值：0。
- **group** (str, 可选) - 通信组名称，如果为 None，Ascend 平台表示为 "hccl_world_group"。默认值：None。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将被拥有相同 *tag* 的 Receive 接收。默认值：0。当前为预留参数。

异常：

- **TypeError** - 输入 *tensor* 的数据类型不为 Tensor，*dst* 不是 int 或 *group* 不是 str。
- **ValueError** - 如果发送线程和目的线程号相同。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore.mint.distributed import init_process_group
>>> from mindspore.mint.distributed import send, recv, get_rank
>>> from mindspore import Tensor
>>> import numpy as np
>>>
# Launch 2 processes, Process 0 sends the array to Process 1.
>>> init_process_group()
>>> this_rank = get_rank()
>>> if this_rank == 0:
...     input_ = Tensor(np.ones([2, 8]).astype(np.float32))
...     send(input_, 1)
>>> if this_rank == 1:
...     x = Tensor(np.zeros([2, 8]).astype(np.float32))
...     out = recv(x, src=0)
...     print(x)
rank 1:
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

mindspore.common.initializer

初始化神经元参数。

class mindspore.common.initializer.**Initializer** (**kwargs)
 初始化器的抽象基类。

说明: **Initializer** 本意是用于在并行模式中延迟 **Tensor** 的数据的初始化, 而非初始化 **Tensor**。如果必须使用 **Initializer** 来初始化 **Tensor**, 大多情况下需要在初始化之后使用 `mindspore.Tensor.init_data()`。

参数:

- **kwargs** (dict) - *Initializer* 的关键字参数。

mindspore.common.initializer.initializer (init, shape=None, dtype=mstype.float32)
 创建并初始化一个 **Tensor**。

参数:

- **init** (Union[**Tensor**, str, **Initializer**, numbers.Number]) - 初始化方式。
 - **Tensor** - 用于初始化 **Tensor** 的 **Tensor**。
 - **str** - *init* 是继承自 *Initializer* 的类的别名, 实际使用时会调用相应的类。*init* 的值可以是 "normal"、"ones" 或 "zeros" 等。
 - **Initializer** - *init* 是继承自 *Initializer*, 用于初始化 **Tensor** 的类。
 - **numbers.Number** - 用于初始化 **Tensor** 的常量。
- **shape** (Union[tuple, list, int]) - 被初始化的 **Tensor** 的 **shape**, 默认值: None。
- **dtype** (mindspore.dtype) - 被初始化的 **Tensor** 的数据类型, 默认值: mstype.float32。

返回:

返回 **shape** 是入参 *shape* 的 **Tensor**。如果 *shape* 是 None, 返回 **Tensor** 的 **shape** 是 *init* 的 **shape**。

异常:

- **TypeError** - 参数 *init* 的类型不正确。
- **ValueError** - 当 *init* 传入 Tensor 对象时, *init* 的 *shape* 与形参 *shape* 内的数值不一致。

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.common.initializer import initializer, One
>>> from mindspore import Parameter
>>> data = Tensor(np.zeros([1, 2, 3]), mindspore.float32)
>>> w1 = Parameter(initializer(data, [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('ones', [1, 2, 3], mindspore.float32))
>>> w3 = Parameter(initializer(One(), [1, 2, 3], mindspore.float32))
>>> w4 = Parameter(initializer(0, [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.TruncatedNormal (*sigma=0.01, mean=0.0, a=-2.0, b=2.0*)

生成一个服从截断正态 (高斯) 分布的随机数组用于初始化 Tensor。

参数:

- **sigma** (float) - 截断正态分布的标准差, 默认值: 0.01。
- **mean** (float) - 截断正态分布的平均值, 默认值: 0.0。
- **a** (float) - 截断区间的下界, 默认值: -2.0。
- **b** (float) - 截断区间的上界, 默认值: 2.0。

样例:

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, TruncatedNormal
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(TruncatedNormal(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('truncatedNormal', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.Normal (*sigma=0.01, mean=0.0*)

生成一个服从正态分布 $N(\text{sigma}, \text{mean})$ 的随机数组用于初始化 Tensor。

$$f(x) = \frac{1}{\sqrt{2 * \text{sigma}}} \exp\left(-\frac{(x - \text{mean})^2}{2 * \text{sigma}^2}\right)$$

参数:

- **sigma** (float) - 正态分布的标准差, 默认值: 0.01。
- **mean** (float) - 正态分布的均值, 默认值: 0.0。

样例:

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Normal
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(Normal(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('normal', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.Uniform (*scale=0.07*)

生成一个服从均匀分布 $U(-\text{scale}, \text{scale})$ 的随机数组用于初始化 Tensor。

参数:

- **scale** (float) - 均匀分布的边界，默认值：0.07。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Uniform
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(Uniform(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('uniform', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.**HeUniform** (*negative_slope=0, mode='fan_in', nonlinearity='leaky_relu'*)
 生成一个服从 HeKaiming 均匀分布 $U(-\text{boundary}, \text{boundary})$ 的随机数组用于初始化 Tensor，其中：

$$\text{boundary} = \text{gain} \times \sqrt{\frac{3}{\text{fan\_mode}}}$$

gain 是一个可选的缩放因子。如果参数 *mode* 是 'fan_in'，则公式中的 *fan_mode* 是权重 Tensor 中输入单元的数量。如果参数 *mode* 是 'fan_out'，则公式中的 *fan_mode* 是权重 Tensor 中输出单元的数量。

有关 HeUniform 算法，详情可参考 <https://arxiv.org/abs/1502.01852>。

参数：

- **negative_slope** (int, float, bool) - 本层激活函数的负数区间斜率（仅适用于非线性激活函数'leaky_relu'），默认值：0。
- **mode** (str) - 可选 'fan_in' 或 'fan_out'。'fan_in' 会保留前向传递中权重方差的量级，'fan_out' 会保留反向传递的量级。默认值：'fan_in'。
- **nonlinearity** (str) - 非线性激活函数，推荐使用 'relu' 或 'leaky_relu'，默认值：'leaky_relu'。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, HeUniform
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(HeUniform(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('he_uniform', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.**HeNormal** (*negative_slope=0, mode='fan_in', nonlinearity='leaky_relu'*)
 生成一个服从 HeKaiming 正态分布 $N(0, \text{sigma}^2)$ 的随机数组用于初始化 Tensor，其中：

$$\text{sigma} = \frac{\text{gain}}{\sqrt{\text{fan\_mode}}}$$

gain 是一个可选的缩放因子。如果参数 *mode* 是 'fan_in'，则公式中的 *fan_mode* 是权重 Tensor 中输入单元的数量。如果参数 *mode* 是 'fan_out'，则公式中的 *fan_mode* 是权重 Tensor 中输出单元的数量。

HeNormal 算法的详细信息，请查看 <https://arxiv.org/abs/1502.01852>。

参数：

- **negative_slope** (int, float) - 本层激活函数的负数区间斜率（仅适用于非线性激活函数'leaky_relu'），默认值：0。
- **mode** (str) - 可选 'fan_in' 或 'fan_out'。'fan_in' 会保留前向传递中权重方差的量级，'fan_out' 会保留反向传递的量级。默认值：'fan_in'。
- **nonlinearity** (str) - 非线性激活函数，推荐使用 'relu' 或 'leaky_relu'，默认值：'leaky_relu'。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, HeNormal
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(HeNormal(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('he_normal', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.XavierNormal (gain=1)

生成一个服从 Xavier 正态分布的随机数组 $N(0, \sigma^2)$ 用于初始化 Tensor, 其中:

$$\sigma = \text{gain} * \sqrt{\frac{2}{n_{in} + n_{out}}}$$

gain 是一个可选的缩放因子。 n_{in} 为权重 Tensor 中输入单元的数量, n_{out} 为权重 Tensor 中输出单元的数量。

有关 XavierNormal 算法的详细信息, 请查看 <http://proceedings.mlr.press/v9/glorot10a.html>。

参数:

- **gain** (float) - 可选的缩放因子, 默认值: 1。

样例:

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, XavierNormal
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(XavierNormal(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('xavier_normal', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.XavierUniform (gain=1)

生成一个服从 Xavier 均匀分布 $U(-\text{boundary}, \text{boundary})$ 的随机数组用于初始化 Tensor, 均匀分布的取值范围为 $[-\text{boundary}, \text{boundary}]$, 其中:

$$\text{boundary} = \text{gain} * \sqrt{\frac{6}{n_{in} + n_{out}}}$$

gain 是一个可选的缩放因子。 n_{in} 为权重 Tensor 中输入单元的数量, n_{out} 为权重 Tensor 中输出单元的数量。

有关 XavierUniform 算法的详细信息, 请查看 <http://proceedings.mlr.press/v9/glorot10a.html>。

参数:

- **gain** (float) - 可选的缩放因子, 默认值: 1。

样例:

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, XavierUniform
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(XavierUniform(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('xavier_uniform', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.One (**kwargs)

生成一个值全为 1 的常量数组用于初始化 Tensor。

样例:

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, One
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(One(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('ones', [1, 2, 3], mindspore.float32))
```


class mindspore.common.initializer.**Zero**(**kwargs)

生成一个值全为 0 的常量数组用于初始化 Tensor。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Zero
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(Zero(), [1, 2, 3], mindspore.float32))
>>> w2 = Parameter(initializer('zeros', [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.**Constant**(value)

生成一个常量数组用于初始化 Tensor。

参数：

- **value** (Union[int, numpy.ndarray]) - 用于初始化的常数值或者数组。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Constant
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(Constant(3), [1, 2, 3], mindspore.float32))
```

class mindspore.common.initializer.**Identity**(**kwargs)

生成一个二维的单位矩阵用于初始化 Tensor。

异常：

- **ValueError** - 被初始化的 Tensor 的维度不等于 2。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Identity
>>> from mindspore import Parameter
>>> w1 = initializer(Identity(), [2, 3], mindspore.float32)
>>> w2 = initializer('identity', [2, 3], mindspore.float32)
```

class mindspore.common.initializer.**Sparse**(sparsity, sigma=0.01)

生成一个二维的稀疏矩阵用于初始化 Tensor。矩阵非 0 的位置的值服从正态分布 $N(0, \sigma)$ 。

参数：

- **sparsity** (float) - 矩阵每列中元素被置 0 的比例。
- **sigma** (float) - 正态分布的标准差，默认值：0.01。

异常：

- **ValueError** - 被初始化的 Tensor 的维度不等于 2。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Sparse
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(Sparse(sparsity=0.1, sigma=0.01), [5, 8], mindspore.float32))
```

class mindspore.common.initializer.**Dirac** (*groups=1*)

利用 Dirac delta 函数生成一个矩阵用于初始化 Tensor。Dirac 初始化方式通常用于卷积层，会尽可能多的保留输入的特性。

参数：

- **groups** (int) - 卷积层中的分组数量，每个组执行相同的初始化。默认值：1。

异常：

- **ValueError** - 被初始化的 Tensor 的维度不在 [3, 4, 5] 的范围内。
- **ValueError** - 被初始化的 Tensor 的第一个维度不能被 groups 整除。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Dirac
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(Dirac(groups=2), [6, 4, 3, 3], mindspore.float32))
>>> w2 = Parameter(initializer("dirac", [6, 4, 3, 3], mindspore.float32))
```

class mindspore.common.initializer.**Orthogonal** (*gain=1.*)

生成一个正交或半正交矩阵用于初始化 Tensor。被初始化的 Tensor 的维度至少为 2。如果维度大于 2，多余的维度将会被展平。

参数：

- **gain** (float) - 可选的比例因子，默认值为 1.0。

异常：

- **ValueError** - 被初始化的 Tensor 的维度小于 2。

样例：

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, Orthogonal
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(Orthogonal(gain=2.), [2, 3, 4], mindspore.float32))
>>> w2 = Parameter(initializer('orthogonal', [2, 3, 4], mindspore.float32))
```

class mindspore.common.initializer.**VarianceScaling** (*scale=1.0, mode='fan_in',
distribution='truncated_normal'*)

生成一个随机的矩阵用于初始化 Tensor。

当 *distribution* 是 'truncated_normal' 或者 'untruncated_normal' 时，矩阵中的值将服从均值为 0，标准差为 $stddev = \sqrt{\frac{scale}{n}}$ 的截断或者非截断正态分布。

如果 *mode* 是 'fan_in'，*n* 是输入单元的数量；如果 *mode* 是 'fan_out'，*n* 是输出单元的数量；如果 *mode* 是 'fan_avg'，*n* 是输入输出单元数量的均值。

当 *distribution* 是 'uniform' 时，矩阵中的值将服从均匀分布 $[-\sqrt{\frac{3*scale}{n}}, \sqrt{\frac{3*scale}{n}}]$ 。

参数：

- **scale** (float) - 比例因子，默认值：1.0。
- **mode** (str) - 其值应为 'fan_in'、'fan_out' 或者 'fan_avg'，默认值：'fan_in'。
- **distribution** (str) - 用于采样的分布类型。其值应为 'uniform'、'truncated_normal' 或 'untruncated_normal'，默认值：'truncated_normal'。

异常:

- **ValueError** - *scale* 小于等于 0。
- **ValueError** - *mode* 不是 'fan_in'、'fan_out' 或者 'fan_avg'。
- **ValueError** - *distribution* 不是 'truncated_normal'、'untruncated_normal' 或者 'uniform'。

样例:

```
>>> import mindspore
>>> from mindspore.common.initializer import initializer, VarianceScaling
>>> from mindspore import Parameter
>>> w1 = Parameter(initializer(VarianceScaling(scale=1.0, mode='fan_out',
...                                         distribution='untruncated_normal'), [2, 3],
mindspore.float32))
>>> w2 = Parameter(initializer('varianceScaling', [2, 3], mindspore.float32))
```


6.1 梯度缩放

<code>mindspore.amp.LossScaler</code>	使用混合精度时，用于管理损失缩放系数（loss scaler）的抽象类。
<code>mindspore.amp.DynamicLossScaler</code>	用于动态调整损失缩放系数的管理器。
<code>mindspore.amp.StaticLossScaler</code>	Static Loss scale 类。
<code>mindspore.amp.LossScaleManager</code>	使用混合精度时，用于管理损失缩放系数（loss scale）的抽象类。
<code>mindspore.amp.DynamicLossScaleManager</code>	动态调整损失缩放系数的管理器，继承自 <code>mindspore.amp.LossScaleManager</code> 。
<code>mindspore.amp.FixedLossScaleManager</code>	损失缩放系数不变的管理器，继承自 <code>mindspore.amp.LossScaleManager</code> 。

6.1.1 mindspore.amp.LossScaler

class `mindspore.amp.LossScaler`

使用混合精度时，用于管理损失缩放系数（loss scaler）的抽象类。

派生类需要实现该类的所有方法。训练过程中，`scale` 和 `unscale` 用于对损失值或梯度进行放大或缩小，以避免数据溢出；`adjust` 用于调整损失缩放系数 `scale_value` 的值。

关于使用 `LossScaler` 进行损失缩放，请查看 [教程](#)。

警告：这是一个实验性 API，后续可能修改或删除。

样例：

```

>>> from mindspore.amp import LossScaler, _grad_scale_map, _grad_unscale_map
>>> from mindspore import ops, Parameter, Tensor
>>> from mindspore.common import dtype as mstype
>>>
>>> class MyLossScaler(LossScaler):
...     def __init__(self, scale_value):
...         self.scale_value = Parameter(Tensor(scale_value, dtype=mstype.float32), name=
↪ "scale_value")
...
...     def scale(self, inputs):
...         inputs = mutable(inputs)
...         return _grad_scale_map(self.scale_value, inputs)
...
...     def unscale(self, inputs):
...         inputs = mutable(inputs)
...         return _grad_unscale_map(self.scale_value, inputs)
...
...     def adjust(self, grads_finite):
...         scale_mul_factor = self.scale_value * self.scale_factor
...         scale_value = ops.select(grads_finite, scale_mul_factor, self.scale_value)
...         ops.assign(self.scale_value, scale_value)
...         return True
>>>
>>> loss_scaler = MyLossScaler(1024)

```

abstract adjust (*grads_finite*)

根据梯度是否为有效值（无溢出）对 *scale_value* 进行调整。

参数：

- **grads_finite** (Tensor) - bool 类型的标量 Tensor，表示梯度是否为有效值（无溢出）。

abstract scale (*inputs*)

对 *inputs* 进行 scale, *inputs* *= *scale_value*。

参数：

- **inputs** (Union(Tensor, tuple(Tensor))) - 损失值或梯度。

abstract unscale (*inputs*)

对 *inputs* 进行 unscale, *inputs* /= *scale_value*。

参数：

- **inputs** (Union(Tensor, tuple(Tensor))) - 损失值或梯度。

6.1.2 mindspore.amp.DynamicLossScaler

class mindspore.amp.DynamicLossScaler (*scale_value, scale_factor, scale_window*)

用于动态调整损失缩放系数的管理器。

动态损失缩放管理器在保证梯度不溢出的情况下，尝试确定最大的损失缩放值 *scale_value*。在梯度不溢出的情况下，*scale_value* 将会每间隔 *scale_window* 步被扩大 *scale_factor* 倍；若存在溢出情况，则会将 *scale_value* 缩小 *scale_factor* 倍，并重置计数器。

警告：这是一个实验性 API，后续可能修改或删除。

参数:

- **scale_value** (Union(float, int)) - 初始梯度放大系数。
- **scale_factor** (int) - 放大/缩小倍数。
- **scale_window** (int) - 无溢出时的连续正常 step 的最大数量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import amp, Tensor
>>> import numpy as np
>>> loss_scaler = amp.DynamicLossScaler(scale_value=2**10, scale_factor=2, scale_window=1)
>>> grads = (Tensor(np.array([np.log(-1), 1.0]), mindspore.float16),
...           Tensor(np.array([0.2]), mindspore.float16))
>>> unscaled_grads = loss_scaler.unscale(grads)
>>> grads_finite = amp.all_finite(unscaled_grads)
>>> loss_scaler.adjust(grads_finite)
True
>>> print(loss_scaler.scale_value.asnumpy())
512.0
```

adjust (*grads_finite*)

根据梯度是否为有效值（无溢出）对 *scale_value* 进行调整。

参数:

- **grads_finite** (Tensor) - bool 类型的标量 Tensor，表示梯度是否为有效值（无溢出）。

教程样例:

- 自动混合精度 - 损失缩放

scale (*inputs*)

根据 *scale_value* 放大 *inputs*。

参数:

- **inputs** (Union(Tensor, tuple(Tensor))) - 损失值或梯度。

返回:

Union(Tensor, tuple(Tensor)), scale 后的值。

教程样例:

- 自动混合精度 - 损失缩放

unscale (*inputs*)

对 *inputs* 进行 unscale, *inputs* /= *scale_value*。

参数:

- **inputs** (Union(Tensor, tuple(Tensor))) - 损失值或梯度。

返回:

Union(Tensor, tuple(Tensor)), unscale 后的值。

教程样例:

- 自动混合精度 - 损失缩放

6.1.3 mindspore.amp.StaticLossScaler

class mindspore.amp.StaticLossScaler(*scale_value*)

Static Loss scale 类。用固定的常数来 scales 和 unscale 损失或梯度。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **scale_value** (Union(float, int)) - 缩放系数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import amp, Tensor
>>> import numpy as np
>>> loss_scaler = amp.StaticLossScaler(scale_value=2**10)
>>> loss_value = Tensor([1.], mindspore.float32)
>>> scaled_loss_value = loss_scaler.scale(loss_value)
>>> print(scaled_loss_value)
[1024.]
>>> grads = (Tensor(np.array([1.5, 1.0]), mindspore.float16),
...            Tensor(np.array([1.2]), mindspore.float16))
>>> unscaled_grads = loss_scaler.unscale(grads)
>>> print(unscaled_grads)
(Tensor(shape=[2], dtype=Float16, value= [ 1.4648e-03,  9.7656e-04]),
 Tensor(shape=[1], dtype=Float16, value= [ 1.1721e-03]))
```

adjust (*grads_finite*)

用于调整 LossScaler 中 *loss_value* 的值。StaticLossScaler 中，*scale_value* 值固定，因此，此方法直接返回 False。

参数：

- **grads_finite** (Tensor) - bool 类型的标量 Tensor，表示梯度是否为有效值（无溢出）。

scale (*inputs*)

对 *inputs* 进行 scale，*inputs* *= *scale_value*。

参数：

- **inputs** (Union(Tensor, tuple(Tensor))) - 损失值或梯度。

返回：

Union(Tensor, tuple(Tensor))，scale 后的值。

unscale (*inputs*)

对 *inputs* 进行 unscale，*inputs* /= *scale_value*。

参数：

- **inputs** (Union(Tensor, tuple(Tensor))) - 损失值或梯度。

返回：

Union(Tensor, tuple(Tensor)), unscale 后的值。

6.1.4 mindspore.amp.LossScaleManager

class mindspore.amp.LossScaleManager

使用混合精度时，用于管理损失缩放系数（loss scale）的抽象类。

派生类需要实现该类的所有方法。`get_loss_scale` 用于获取当前的梯度放大系数。`update_loss_scale` 用于更新梯度放大系数，该方法将在训练过程中被调用。`get_update_cell` 用于获取更新梯度放大系数的 `mindspore.nn.Cell` 实例，该实例将在训练过程中被调用。当前通常使用 `get_update_cell` 方式。

例如：`mindspore.amp.FixedLossScaleManager` 和 `mindspore.amp.DynamicLossScaleManager`。

get_loss_scale()

获取梯度放大系数（loss scale）的值。

get_update_cell()

获取用于更新梯度放大系数的 `mindspore.nn.Cell` 实例。

update_loss_scale(overflow)

根据 `overflow` 状态更新梯度放大系数（loss scale）。

参数：

- **overflow** (bool) - 表示训练过程是否溢出。

6.1.5 mindspore.amp.DynamicLossScaleManager

class mindspore.amp.DynamicLossScaleManager (`init_loss_scale=2 ** 24, scale_factor=2, scale_window=2000`)

动态调整损失缩放系数的管理器，继承自 `mindspore.amp.LossScaleManager`。

参数：

- **init_loss_scale** (float, 可选) - 初始梯度放大系数。默认值：2 ** 24。
- **scale_factor** (int, 可选) - 放大/缩小倍数。默认值：2。
- **scale_window** (int, 可选) - 无溢出时的连续正常 step 的最大数量。默认值：2000。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import amp, nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_scale_manager = amp.DynamicLossScaleManager()
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = ms.Model(net, loss_scale_manager=loss_scale_manager, optimizer=optim)
```

get_drop_overflow_update()

该值表示是否在发生溢出时放弃本轮参数更新。

返回：

bool, 值始终为 True。

get_loss_scale()

返回当前梯度放大系数。

返回:

float, 梯度放大系数。

get_update_cell()

返回用于更新梯度放大系数的 `mindspore.nn.Cell` 实例, `mindspore.nn.TrainOneStepWithLossScaleCell` 会调用该实例。

返回:

`mindspore.nn.DynamicLossScaleUpdateCell` 实例, 用于更新梯度放大系数。

update_loss_scale(overflow)

根据溢出状态更新梯度放大系数。如果发生溢出, 减小梯度放大系数, 否则增大梯度放大系数。

参数:

- **overflow** (bool) - 表示是否溢出。

6.1.6 mindspore.amp.FixedLossScaleManager

class mindspore.amp.FixedLossScaleManager (*loss_scale=128.0, drop_overflow_update=True*)

损失缩放系数不变的管理器, 继承自 `mindspore.amp.LossScaleManager`。

参数:

- **loss_scale** (float, 可选) - 梯度放大系数。注: 如果将 `drop_overflow_update` 设为 False, 则定义优化器时需要将优化器的 `loss_scale` 设为相同的值。默认值: 128.0。
- **drop_overflow_update** (bool, 可选) - 出现溢出时, 是否执行优化器。如果值为 True, 则出现溢出时不会执行优化器。默认值: True。

样例:

```
>>> import mindspore as ms
>>> from mindspore import amp, nn
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_scale = 1024.0
>>> loss_scale_manager = amp.FixedLossScaleManager(loss_scale, False)
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9, loss_
↪ scale=loss_scale)
>>> model = ms.Model(net, loss_scale_manager=loss_scale_manager, optimizer=optim)
```

get_drop_overflow_update()

返回 `drop_overflow_update`, 该值表示是否在发生溢出时放弃本轮参数更新。

返回:

bool, `drop_overflow_update` 的值。

get_loss_scale()

获取 loss scale 值。

返回:

bool, *loss_scale* 的值。

get_update_cell()

返回用于更新 *loss_scale* 值的 *mindspore.nn.Cell* 实例, *mindspore.nn.TrainOneStepWithLossScaleCell* 会调用该实例。该类使用固定的梯度放大系数, 因此该实例不执行任何操作。

返回:

None 或 *Cell*。当 *drop_overflow_update* 为 True 时, 返回 *mindspore.nn.FixedLossScaleUpdateCell* 实例, 当 *drop_overflow_update* 为 False 时, 返回 None。

update_loss_scale(overflow)

更新 *loss scale* 值。类 *mindspore.amp.FixedLossScaleManager* 中, 该方法不执行任何操作。

参数:

- **overflow** (bool) - 表示是否溢出。

6.2 数据类型自动转换

<i>mindspore.amp.auto_mixed_precision</i>	返回一个经过自动混合精度处理的网络。
<i>mindspore.amp.build_train_network</i>	构建混合精度训练网络。
<i>mindspore.amp.custom_mixed_precision</i>	通过配置白名单或黑名单, 对 <i>Cell</i> 进行自定义混合精度处理。
<i>mindspore.amp.get_black_list</i>	提供用于自动混合精度 <i>amp_level</i> 为 o2 等级的内置黑名单的拷贝。
<i>mindspore.amp.get_white_list</i>	提供用于自动混合精度 <i>amp_level</i> 为 o1 等级的内置白名单的拷贝。

6.2.1 mindspore.amp.auto_mixed_precision

mindspore.amp.auto_mixed_precision (*network*, *amp_level*='O0', *dtype*=*mstype.float16*)

返回一个经过自动混合精度处理的网络。

该接口会对输入网络进行自动混合精度处理, 处理后的网络里的 *Cell* 和算子增加了精度转换操作, 以低精度进行计算, 如 *mstype.float16* 或 *mstype.bfloat16*。*Cell* 和算子的输入和参数被转换成低精度浮点数, 计算结果被转换回全精度浮点数, 即 *mstype.float32*。

amp_level 及其对应名单决定了哪些 *Cell* 和算子需要进行精度转换。

当 *amp_level* 配置为 o0 时, 不对 *Cell* 和算子进行精度转换。

当 *amp_level* 配置为 o1 时, 白名单内的 *Cell* 和算子会被转换为低精度运算。白名单的具体内容可参考 *mindspore.amp.get_white_list()*。

当 *amp_level* 配置为 o2 时, 黑名单内的 *Cell* 保持全精度运算, 名单外的 *Cell* 会被转换为低精度运算。黑名单的具体内容可参考 *mindspore.amp.get_black_list()*。

当 *amp_level* 配置为 o3 时, 所有 *Cell* 和算子都转换为低精度运算。

当 *amp_level* 配置为 auto 时, *auto_whitelist* 名单里的算子会被转换为低精度运算, *auto_blacklist* 名单里的算子会被转换为全精度运算, *promote_list* 名单里的算子会被转换为算子输入中最高精度的浮点类型, 名单外的算子使用输入的类型进行计算。

auto_whitelist 名单里的算子包括:

Conv2D、Conv3D、Conv2DTranspose、Conv3DTranspose、Convolution、MatMul、MatMulExt、BatchMatMul、BatchMatMulExt、PReLU、Einsum、Dense、Addmm

auto_blacklist 名单里的算子包括：

Pow、ACos、Asin、Cosh、Erfinv、Exp、Expml、Log、Log1p、Reciprocal、Rsqrt、Sinh、Tan、Softplus、SoftplusExt、LayerNorm、LayerNormExt、BatchNorm、BatchNormExt、GroupNorm、KLDivLoss、SmoothL1Loss、MultilabelMarginLoss、SoftMarginLoss、TripletMarginLoss、MultiMarginLoss、BCEWithLogitsLoss、Pdist、Cdist、Renorm、ReduceProd、Softmax、LogSoftmax、CumProd、CumSum、CumsumExt、ProdExt、SumExt、Norm、MSELossExt

promote_list 名单里的算子包括：

Addcdiv、Addcmul、Cross、_PyboostCrossPrim、Dot、GridSampler2D、GridSampler3D、BiasAdd、AddN、Concat

关于自动混合精度的详细介绍，请参考 [自动混合精度](#)。

说明：

- 重复调用混合精度接口，如 *custom_mixed_precision* 和 *auto_mixed_precision*，可能导致网络层数增大，性能降低。
- 如果使用 *mindspore.train.Model* 和 *mindspore.amp.build_train_network()* 等接口来训练经过 *custom_mixed_precision* 和 *auto_mixed_precision* 等混合精度接口转换后的网络，则需要将 *amp_level* 配置为 00 以避免重复的精度转换。
- 当 *amp_level* 配置为 auto 时，网络输出的类型可能是低精度类型，此时可能需要手动转换类型以避免 loss 函数出现类型不一致的报错。
- 当 *amp_level* 配置为 auto，而网络里的 Cell 配置了 *to_float* 时，*to_float* 指定的精度优先生效。

警告：auto 等级的 *amp_level* 是实验性 API，后续可能修改或删除。

参数：

- **network** (Union[Cell, function]) - 定义网络结构。仅当 *amp_level* 配置为 auto 时支持 Function 类型。
- **amp_level** (str) - 支持 ["00", "01", "02", "03", "auto"]。默认值："00"。
 - "00" - 不变化。
 - "01" - 仅将白名单内的 Cell 和算子转换为低精度运算，其余部分保持全精度运算。
 - "02" - 黑名单内的 Cell 和算子保持全精度运算，其余部分都转换为低精度运算。
 - "03" - 将网络全部转为低精度运算。
 - "auto" - 将 *auto_whitelist* 名单内的算子转换为低精度运算，*auto_blacklist* 名单内的算子转换为全精度运算，*promote_list* 名单内的算子转换为算子输入中最高精度的浮点类型，名单外的算子使用输入的类型进行计算。
- **dtype** (Type) - 低精度计算时使用的数据类型，可以是 *mstype.float16* 或 *mstype.bfloat16*。默认值：*mstype.float16*。

异常：

- **TypeError** - *network* 不是 Cell 或函数。
- **ValueError** - *amp_level* 不在支持范围内。
- **ValueError** - *dtype* 既不是 *mstype.float16* 也不是 *mstype.bfloat16*。

样例:

```
>>> from mindspore import amp
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> network = LeNet5()
>>> amp_level = "O1"
>>> net = amp.auto_mixed_precision(network, amp_level)
```

6.2.2 mindspore.amp.build_train_network

`mindspore.amp.build_train_network(network, optimizer, loss_fn=None, level='O0', boost_level='O0', **kwargs)`

构建混合精度训练网络。

说明:

- 使用 `mindspore.amp.custom_mixed_precision()` 或 `mindspore.amp.auto_mixed_precision()` 进行精度转换后, 不支持再次使用其他接口进行精度转换。如果使用 `build_train_network` 来训练转换后的网络, 则需要将 `level` 配置为 `O0` 以避免重复的精度转换。

参数:

- network** (Cell) - 定义网络结构。
- optimizer** (`mindspore.nn.Optimizer`) - 定义优化器, 用于更新权重参数。
- loss_fn** (Union[None, Cell]) - 定义损失函数。如果为 `None`, `network` 中应该包含损失函数。默认值: `None`。
- level** (str) - 支持 ['O0', 'O1', 'O2', 'O3', 'auto']。默认值: 'O0'。
`level` 的详细配置信息可参考 `mindspore.amp.auto_mixed_precision()`。
 由 `level` 配置的 `keep_batchnorm_fp32`, `cast_model_type` 和 `loss_scale_manager` 可能会被 `kwargs` 里的配置覆盖。
- boost_level** (str) - `mindspore.boost` 中参数 `level` 的选项, 设置 `boost` 的训练模式级别。支持 ['O0', 'O1', 'O2']。默认值: 'O0'。
 - 'O0' - 不变化。
 - 'O1' - 开启 `boost` 模式, 性能提升 20% 左右, 准确率与原始准确率相同。
 - 'O2' - 开启 `boost` 模式, 性能提升 30% 左右, 准确率降低小于 3%。如果设置了 'O1' 或 'O2' 模式, `boost` 相关库将自动生效。
- cast_model_type** (`mindspore.dtype`) - 支持 `float16`, `float32`。如果设置了该参数, 网络将被转化为设置的数据类型, 而不会根据设置的 `level` 进行转换。
- keep_batchnorm_fp32** (bool) - 当网络被设置为 `float16` 时, 配置为 `True`, 则 `BatchNorm` 将保持在 `float32` 下运行。设置 `level` 不会影响该属性。
- loss_scale_manager** (Union[None, `LossScaleManager`]) - 如果不为 `None`, 必须是 `mindspore.amp.LossScaleManager` 的子类, 用于缩放损失系数 (`loss scale`)。设置 `level` 不会影响该属性。

异常:

- ValueError** - 在 CPU 上, 属性 `loss_scale_manager` 不是 `None` 或 `mindspore.amp.FixedLossScaleManager` (其属性 `drop_overflow_update=False`)。

样例:

```
>>> from mindspore import amp, nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> network = LeNet5()
>>> net_loss = nn.SoftmaxCrossEntropyWithLogits(reduction="mean")
>>> net_opt = nn.Momentum(network.trainable_params(), learning_rate=0.01, momentum=0.9)
>>> amp_level="O3"
>>> net = amp.build_train_network(network, net_opt, net_loss, amp_level)
```

6.2.3 mindspore.amp.custom_mixed_precision

`mindspore.amp.custom_mixed_precision(network, *, white_list=None, black_list=None, dtype=mstype.float16)`

通过配置白名单或黑名单，对 Cell 进行自定义混合精度处理。当提供 `white_list` 时，网络中包含在 `white_list` 里的 Primitive 或 Cell 会进行精度转换。当提供 `black_list` 时，网络中不包含在 `black_list` 里的 Cell 会进行精度转换。需要提供 `white_list` 和 `black_list` 中的一个。

说明：

- 重复调用混合精度接口，如 `custom_mixed_precision` 和 `auto_mixed_precision`，可能导致网络层数增大，性能降低。
- 如果使用 `mindspore.train.Model` 和 `mindspore.amp.build_train_network()` 等接口来训练经过 `custom_mixed_precision` 和 `auto_mixed_precision` 等混合精度接口转换后的网络，则需要将 `mindspore.train.Model` 和 `mindspore.amp.build_train_network()` 等接口的 `amp_level` 或 `level` 配置为 00 以避免重复的精度转换。
- 当使用黑名单时，Primitive 类型还未支持。

参数：

- **network** (Cell) - 定义网络结构。

关键字参数：

- **white_list** (list[Primitive, Cell], 可选) - 自定义混合精度的白名单。默认值：None。
- **black_list** (list[Cell], 可选) - 自定义混合精度的黑名单。默认值：None。
- **dtype** (Type) - 低精度计算时使用的数据类型，可以是 `mstype.float16` 或 `mstype.bfloat16`。默认值：`mstype.float16`。

返回：

`network` (Cell)，支持混合精度的网络。

异常：

- **TypeError** - `network` 的类型不是 Cell。
- **ValueError** - `white_list` 和 `black_list` 都没提供。
- **ValueError** - 同时提供了 `white_list` 和 `black_list`。
- **ValueError** - `dtype` 既不是 `mstype.float16` 也不是 `mstype.bfloat16`。

样例：

```
>>> from mindspore import amp, nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> custom_white_list = amp.get_white_list()
>>> custom_white_list.append(nn.Flatten)
>>> net = amp.custom_mixed_precision(net, white_list=custom_white_list)
```

6.2.4 mindspore.amp.get_black_list

`mindspore.amp.get_black_list()`

提供用于自动混合精度 `amp_level` 为 o2 等级的内置黑名单的拷贝。

当前的内置黑名单内容为：

```
[mindspore.nn.BatchNorm1d,      mindspore.nn.BatchNorm2d,      mindspore.nn.BatchNorm3d,
mindspore.nn.LayerNorm]
```

返回：

`list`：内置黑名单的拷贝。

样例：

```
>>> from mindspore import amp
>>> black_list = amp.get_black_list()
>>> print(black_list)
[<class 'mindspore.nn.layer.normalization.BatchNorm1d'>, <class 'mindspore.nn.layer.
↪normalization.BatchNorm2d'>,
<class 'mindspore.nn.layer.normalization.BatchNorm3d'>, <class 'mindspore.nn.layer.
↪normalization.LayerNorm'>]
```

6.2.5 mindspore.amp.get_white_list

`mindspore.amp.get_white_list()`

提供用于自动混合精度 `amp_level` 为 o1 等级的内置白名单的拷贝。

当前的内置白名单内容为：

```
[mindspore.nn.Conv1d,      mindspore.nn.Conv2d,      mindspore.nn.Conv3d,      mindspore.nn.
Conv1dTranspose, mindspore.nn.Conv2dTranspose, mindspore.nn.Conv3dTranspose, mindspore.
nn.Dense, mindspore.nn.LSTMCell, mindspore.nn.RNNCell, mindspore.nn.GRUCell, mindspore.
ops.Conv2D,      mindspore.ops.Conv3D,      mindspore.ops.Conv2DTranspose,      mindspore.ops.
Conv3DTranspose, mindspore.ops.MatMul, mindspore.ops.BatchMatMul, mindspore.ops.PReLU,
mindspore.ops.ReLU, mindspore.ops.Ger]
```

返回：

`list`：内置白名单的拷贝。

样例：

```
>>> from mindspore import amp
>>> white_list = amp.get_white_list()
>>> print(white_list)
[<class 'mindspore.nn.layer.conv.Conv1d'>, <class 'mindspore.nn.layer.conv.Conv2d'>,
```

(下页继续)

(续上页)

```
<class 'mindspore.nn.layer.conv.Conv3d'>, <class 'mindspore.nn.layer.conv.Conv1dTranspose'>,
<class 'mindspore.nn.layer.conv.Conv2dTranspose'>, <class 'mindspore.nn.layer.conv.
↪Conv3dTranspose'>,
<class 'mindspore.nn.layer.basic.Dense'>, <class 'mindspore.nn.layer.rnn_cells.LSTMCell'>,
<class 'mindspore.nn.layer.rnn_cells.RNNCell'>, <class 'mindspore.nn.layer.rnn_cells.GRUCell
↪'>,
<class 'mindspore.ops.operations.nn_ops.Conv2D'>, <class 'mindspore.ops.operations.nn_ops.
↪Conv3D'>,
<class 'mindspore.ops.operations.nn_ops.Conv2DTranspose'>,
<class 'mindspore.ops.operations.nn_ops.Conv3DTranspose'>,
<class 'mindspore.ops.operations.nn_ops.Conv2DBackpropInput'>,
<class 'mindspore.ops.operations.math_ops.MatMul'>, <class 'mindspore.ops.operations.math_
↪ops.BatchMatMul'>,
<class 'mindspore.ops.operations.nn_ops.PReLU'>, <class 'mindspore.ops.operations.nn_ops.
↪ReLU'>,
<class 'mindspore.ops.operations.math_ops.Ger'>]
```

6.3 溢出检测

`mindspore.amp.all_finite`

检查输入是否有溢出，返回为 `True` 为溢出，`False` 为无溢出。

6.3.1 mindspore.amp.all_finite

`mindspore.amp.all_finite(inputs)`

检查输入是否有溢出，返回为 `True` 为溢出，`False` 为无溢出。

警告： 这是一个实验性 API，后续可能修改或删除。

此接口只在整网训练情况下用于判断梯度是否溢出，返回结果在不同后端可能存在差异。

参数：

- **inputs** (Union(tuple(Tensor), list(Tensor))) - 可迭代的 Tensor。

返回：

Tensor，布尔类型的标量 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import amp, Tensor
>>> import numpy as np
>>> x = (Tensor(np.array([np.log(-1), 1, np.log(0)])), Tensor(np.array([1.0])))
>>> output = amp.all_finite(x)
```


教程样例：

- 自动混合精度 - 损失缩放

7.1 模型

`mindspore.train.Model`模型训练或推理的高阶接口。

7.1.1 mindspore.train.Model

```
class mindspore.train.Model (network, loss_fn=None, optimizer=None, metrics=None, eval_network=None,  
                             eval_indexes=None, amp_level='OO', boost_level='OO', **kwargs)
```

模型训练或推理的高阶接口。*Model* 会根据用户传入的参数，封装可训练或推理的实例。

说明：

- 如果使用混合精度功能，需要同时设置 *optimizer* 参数，否则混合精度功能不生效。当使用混合精度时，优化器中的 *global_step* 可能与模型中的 *cur_step_num* 不同。
 - 使用 *custom_mixed_precision* 或 *auto_mixed_precision* 进行精度转换后，不支持再次使用其他接口进行精度转换。如果使用 *Model* 来训练转换后的网络，则需要将 *amp_level* 配置为 *oo* 以避免重复的精度转换。
-

参数：

- **network** (Cell) - 用于训练或推理的神经网络。
- **loss_fn** (Cell) - 损失函数。如果 *loss_fn* 为 *None*，那么 *network* 中需要进行损失函数计算。默认值： *None* 。
- **optimizer** (Cell) - 用于更新网络权重的优化器。如果 *optimizer* 为 *None*，那么 *network* 的网络结构里需要包括反向传播和权重更新逻辑。默认值： *None* 。
- **metrics** (Union[dict, set]) - 用于模型评估的一组评价函数。例如： {'accuracy', 'recall'}。默认值： *None* 。

- **eval_network** (Cell) - 用于评估的神经网络。未定义情况下, *Model* 会使用 *network* 和 *loss_fn* 封装一个 *eval_network*。默认值: `None`。
- **eval_indexes** (list) - 在定义 *eval_network* 的情况下使用。如果 *eval_indexes* 为默认值 `None`, *Model* 会将 *eval_network* 的所有输出传给 *metrics*。如果配置 *eval_indexes*, 必须包含三个元素, 分别为损失值、预测值和标签在 *eval_network* 输出中的位置, 此时, 损失值将传给损失评价函数, 预测值和标签将传给其他评价函数。推荐使用评价函数 `mindspore.train.Metric.set_indexes()` 代替 *eval_indexes*。默认值: `None`。
- **amp_level** (str) - `mindspore.amp.build_train_network()` 的可选参数 *level*, *level* 为混合精度等级, 该参数支持 ["O0", "O1", "O2", "O3", "auto"]。默认值: "O0"。

amp_level 的详细配置信息可参考 `mindspore.amp.auto_mixed_precision()`。

通过 *kwargs* 设置 *keep_batchnorm_fp32*, 可修改 BatchNorm 的精度策略, *keep_batchnorm_fp32* 必须为 bool 类型; 通过 *kwargs* 设置 *loss_scale_manager*, 可修改损失缩放策略, *loss_scale_manager* 必须为 `mindspore.amp.LossScaleManager` 的子类。

- **boost_level** (str) - `mindspore.boost` 的可选参数, 为 boost 模式训练等级。支持 ["O0", "O1", "O2"]。默认值: "O0"。
 - "O0": 不变化。
 - "O1": 启用 boost 模式, 性能将提升约 20%, 准确率保持不变。
 - "O2": 启用 boost 模式, 性能将提升约 30%, 准确率下降小于 3%。

如果想自行配置 boost 模式, 可以将 *boost_config_dict* 设置为 *boost.py*。

为使 *boost_level* 功能生效, 必须配置优化器 *optimizer*, 同时至少需要设置评估网络 *eval_network* 或评估指标 *metric* 中的一项 (也可两项都配置)。

注意: 当前默认开启的优化仅适用部分网络, 并非所有网络都能获得相同收益。建议在 Ascend 平台的图模式下开启 boost 模式, 同时为了获取更好的加速效果, 请参考 `mindspore.boost.AutoBoost` 配置 *boost_config_dict*。

样例:

```
>>> from mindspore import nn
>>> from mindspore.train import Model
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim, metrics=None)
>>> model.train_network
>>> model.predict_network
>>> model.eval_network
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> model.train(2, dataset)
```

build (*train_dataset=None, valid_dataset=None, sink_size=-1, epoch=1, sink_mode=True*)

编译构建计算图和数据图。

警告: 这是一个实验性 API, 后续可能修改或删除。

说明：如果预先调用该接口构建计算图，那么 `Model.train` 会直接执行计算图。预构建计算图目前仅支持 GRAPH_MODE 模式和 Ascend 处理器。

参数：

- **train_dataset** (Dataset) - 一个训练集迭代器。如果定义了 `train_dataset`，将会构建训练计算图。默认值：None。
- **valid_dataset** (Dataset) - 一个验证集迭代器。如果定义了 `valid_dataset`，将会构建验证计算图，此时 `Model` 中的 `metrics` 不能为 None。默认值：None。
- **sink_size** (int) - 控制每次数据下沉的 `step` 数量。默认值：-1。
- **epoch** (int) - 控制训练轮次。默认值：1。
- **sink_mode** (bool) - 是否开启数据下沉模式。默认值：True。

样例：

```
>>> from mindspore import nn
>>> from mindspore.train import Model
>>> from mindspore.amp import FixedLossScaleManager
>>>
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> loss_scale_manager = FixedLossScaleManager()
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim, metrics=None,
...               loss_scale_manager=loss_scale_manager)
>>> model.build(dataset, epoch=2)
>>> model.train(2, dataset)
```

eval (`valid_dataset`, `callbacks=None`, `dataset_sink_mode=False`)

模型评估接口。

使用 PyNative 模式或 CPU 处理器时，模型评估流程将以非下沉模式执行。

说明：如果 `dataset_sink_mode` 配置为 `True`，数据将被发送到处理器中。此时数据集与模型绑定，数据集仅能在当前模型中使用。如果处理器是 Ascend，数据特征将被逐一传输。每次数据传输的上限是 256M。该接口会构建并执行计算图。如果使用前先执行了 `Model.build`，那么它会直接执行计算图而不构建。

参数：

- **valid_dataset** (Dataset) - 评估模型的数据集。
- **callbacks** (Optional[list(Callback), Callback]) - 评估过程中需要执行的回调对象或回调对象列表。默认值：None。
- **dataset_sink_mode** (bool) - 数据是否直接下沉至处理器进行处理。默认值：False。

返回：

Dict, key 是用户定义的评价指标名称, value 是以推理模式运行的评估结果。

样例:

```
>>> from mindspore import nn
>>> from mindspore.train import Model
>>>
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> model = Model(net, loss_fn=loss, optimizer=None, metrics={'acc'})
>>> acc = model.eval(dataset, dataset_sink_mode=False)
```

property eval_network

获取该模型的评价网络。

返回:

评估网络实例。

fit (epoch, train_dataset, valid_dataset=None, valid_frequency=1, callbacks=None, dataset_sink_mode=False, valid_dataset_sink_mode=False, sink_size=-1, initial_epoch=0)

模型边训练边推理接口。

如果 *valid_dataset* 不为 None, 在训练过程中同时执行推理。

更多详细信息请参考 `mindspore.train.Model.train()` 和 `mindspore.train.Model.eval()`。

参数:

- **epoch** (int) - 训练执行轮次。通常每个 epoch 都会使用全量数据集进行训练。当 *dataset_sink_mode* 设置为 True 且 *sink_size* 大于零时, 则每个 epoch 训练次数为 *sink_size* 而不是数据集的总步数。如果 *epoch* 与 *initial_epoch* 一起使用, 它表示断点续训多少个 epoch。
- **train_dataset** (Dataset) - 训练数据集迭代器。如果定义了 *loss_fn*, 则数据和标签会被分别传给 *network* 和 *loss_fn*, 此时数据集需要返回一个元组 (data, label)。如果数据集中有多个数据或者标签, 可以设置 *loss_fn* 为 None, 并在 *network* 中实现损失函数计算, 此时数据集返回的所有数据组成的元组 (data1, data2, data3, ...) 会传给 *network*。
- **valid_dataset** (Dataset) - 评估模型的数据集迭代器。默认值: None。
- **valid_frequency** (int, list) - 此参数只有在 *valid_dataset* 不为 None 时生效。如果为 int 类型, 表示执行推理的频率, 例如 *valid_frequency=2*, 则每 2 个训练 epoch 执行一次推理; 如果为 list 类型, 指明在哪几个 epoch 时执行推理, 例如 *valid_frequency=[1, 5]*, 则在第 1 个和第 5 个 epoch 执行推理。默认值: 1。
- **callbacks** (Optional[list[Callback], Callback]) - 训练过程中需要执行的回调对象或者回调对象列表。默认值: None。
- **dataset_sink_mode** (bool) - 训练数据是否直接下沉至处理器进行处理。使用 PYNATIVE_MODE 模式或 CPU 处理器时, 模型训练流程将以非下沉模式执行。默认值: False。
- **valid_dataset_sink_mode** (bool) - 推理数据是否直接下沉至处理器进行处理。默认值: False。
- **sink_size** (int) - 控制每次数据下沉的 step 数量。*dataset_sink_mode* 为 False 时, *sink_size* 设置无效。如果 *sink_size=-1*, 则每一次 epoch 下沉完整数据集。如果 *sink_size>0*, 则每一次 epoch 下沉数据量为 *sink_size* 的数据集。默认值: -1。
- **initial_epoch** (int) - 从哪个 epoch 开始训练, 一般用于中断恢复训练场景。默认值: 0。

样例:

```
>>> from mindspore import nn
>>> from mindspore.train import Model
>>>
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> train_dataset = create_dataset("train")
>>> valid_dataset = create_dataset("test")
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim, metrics={"accuracy"})
>>> model.fit(2, train_dataset, valid_dataset)
```

infer_predict_layout (*predict_data, skip_backend_compile=False)

在 `AutoParallel(Cell)` 使能的自动并行模式下, 为预测网络生成参数 layout。数据可以是单个或多个张量。

说明: 同一批次数据应放在一个张量中。

参数:

- **predict_data** (Union[Tensor, list[Tensor], tuple[Tensor]], 可选) - 预测样本, 数据可以是单个张量、张量列表或张量元组。
- **skip_backend_compile** (bool) - 生成参数 layout 时跳过后端编译流程。一般用于后端编译模型大小超过卡上内存的场景, 其他场景不建议开启, 开启时本次编译的缓存无法在二次编译时被使用。默认值: False。

返回:

Dict, 用于加载分布式 checkpoint 的参数 layout 字典。它总是作为 `load_distributed_checkpoint()` 函数的一个入参。

异常:

- **RuntimeError** - 非图模式 (GRAPH_MODE) 将会抛出该异常。

样例:

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> from mindspore.train import Model
>>> from mindspore.ops import operations as P
>>> from mindspore import context
>>> from mindspore.communication import init
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>>
>>> class Net(nn.Cell):
>>>     def __init__(self):
>>>         super(Net, self).__init__()
>>>         self.fc1 = nn.Dense(128, 768, activation='relu')
>>>         self.fc2 = nn.Dense(128, 768, activation='relu')
>>>         self.fc3 = nn.Dense(128, 768, activation='relu')
>>>         self.fc4 = nn.Dense(768, 768, activation='relu')
```

(下页继续)

(续上页)

```

>>> self.relu4 = nn.ReLU()
>>> self.relu5 = nn.ReLU()
>>> self.transpose = P.Transpose()
>>> self.matmul1 = P.MatMul()
>>> self.matmul2 = P.MatMul()
>>>
>>> def construct(self, x):
>>>     q = self.fc1(x)
>>>     k = self.fc2(x)
>>>     v = self.fc3(x)
>>>     k = self.transpose(k, (1, 0))
>>>     c = self.relu4(self.matmul1(q, k))
>>>     s = self.relu5(self.matmul2(c, v))
>>>     s = self.fc4(s)
>>>     return s
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> inputs = Tensor(np.ones([32, 128]).astype(np.float32))
>>> net = Net()
>>> parallel_net = AutoParallel(net, parallel_mode='semi_auto')
>>> model = Model(parallel_net)
>>> predict_map = model.infer_predict_layout(inputs)

```

infer_train_layout (train_dataset, dataset_sink_mode=True, sink_size=-1)

在 `AutoParallel(Cell)` 使能的自动并行模式下，为训练网络生成参数 layout。当前仅支持在数据下沉模式下使用。

警告：这是一个实验性 API，后续可能修改或删除。

说明：这是一个预编译函数。参数必须与 `Model.train()` 函数相同。

参数：

- **train_dataset** (Dataset) - 一个训练数据集迭代器。如果没有损失函数 (`loss_fn`)，返回一个包含多个数据的元组 (`data1, data2, data3, ...`) 并传递给网络。否则，返回一个元组 (`data, label`)，数据和标签将被分别传递给网络和损失函数。
- **dataset_sink_mode** (bool) - 决定是否以数据集下沉模式进行训练。默认值：True。PyNative 模式下或处理器为 CPU 时，训练模型流程使用的是数据不下沉 (non-sink) 模式。默认值：True。
- **sink_size** (int) - 控制每次数据下沉的 step 数量。`dataset_sink_mode` 为 False 时，`sink_size` 设置无效。如果 `sink_size == -1`，则每一次 epoch 下沉完整数据集。如果 `sink_size > 0`，则每一次 epoch 下沉数据量为 `sink_size` 的数据集。默认值：-1。

返回：

Dict，用于加载分布式 checkpoint 的参数 layout 字典。

样例：

```

>>> # This example should be run with multiple devices. Refer to the tutorial >_
    ↪ Distributed Training on
>>> # mindspore.cn.

```

(下页继续)

(续上页)

```

>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, nn
>>> from mindspore.train import Model
>>> from mindspore.communication import init
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>>
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> parallel_net = AutoParallel(net)
>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> loss_scale_manager = ms.FixedLossScaleManager()
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(parallel_net, loss_fn=loss, optimizer=optim, metrics=None,
...               loss_scale_manager=loss_scale_manager)
>>> layout_dict = model.infer_train_layout(dataset)

```

predict (*predict_data, backend=None, config=None)

输入样本得到预测结果。

参数:

- **predict_data** (Union[Tensor, list[Tensor], tuple[Tensor]], 可选) - 预测样本, 数据可以是单个张量、张量列表或张量元组。
- **backend** (str) - 选择预测后端, 该参数为实验性质特性, 主要用于 MindSpore Lite 云侧推理。默认值: None。
- **config** (dict, 可选) - 当后端为 'lite' 时, config 参数使能。config 有三种配置形式:
 1. **configPath** 定义配置文件的路径, 用于在构建模型期间传递用户定义选项。默认值: ""。

```
config = {"configPath" : "/home/user/config.ini"}
```

以下是 /home/user/config.ini 文件的内容:

```

[ascend_context]
rank_table_file=[path_a] (存储rank table文件的初始路径)
[execution_plan]
[op_name1]=data_type:float16 (名字为op_name1的算子设置数据类型为float16)
[op_name2]=data_type:float32 (名字为op_name2的算子设置数据类型为float32)

```

2. 将用户配置内容形成参数字典, 方式如下:

```

config = {"ascend_context" : {"rank_table_file" : "path_b"}, "execution_plan" : {
  ↳ "op_name1" : "data_type:float16", "op_name2" : "data_type:float32"}}

```

3. 当同时配置 **configPath** 和参数字典, 其中参数字典的优先级高于配置文件里的内容, 方式如下:

```
config = {"configPath" : "/home/user/config.ini", "ascend_context" : {"rank_table_
↪file" : "path_b"}, "execution_plan" : {"op_name3" : "data_type:float16", "op_
↪name4" : "data_type:float32"}}
```

注意：在示例中 `configPath` 中的 `rank_table_file=[path_a]`，字典中 `"ascend_context": {"rank_table_file": "path_b"}`，此时以字典中的 `"path_b"` 为准。

返回：

返回预测结果，类型是 `Tensor` 或 `Tensor` 元组。

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.train import Model
>>>
>>> input_data = Tensor(np.random.randint(0, 255, [1, 1, 32, 32]), mindspore.float32)
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> model = Model(LeNet5())
>>> result = model.predict(input_data)
```

property predict_network

获得该模型的预测网络。

返回：

预测网络实例。

train (*epoch*, *train_dataset*, *callbacks=None*, *dataset_sink_mode=False*, *sink_size=-1*, *initial_epoch=0*)

模型训练接口。

使用 `PYNATIVE_MODE` 模式或 `CPU` 处理器时，模型训练流程将以非下沉模式执行。

说明：

- 如果 `dataset_sink_mode` 配置为 `True`，数据将被送到处理器中。如果处理器是 `Ascend`，数据特征将被逐一传输，每次数据传输的上限是 256M。
- 如果 `dataset_sink_mode` 配置为 `True`，在 `PyNative` 模式，每个 `step` 结束时调用 `Callback` 实例的 `step_end` 方法。在 `Graph` 模式，每个 `epoch` 结束时调用 `Callback` 实例的 `step_end` 方法。
- 如果 `dataset_sink_mode` 配置为 `True`，数据集仅能在当前模型中使用。
- 如果 `sink_size` 大于零，每次 `epoch` 可以无限次遍历数据集，直到遍历数据量等于 `sink_size` 为止。
- 每次 `epoch` 将从上一次遍历的最后位置继续开始遍历。该接口会构建并执行计算图，如果使用前先执行了 `Model.build`，那么它会直接执行计算图而不构建。

参数：

- **epoch** (int) - 训练执行轮次。通常每个 `epoch` 都会使用全量数据集进行训练。当 `dataset_sink_mode` 设置为 `True` 且 `sink_size` 大于零时，则每个 `epoch` 训练次数为 `sink_size` 而不是数据集的总步数。如果 `epoch` 与 `initial_epoch` 一起使用，它表示断点续训多少个 `epoch`。
- **train_dataset** (Dataset) - 一个训练数据集迭代器。如果定义了 `loss_fn`，则数据和标签会被分别传给 `network` 和 `loss_fn`，此时数据集需要返回一个元组 (`data`, `label`)。如果数据集中有多个数据或者标签，

可以设置 `loss_fn` 为 `None`，并在 `network` 中实现损失函数计算，此时数据集返回的所有数据组成的元组 (`data1, data2, data3, ...`) 会传给 `network`。

- **callbacks** (Optional[list[Callback], Callback]) - 训练过程中需要执行的回调对象或者回调对象列表。默认值: `None`。
- **dataset_sink_mode** (bool) - 数据是否直接下沉至处理器进行处理。使用 `PYNATIVE_MODE` 模式或 CPU 处理器时，模型训练流程将以非下沉模式执行。默认值: `False`。
- **sink_size** (int) - 控制每次数据下沉的 `step` 数量。`dataset_sink_mode` 为 `False` 时，`sink_size` 设置无效。如果 `sink_size=-1`，则每一次 `epoch` 下沉完整数据集。如果 `sink_size>0`，则每一次 `epoch` 下沉数据量为 `sink_size` 的数据集。默认值: `-1`。
- **initial_epoch** (int) - 从哪个 `epoch` 开始训练，一般用于中断恢复训练场景。默认值: `0`。

样例:

```
>>> import mindspore as ms
>>> from mindspore import nn
>>> from mindspore.train import Model
>>>
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> loss_scale_manager = ms.FixedLossScaleManager(1024., False)
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim, metrics=None,
...               loss_scale_manager=loss_scale_manager)
>>> model.train(2, dataset)
```

property train_network

获得该模型的训练网络。

返回:

训练网络实例。

7.2 回调函数

<code>mindspore.train.BackupAndRestore</code>	在训练过程中备份和恢复训练参数的回调函数。
<code>mindspore.train.Callback</code>	用于构建 <code>Callback</code> 函数的基类。
<code>mindspore.train.CheckpointConfig</code>	保存 <code>checkpoint</code> 时的配置策略。
<code>mindspore.train.EarlyStopping</code>	当监控的指标停止改进时，停止训练。
<code>mindspore.train.FlopsUtilizationCollector</code>	<code>FlopsUtilizationCollector</code> 接口统计模型利用率信息 MFU，硬件利用率信息 HFU。
<code>mindspore.train.History</code>	将网络输出和评估指标的相关信息记录到 <code>History</code> 对象中。
<code>mindspore.train.LambdaCallback</code>	用于自定义简单的 <code>callback</code> 。
<code>mindspore.train.LearningRateScheduler</code>	用于在训练期间更改学习率。
<code>mindspore.train.LossMonitor</code>	训练场景下，监控训练的 <code>loss</code> ；边训练边推理场景下，监控训练的 <code>loss</code> 和推理的 <code>metrics</code> 。

下页继续

表 2 - 续上页

<code>mindspore.train.ModelCheckpoint</code>	<code>checkpoint</code> 的回调函数。
<code>mindspore.train.OnRequestExit</code>	响应用户关闭请求，退出训练或推理进程，保存 <code>checkpoint</code> 和 <code>mindir</code> 。
<code>mindspore.train.ReduceLROnPlateau</code>	动态调整学习率，当 <code>monitor</code> 停止改进时，降低学习率。
<code>mindspore.train.RunContext</code>	保存和管理模型的相关信息。
<code>mindspore.train.TrainFaultTolerance</code>	该回调函数用于开启 MindIO 的 TTP 特性，会嵌入训练的流程，完成 TTP 的初始化、上报、异常处理等操作。
<code>mindspore.train.TimeMonitor</code>	监控训练或推理的时间。

7.2.1 mindspore.train.BackupAndRestore

`class mindspore.train.BackupAndRestore (backup_dir, save_freq='epoch', delete_checkpoint=True)`
在训练过程中备份和恢复训练参数的回调函数。

说明：只能在训练过程使用这个方法。

参数：

- `backup_dir` (str) - 保存和恢复 `checkpoint` 文件的路径。
- `save_freq` (Union["epoch", int]) - 当设置为 "epoch" 时，在每个 epoch 进行备份；当设置为整数时，将在每隔 `save_freq` 个 epoch 进行备份。默认值： "epoch" 。
- `delete_checkpoint` (bool) - 如果 `delete_checkpoint=True`，将在训练结束的时候删除备份文件，否则保留备份文件。默认值： True 。

异常：

- `ValueError` - 如果 `backup_dir` 参数不是 str 类型。
- `ValueError` - 如果 `save_freq` 参数不是 "epoch" 或 int 类型。
- `ValueError` - 如果 `delete_checkpoint` 参数不是 bool 类型。

样例：

```
>>> from mindspore import nn
>>> from mindspore.train import Model, BackupAndRestore, RunContext
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim)
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> backup_ckpt = BackupAndRestore("backup")
>>> model.train(10, dataset, callbacks=backup_ckpt)
```

`on_train_begin(run_context)`
在训练开始时，加载备份的 `checkpoint` 文件。

参数：

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 `mindspore.train.RunContext`。

on_train_end (*run_context*)

在训练结束时，判断是否删除备份的 checkpoint 文件。

参数：

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 `mindspore.train.RunContext`。

on_train_epoch_end (*run_context*)

在每个 epoch 结束时，判断是否需要备份 checkpoint 文件。

参数：

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 `mindspore.train.RunContext`。

7.2.2 mindspore.train.Callback

class `mindspore.train.Callback`

用于构建 Callback 函数的基类。Callback 函数是一个上下文管理器，在运行模型时被调用。可以使用此机制进行一些自定义操作。

Callback 类的每个方法对应训练或推理过程的不同阶段，这些方法有相同的入参 *run_context*，用于保存训练或推理过程中模型的相关信息。定义 Callback 子类或自定义 Callback 时，请根据需要重写名称前缀为“on_train”或“on_eval”的方法，否则自定义的 Callback 在 *model.fit* 中使用时会产生错误。

自定义 Callback 场景下，在类方法中通过 *RunContext.original_args()* 方法可以获取模型训练或推理过程中已有的上下文信息，此信息为一个存储了已有属性的字典型变量。用户也可以在此信息中添加其他的自定义属性。此外，通过调用 *request_stop* 方法来停止训练过程。

样例：

```
>>> import numpy as np
>>> from mindspore import nn
>>> from mindspore import dataset as ds
>>> from mindspore.train import Model, Callback
>>> class Print_info(Callback):
...     def step_end(self, run_context):
...         cb_params = run_context.original_args()
...         print("step_num: ", cb_params.cur_step_num)
>>>
>>> print_cb = Print_info()
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> net = nn.Dense(10, 5)
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim)
>>> model.train(1, dataset, callbacks=print_cb)
step_num: 1
step_num: 2
```

begin (*run_context*)

在网络执行之前被调用一次。与 *on_train_begin* 和 *on_eval_begin* 方法具有兼容性。

说明: *begin* 接口将在后续版本中废弃，请使用 *on_train_begin* 和 *on_eval_begin*。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考[mindspore.train.RunContext](#)。

end (*run_context*)

在网络执行之后被调用一次。与 *on_train_end* 和 *on_eval_end* 方法具有兼容性。

说明: *end* 接口将在后续版本中废弃, 请使用 *on_train_end* 和 *on_eval_end*。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考[mindspore.train.RunContext](#)。

epoch_begin (*run_context*)

在每个 epoch 开始之前被调用。与 *on_train_epoch_begin* 和 *on_eval_epoch_begin* 方法具有兼容性。

说明: *epoch_begin* 接口将在后续版本中废弃, 请使用 *on_train_epoch_begin* 和 *on_eval_epoch_begin*。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考[mindspore.train.RunContext](#)。

epoch_end (*run_context*)

在每个 epoch 结束后被调用。与 *on_train_epoch_end* 和 *on_eval_epoch_end* 方法具有兼容性。

说明: *epoch_end* 接口将在后续版本中废弃, 请使用 *on_train_epoch_end* 和 *on_eval_epoch_end*。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考[mindspore.train.RunContext](#)。

on_eval_begin (*run_context*)

在网络执行推理之前被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考[mindspore.train.RunContext](#)。

on_eval_end (*run_context*)

在网络执行推理之后被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考[mindspore.train.RunContext](#)。

on_eval_epoch_begin (*run_context*)

在推理的 epoch 开始之前被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考[mindspore.train.RunContext](#)。

on_eval_epoch_end (*run_context*)

在推理的 epoch 结束后被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_eval_step_begin (*run_context*)

在推理的每个 step 开始之前被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_eval_step_end (*run_context*)

在推理的每个 step 完成后被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_train_begin (*run_context*)

在网络执行训练之前被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_train_end (*run_context*)

在网络执行训练结束后被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_train_epoch_begin (*run_context*)

在训练的每个 epoch 开始之前被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_train_epoch_end (*run_context*)

在训练的每个 epoch 结束后被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_train_step_begin (*run_context*)

在训练的每个 step 开始之前被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

on_train_step_end (*run_context*)

在训练的每个 step 完成后被调用。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

step_begin (*run_context*)

在每个 step 开始之前被调用。与 *on_train_step_begin* 和 *on_eval_step_begin* 方法具有兼容性。

说明: *step_begin* 接口将在后续版本中废弃, 请使用 *on_train_step_begin* 和 *on_eval_step_begin*。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 `mindspore.train.RunContext`。

step_end (*run_context*)

在每个 step 完成后被调用。与 `on_train_step_end` 和 `on_eval_step_end` 方法具有兼容性。

说明: `step_end` 接口将在后续版本中废弃, 请使用 `on_train_step_end` 和 `on_eval_step_end`。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 `mindspore.train.RunContext`。

7.2.3 mindspore.train.CheckpointConfig

```
class mindspore.train.CheckpointConfig (save_checkpoint_steps=1, save_checkpoint_seconds=0,
                                         keep_checkpoint_max=5, keep_checkpoint_per_n_minutes=0,
                                         integrated_save=True, async_save=False, saved_network=None,
                                         append_info=None, enc_key=None, enc_mode='AES-GCM',
                                         exception_save=False, crc_check=False, remove_redundancy=False,
                                         format='ckpt', **kwargs)
```

保存 checkpoint 时的配置策略。

说明:

- 在训练过程中, 如果数据集是通过数据通道传输的, 建议将 `save_checkpoint_steps` 设为循环下沉 step 数量的整数倍数。否则, 保存 checkpoint 的时机可能会有偏差。建议同时只设置一种触发保存 checkpoint 策略和一种保留 checkpoint 文件总数策略。如果同时设置了 `save_checkpoint_steps` 和 `save_checkpoint_seconds`, 则 `save_checkpoint_seconds` 无效。如果同时设置了 `keep_checkpoint_max` 和 `keep_checkpoint_per_n_minutes`, 则 `keep_checkpoint_per_n_minutes` 无效。
- `enc_mode` 和 `crc_check` 参数互斥, 不能同时配置。

参数:

- **save_checkpoint_steps** (int) - 每隔多少个 step 保存一次 checkpoint。默认值: 1。
- **save_checkpoint_seconds** (int) - 每隔多少秒保存一次 checkpoint。不能与 `save_checkpoint_steps` 同时使用。默认值: 0。
- **keep_checkpoint_max** (int) - 最多保存多少个 checkpoint 文件。默认值: 5。
- **keep_checkpoint_per_n_minutes** (int) - 每隔多少分钟保存一个 checkpoint 文件。不能与 `keep_checkpoint_max` 同时使用。默认值: 0。
- **integrated_save** (bool) - 在自动并行场景下, 是否合并保存拆分后的 Tensor。合并保存功能仅支持在自动并行场景中使用, 在手动并行场景中不支持。默认值: True。
- **async_save** (Union[bool, str], 可选) - 是否使用异步保存 checkpoint 文件或 safetensors 文件。True 时默认使用异步线程; 如果是 str 类型, 选择异步保存方式, 可以是 "process" 或 "thread"。默认值: False。
- **saved_network** (Cell) - 保存在 checkpoint 文件中的网络。如果 `saved_network` 没有被训练, 则保存 `saved_network` 的初始值。默认值: None。
- **append_info** (list) - 保存在 checkpoint 文件中的信息。支持 "epoch_num"、"step_num" 和 dict 类型。dict 的 key 必须是 str, dict 的 value 必须是 int、float、bool、string、Parameter 或 Tensor 中的一个。如果为 None, 则 `append_dict` 为空。默认值: None。

- **enc_key** (Union[None, bytes]) - 用于加密的字节类型 **key**。如果值为 **None**，则不需要加密。默认值: **None**。
- **enc_mode** (str) - 仅当 **enc_key** 不设为 **None** 时，该参数有效。指定加密模式，目前支持 'AES-GCM'、'AES-CBC' 和 'SM4-CBC'。默认值: 'AES-GCM'。
- **exception_save** (bool) - 当有异常发生时，是否保存当前 **checkpoint** 文件。默认值: **False**。
- **crc_check** (bool) - 是否在保存 **checkpoint** 时进行 **crc32** 计算，并将计算结果保存到 **ckpt** 尾部。默认值: **False**。
- **remove_redundancy** (bool) - 是否开启去冗余保存 **checkpoint**。去冗余是指去除数据并行模式下的冗余数据。默认值: **False**，不开启去冗余保存。
- **format** (str) - 输出文件的格式，可以是 "ckpt" 或 "safetensors"。默认值: "ckpt"。
- **kwargs** (dict) - 配置选项字典。

异常:

- **ValueError** - 输入参数的类型不正确。

样例:

```
>>> from mindspore import nn
>>> from mindspore.train import Model, CheckpointConfig, ModelCheckpoint
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim)
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> config = CheckpointConfig(save_checkpoint_seconds=100, keep_checkpoint_per_n_minutes=5,
→ saved_network=net)
>>> config.save_checkpoint_steps
1
>>> config.save_checkpoint_seconds
>>> config.keep_checkpoint_max
5
>>> config.keep_checkpoint_per_n_minutes
>>> config.integrated_save
True
>>> config.async_save
False
>>> config.saved_network
>>> config.enc_key
>>> config.enc_mode
'AES-GCM'
>>> config.append_dict
>>> config.get_checkpoint_policy
>>> ckpoint_cb = ModelCheckpoint(prefix='LeNet5', directory='./checkpoint', config=config)
>>> model.train(10, dataset, callbacks=ckpoint_cb)
```

property append_dict

获取需要额外保存到 **checkpoint** 中的字典的值。

返回:

dict: 字典中的值。

property async_save

获取是否异步保存 checkpoint 或者异步保存方式。

返回：

(bool, str): 是否异步保存 checkpoint 或异步保存方式。

property crc_check

获取是否启用 crc 校验。

返回：

bool: 是否启用 crc 校验。

property enc_key

获取加密的 key 值。

返回：

(None, bytes): 加密的 key 值。

property enc_mode

获取加密模式。

返回：

str: 加密模式。

get_checkpoint_policy()

获取 checkpoint 的保存策略。

返回：

dict: checkpoint 的保存策略。

property integrated_save

获取是否合并保存拆分后的 Tensor。

返回：

bool: 获取是否合并保存拆分后的 Tensor。

property keep_checkpoint_max

获取最多保存 checkpoint 文件的数量。

返回：

int: 最多保存 checkpoint 文件的数量。

property keep_checkpoint_per_n_minutes

获取每隔多少分钟保存一个 checkpoint 文件。

返回：

int: 每隔多少分钟保存一个 checkpoint 文件。

property map_param_inc

获取是否增量保存 MapParameter。

返回：

bool: 是否增量保存 MapParameter。

property save_checkpoint_seconds

获取每隔多少秒保存一次 checkpoint 文件。

返回：

int: 每隔多少秒保存一次 checkpoint 文件。

property save_checkpoint_steps

获取每隔多少个 step 保存一次 checkpoint 文件。

返回:

int: 每隔多少个 step 保存一次 checkpoint 文件。

property saved_network

获取需要保存的网络。

返回:

Cell: 需要保存的网络。

7.2.4 mindspore.train.EarlyStopping

class mindspore.train.EarlyStopping (*monitor='eval_loss', min_delta=0, patience=0, verbose=False, mode='auto', baseline=None, restore_best_weights=False*)

当监控的指标停止改进时，停止训练。

假设 *monitor* 是 'accuracy', 那么, *mode* 将为 'max', 因为训练的目标是准确率的提高。*model.fit()* 边训练边验证场景下, 将记录 *monitor* 的变化。当在 *patience* 个 *epoch* 范围内指标效果变好的程度没有超过 *min_delta* 时, 将调用 *run_context.request_stop()* 方法来终止训练。

参数:

- **monitor** (str) - 监控指标。如果是边训练边推理场景, 合法的 *monitor* 配置值可以为 'loss'、'eval_loss' 以及实例化 *Model* 时传入的 *metric* 名称; 如果在训练时不做推理, 合法的 *monitor* 配置值为 'loss'。当 *monitor* 为 'loss' 时, 如果训练网络有多个输出, 默认取第一个值为训练损失值。默认值: 'eval_loss'。
- **min_delta** (float) - *monitor* 指标变化的最小阈值, 超过此阈值才视为 *monitor* 的变化。默认值: 0。
- **patience** (int) - *monitor* 相对历史最优值变好超过 *min_delta*, 视为当前 *epoch* 的模型效果有所改善。*patience* 为等待的无改善 *epoch* 的数量。当内部等待的 *epoch* 数 *self.wait* 大于等于 *patience* 时, 训练停止。默认值: 0。
- **verbose** (bool) - 是否打印相关信息。默认值: False。
- **mode** (str) - {'auto', 'min', 'max'} 中的一种。默认值: 'auto'。
 - 'min' 模式下, 将在指标不再减小时, 执行早停。
 - 'max' 模式下, 将在指标不再增大时, 执行早停。
 - 'auto' 模式下, 将根据当前 *monitor* 指标的特点自动设置。
- **baseline** (float) - 模型效果的基线。当前 *monitor* 相对历史最优值变好, 且好于 *baseline* 时, 内部的等待 *epoch* 计数器被清零。默认值: None。
- **restore_best_weights** (bool) - 是否自动保存最优模型的权重。默认值: False。

异常:

- **ValueError** - 当 *mode* 不在 {'auto', 'min', 'max'} 中。
- **ValueError** - 当传入的 *monitor* 返回值不是标量。

样例:

```
>>> from mindspore import nn
>>> from mindspore.train import Model, EarlyStopping
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim, metrics={"acc"})
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> cb = EarlyStopping(monitor="acc", patience=3, verbose=True)
>>> model.fit(10, dataset, callbacks=cb)
```

on_train_begin(*run_context*)

训练开始时初始化相关的变量。

参数:

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 `mindspore.train.RunContext`。

on_train_end(*run_context*)

打印第几个 epoch 执行早停。

参数:

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 `mindspore.train.RunContext`。

on_train_epoch_end(*run_context*)

训练过程中, 若监控指标在等待 *patience* 个 epoch 后仍没有改善, 则停止训练。

参数:

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 `mindspore.train.RunContext`。

7.2.5 mindspore.train.FlopsUtilizationCollector

class mindspore.train.FlopsUtilizationCollector (*data_size*, *computility*=1, *full_flops*=True, *enable_ma_collector*=False)

FlopsUtilizationCollector 接口统计模型利用率信息 MFU, 硬件利用率信息 HFU。

当前接口只统计 MatMul、BatchMatMul、FlashAttentionScore、Conv2D 算子的正反向 flops 信息。

只支持静态图静态 shape 模式。

参数:

- **data_size** (int) - 表示每隔多少个 step 打印一次信息。
- **computility** (int) - 表示每张计算卡的峰值算力。默认值: 1。
- **full_flops** (bool) - 表示是否统计完整的模型信息。如果设置为 True, 会统计完整的模型信息; 如果设置为 False, 将会统计对应每张卡的分片模型信息。默认值: True。
- **enable_ma_collector** (bool) - 表示是否将 flops 写日志, 提供给云上任务进行采集。默认值: False。

异常:

- **TypeError** - *data_size* 不是正整数。
- **TypeError** - *full_flops* 不是布尔类型。

- **TypeError** - `enable_ma_collector` 不是布尔类型。
- **AssertionError** - 训练模式不是静态图或者不是静态 shape。

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> from mindspore import nn
>>> from mindspore.train import Model, FlopsUtilizationCollector
>>> from mindspore import context
>>> context.set_context(mode=context.GRAPH_MODE)
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> net = nn.Dense(10, 5)
>>> crit = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> opt = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> flops_callback = FlopsUtilizationCollector(train_dataset.get_dataset_size(),
->comptility=10e6)
>>> model = Model(network=net, optimizer=opt, loss_fn=crit, metrics={"recall"})
>>> model.train(2, train_dataset, callbacks=[flops_callback])
Full model flops is 6400, Full hardware flops is 6400, Shard model flops is 6400, Shard
->hardware flops is 6400
Train per step time: 135.572 ms, mfu:0.47% hfu:0.47%
Train per step time: 1.317 ms, mfu:48.59% hfu:48.59%
```

step_begin (*run_context*)
在 step 开始时记录时间。

参数:

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

step_end (*run_context*)
在 step 结束时打印模型利用率信息 MFU，硬件利用率信息 HFU。

参数:

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

7.2.6 mindspore.train.History

class mindspore.train.History

将网络输出和评估指标的相关信息记录到 *History* 对象中。

- 用户不自定义训练网络或评估网络情况下，则记录损失值。
- 用户自定义了训练网络或评估网络的情况下，
 - 如果定义的网络返回 *Tensor* 或 *numpy.ndarray*，则记录此返回值均值。
 - 如果返回 *tuple* 或 *list*，则记录第一个元素。

说明: 通常使用在[mindspore.train.Model.train\(\)](#) 和[mindspore.train.Model.fit\(\)](#) 中。

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> from mindspore import nn
>>> from mindspore.train import Model, History
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> net = nn.Dense(10, 5)
>>> crit = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> opt = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> history_cb = History()
>>> model = Model(network=net, optimizer=opt, loss_fn=crit, metrics={"recall"})
>>> model.train(2, train_dataset, callbacks=[history_cb])
>>> print(history_cb.epoch)
{'epoch': [1, 2]}
>>> print(history_cb.history)
{'net_output': [1.607877, 1.6033841]}
```

begin (*run_context*)

训练开始时，初始化 History 对象的 epoch 属性。

参数：

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

epoch_end (*run_context*)

epoch 结束时，记录网络输出和评估指标的相关信息。

参数：

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

7.2.7 mindspore.train.LambdaCallback

```
class mindspore.train.LambdaCallback (on_train_epoch_begin=None, on_train_epoch_end=None,  
                                       on_train_step_begin=None, on_train_step_end=None, on_train_begin=None,  
                                       on_train_end=None, on_eval_epoch_begin=None, on_eval_epoch_end=None,  
                                       on_eval_step_begin=None, on_eval_step_end=None, on_eval_begin=None,  
                                       on_eval_end=None)
```

用于自定义简单的 callback。

使用匿名函数构建 callback，定义的匿名函数将在 `mindspore.train.Model.{train | eval | fit}` 的对应阶段被调用。

请注意，callback 的每个阶段都需要一个位置参数：`run_context`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **on_train_epoch_begin** (Function) - 训练的每个 epoch 开始时被调用。默认值：None。
- **on_train_epoch_end** (Function) - 训练的每个 epoch 结束时被调用。默认值：None。
- **on_train_step_begin** (Function) - 训练的每个 step 开始时被调用。默认值：None。
- **on_train_step_end** (Function) - 训练的每个 step 结束时被调用。默认值：None。
- **on_train_begin** (Function) - 模型训练开始时被调用。默认值：None。

- **on_train_end** (Function) - 模型训练结束时被调用。默认值: None。
- **on_eval_epoch_begin** (Function) - 推理的每个 epoch 开始时被调用。默认值: None。
- **on_eval_epoch_end** (Function) - 推理的每个 epoch 结束时被调用。默认值: None。
- **on_eval_step_begin** (Function) - 推理的每个 step 开始时被调用。默认值: None。
- **on_eval_step_end** (Function) - 推理的每个 step 结束时被调用。默认值: None。
- **on_eval_begin** (Function) - 模型推理开始时被调用。默认值: None。
- **on_eval_end** (Function) - 模型推理结束时被调用。默认值: None。

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> from mindspore import nn
>>> from mindspore.train import Model, LambdaCallback
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> net = nn.Dense(10, 5)
>>> crit = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> opt = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> lambda_callback = LambdaCallback(on_train_epoch_end=
... lambda run_context: print("loss:", run_context.original_args().net_outputs))
>>> model = Model(network=net, optimizer=opt, loss_fn=crit, metrics={"recall"})
>>> model.train(2, train_dataset, callbacks=[lambda_callback])
loss: 1.6127687
loss: 1.6106578
```

7.2.8 mindspore.train.LearningRateScheduler

class mindspore.train.LearningRateScheduler (*learning_rate_function*)

用于在训练期间更改学习率。

参数:

- **learning_rate_function** (Function) - 在训练期间更改学习率的函数。

样例:

```
>>> import numpy as np
>>> from mindspore import nn
>>> from mindspore.train import Model, LearningRateScheduler
>>> from mindspore import dataset as ds
...
>>> def learning_rate_function(lr, cur_step_num):
...     if cur_step_num%1000 == 0:
...         lr = lr*0.1
...     return lr
...
>>> lr = 0.1
>>> momentum = 0.9
>>> net = nn.Dense(10, 5)
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), learning_rate=lr, momentum=momentum)
>>> model = Model(net, loss_fn=loss, optimizer=optim)
```

(下页继续)

(续上页)

```

...
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> model.train(1, dataset, callbacks=[LearningRateScheduler(learning_rate_function)],
...           dataset_sink_mode=False)

```

step_end(*run_context*)

在 step 结束时更改学习率。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

7.2.9 mindspore.train.LossMonitor

class mindspore.train.LossMonitor(*per_print_times=1*)

训练场景下, 监控训练的 loss; 边训练边推理场景下, 监控训练的 loss 和推理的 metrics。

如果 loss 是 NAN 或 INF, 则终止训练。

说明: 如果 *per_print_times* 为 0, 则不打印 loss。

参数:

- **per_print_times** (int) - 表示每隔多少个 step 打印一次 loss。默认值: 1。

异常:

- **ValueError** - 当 *per_print_times* 不是整数或小于零。

样例:

```

>>> from mindspore import nn
>>> from mindspore.train import Model, LossMonitor
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim)
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> loss_monitor = LossMonitor()
>>> model.train(10, dataset, callbacks=loss_monitor)

```

on_train_epoch_end(*run_context*)

LossMonitor 用于 [mindspore.train.Model.fit\(\)](#), 即边训练边推理场景时, 打印训练的 loss 和当前 epoch 推理的 metrics。

参数:

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 [mindspore.train.RunContext](#)。

step_end(*run_context*)

step 结束时打印训练 loss。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 [mindspore.train.RunContext](#)。

7.2.10 mindspore.train.ModelCheckpoint

class mindspore.train.**ModelCheckpoint** (*prefix='CKP', directory=None, config=None*)

checkpoint 的回调函数。

在训练过程中，调用该方法可以保存网络参数。

说明：在分布式训练场景下，请为每个训练进程指定不同的目录，来保存 checkpoint 文件。否则，可能会训练失败。如果在 [Model](#) 方法中使用此回调函数，默认会把优化器中的参数保存到 checkpoint 文件中。

参数：

- **prefix** (Union[str, callable object]) - checkpoint 文件的前缀名称，或者用来生成名称的可调用对象。默认值：'CKP'。
- **directory** (Union[str, callable object]) - 保存 checkpoint 文件的文件夹路径，或者用来生成路径的可调用对象。默认情况下，文件保存在当前目录下。默认值：None。
- **config** (CheckpointConfig) - checkpoint 策略配置。默认值：None。

异常：

- **ValueError** - 如果 prefix 参数不是 str 类型或包含 '/' 字符，且不是可调用对象。
- **ValueError** - 如果 directory 参数不是 str 类型，且不是可调用对象。
- **TypeError** - config 不是 CheckpointConfig 类型。

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> from mindspore import nn
>>> from mindspore.train import Model, ModelCheckpoint
>>>
>>> data = {"x": np.float32(np.random.rand(64, 10)), "y": np.random.randint(0, 5, (64,))}
>>> train_dataset = ds.NumpySlicesDataset(data=data).batch(32)
>>> net = nn.Dense(10, 5)
>>> crit = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> opt = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> ckpt_callback = ModelCheckpoint(prefix="myckpt")
>>> model = Model(network=net, optimizer=opt, loss_fn=crit)
>>> model.train(2, train_dataset, callbacks=[ckpt_callback])
```

end(*run_context*)

在训练结束后，会保存最后一个 step 的 checkpoint。

参数：

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 [mindspore.train.RunContext](#)。

property latest_ckpt_file_name
返回最新的 checkpoint 路径和文件名。

step_end(*run_context*)
在 step 结束时保存 checkpoint。

参数:

- **run_context** (RunContext) - 包含模型的一些基本信息。详情请参考 `mindspore.train.RunContext`。

7.2.11 mindspore.train.OnRequestExit

class mindspore.train.OnRequestExit (*save_ckpt=True, save_mindir=True, file_name='Net', directory='.', sig=signal.SIGTERM, config_file=None*)

响应用户关闭请求，退出训练或推理进程，保存 checkpoint 和 mindir。

在训练开始前，注册 OnRequestExit 回调，当用户想要退出训练进程并保存训练数据时，可通过发送注册的退出信号 *sig* 到训练进程，或者将 *config_file* 参数所指定的 JSON 文件中的 *GracefulExit* 字段修改为 1。训练进程执行完当前 step 后，保存当前训练状态，包括 checkpoint 和 mindir，然后退出训练过程。

参数:

- **save_ckpt** (bool) - 退出训练或推理进程时，是否保存 checkpoint。默认值: True。
- **save_mindir** (bool) - 退出训练或推理进程时，是否保存 mindir。默认值: True。
- **file_name** (str) - 退出训练或推理进程时，保存的 checkpoint 和 mindir 的名字。checkpoint 文件加.ckpt 后缀，mindir 文件加.mindir 后缀。默认值: 'Net'。
- **directory** (str) - 退出训练或推理进程时，指定文件保存目录。在该目录下会根据 rank_id 生成 rank_{id} 目录，存储 checkpoint 和 mindir 文件。默认值: './'。
- **sig** (int) - 用户注册的退出信号，该信号必须是可捕获可忽略的。当进程收到该信号时，退出训练或者推理。默认值: signal.SIGTERM。
- **config_file** (str) - 进程优雅退出 JSON 格式配置文件，关键字为: {"GracefulExit": 1}。默认值: None。

异常:

- **ValueError** - *save_ckpt* 不是 bool 值。
- **ValueError** - *save_mindir* 不是 bool 值。
- **ValueError** - *file_name* 不是字符串。
- **ValueError** - *directory* 不是字符串。
- **ValueError** - *sig* 不是 int 值，或者是 signal.SIGTERM。

样例:

```
>>> from mindspore import nn
>>> from mindspore.train import Model, TimeMonitor
>>> import mindspore as ms
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim)
```

(下页继续)

(续上页)

```
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> on_request_exit = ms.train.OnRequestExit(file_name='LeNet5')
>>> model.train(10, dataset, callbacks=on_request_exit)
```

on_eval_begin (*run_context*)

在推理开始时，注册用户传入退出信号的处理程序。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_eval_end (*run_context*)

在推理结束时，如果接收到退出信号，根据用户配置，保存 checkpoint 和 mindir。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_eval_step_end (*run_context*)

在推理 step 结束时，如果接收到退出信号，将 *run_context* 的 *_stop_requested* 属性置为 True。在本轮推理结束后，退出推理。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_begin (*run_context*)

在训练开始时，注册用户传入退出信号的处理程序。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_end (*run_context*)

在训练结束时，如果接收到退出信号，根据用户配置，保存 checkpoint 和 mindir。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_epoch_end (*run_context*)

在训练 epoch 结束时，如果接收到退出信号，将 *run_context* 的 *_stop_requested* 属性置为 True。在本轮训练结束后，退出训练。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_step_begin (*run_context*)

step 开始时校验是否有退出信号，或者校验 *config_file* 文件中的 *GracefulExit* 是否为 1。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_step_end (*run_context*)

根据配置保存 checkpoint 文件或 mindir 文件，并退出训练。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

7.2.12 mindspore.train.ReduceLROnPlateau

class mindspore.train.ReduceLROnPlateau (monitor='eval_loss', factor=0.1, patience=10, verbose=False, mode='auto', min_delta=1e-4, cooldown=0, min_lr=0)

动态调整学习率，当 *monitor* 停止改进时，降低学习率。

训练中一旦学习停滞（loss 不再降低或 acc 不再提高），会降低 2-10 倍的学习率使模型继续学习。此回调监控训练过程，当在 *patience* 个 epoch 范围内指标效果没有变好时，学习率就会降低。

说明：暂不支持分组学习率场景。

参数：

- **monitor** (str) - 监控指标。如果是边训练边推理场景，配置值可以为 "loss"、"eval_loss"，以及实例化 *Model* 时传入的 *metric* 名称；如果在训练时不做推理，配置值可以为 "loss"。当 *monitor* 为 "loss" 时，如果训练网络有多个输出，默认取第一个值为训练损失值。默认值："eval_loss"。
- **factor** (float) - 学习率变化系数，范围在 0-1 之间。默认值：0.1。
- **patience** (int) - *monitor* 相对历史最优值变好超过 *min_delta* 时，视为当前 epoch 的模型效果有所改善。*patience* 为等待的无改善 epoch 的数量，当内部等待的 epoch 数 *self.wait* 大于等于 *patience* 时，训练停止。默认值：10。
- **verbose** (bool) - 是否打印相关信息。默认值：False。
- **mode** (str) - 'auto'、'min'、'max' 中的一种。默认值：'auto'。
 - 'min' 模式下，将在指标不再减小时，改变学习率。
 - 'max' 模式下，将在指标不再增大时，改变学习率。
 - 'auto' 模式，将根据当前 *monitor* 指标的特点自动设置。
- **min_delta** (float) - *monitor* 指标变化的最小阈值，超过此阈值才视为 *monitor* 的变化。默认值：1e-4。
- **cooldown** (int) - 减小学习率后，在接下来的 *cooldown* 个 epoch 中不执行操作。默认值：0。
- **min_lr** (float) - 学习率最小设定值。默认值：0。

异常：

- **ValueError** - 当 *mode* 不在 {'auto', 'min', 'max'} 中。
- **ValueError** - 分组学习率场景，或动态学习率场景下，当获取到的学习率不是 *parameter* 类型。
- **ValueError** - 当传入的 *monitor* 返回值不是标量。

样例：

```
>>> from mindspore import nn
>>> from mindspore.train import Model, ReduceLROnPlateau
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim, metrics={"acc"})
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
```

(下页继续)

(续上页)

```
>>> cb = ReduceLROnPlateau(monitor="acc", patience=3, verbose=True)
>>> model.fit(10, dataset, callbacks=cb)
```

on_train_begin (*run_context*)
训练开始时初始化相关的变量。

- 参数:
- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 *mindspore.train.RunContext*。

on_train_epoch_end (*run_context*)
训练过程中, 若监控指标在等待 *patience* 个 epoch 后仍没有改善, 则改变学习率。

- 参数:
- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 *mindspore.train.RunContext*。

7.2.13 mindspore.train.RunContext

class mindspore.train.RunContext (*original_args*)
保存和管理模型的相关信息。

RunContext 主要用于收集训练或推理过程中模型的上下文相关信息, 并作为入参传入 *callback* 对象中来实现信息的共享。

Callback 的类方法中, 调用 *RunContext.original_args()* 可以获取模型当前的上下文信息, 用户也可以为此信息添加额外的自定义属性, 同时 *request_stop()* 方法可以控制训练过程的停止。

RunContext.original_args() 存储的模型信息为一个字典型变量, 在训练和推理过程会存储不同的属性。详情如下:

训练过程支持的属性	推理过程支持的属性	说明
train_network		包含了优化器和损失的训练网络
epoch_num		训练的 epoch 数
train_dataset		训练集
loss_fn		损失函数
optimizer		优化器
parallel_mode		并行模式
device_number		设备编号
train_dataset_element		当前 step 的训练数据
last_save_ckpt_step		最后一次存储 ckpt 的 step
latest_ckpt_file		ckpt 文件名
cur_epoch_num		当前的 epoch
	eval_network	评估网络
	valid_dataset	验证集
	metrics	评估指标
mode	mode	"train" 或 "eval" 模式
batch_num	batch_num	训练或推理的 batch 数
list_callback	list_callback	回调列表
network	network	基础的网络结构
cur_step_num	cur_step_num	当前的训练或推理的 step
dataset_sink_mode	dataset_sink_mode	训练或推理的数据是否下沉
net_outputs	net_outputs	训练或推理的网络输出

参数:

- **original_args** (dict) - 模型的相关信息。

样例:

```
>>> from mindspore import Tensor
>>> from mindspore.train import RunContext
>>> cb_params = {}
>>> cb_params["cur_epoch_num"] = 4
>>> cb_params["epoch_num"] = 4
>>> cb_params["cur_step_num"] = 2
>>> cb_params["batch_num"] = 2
>>> cb_params["net_outputs"] = Tensor(2.0)
>>> run_context = RunContext(cb_params)
>>> whether_stop = run_context.get_stop_requested()
```

get_stop_requested()

获取是否停止训练的标志。

返回:

bool, 如果为 True, 则 *Model.train()* 停止迭代。

original_args()

获取模型相关信息的对象。

返回:

dict, 含有模型的相关信息的对象。

request_stop()

在训练期间设置停止请求。

可以使用此函数请求停止训练。*Model.train()* 会检查是否调用此函数。

7.2.14 mindspore.train.TrainFaultTolerance

class mindspore.train.TrainFaultTolerance (*ckpt_save_path=None, **kwargs*)

该回调函数用于开启 MindIO 的 TTP 特性, 会嵌入训练的流程, 完成 TTP 的初始化、上报、异常处理等操作。

说明: 该特性仅支持 Ascend 后端的静态图模式, 并且只支持 sink_size 值小于等于 1 的场景。

参数:

- **ckpt_save_path** (str) - 异常发生时 ckpt 保存的路径, 该路径是一个目录。保存时, 会在该目录下创建新的名为 'ttp_saved_checkpoints-step_{cur_step_num}' 目录。默认值为: None。
- **kwargs** (dict) - 其他字典类型参数。

异常:

- **Exception** - TTP 初始化失败, 会对外抛 Exception 异常。
- **ModuleNotFoundError** - Mindio TTP whl 包未安装。

样例:

说明：在运行 TrainFaultTolerance 的用例之前，需要配置相应环境变量。推荐使用 msrun 进行分布式的启动，参考 msrun 启动方式。用例应该在 4 张卡上运行。

```
>>> import numpy as np
>>> import os
>>> import math
>>> import mindspore as ms
>>> import mindspore.dataset as ds
>>> from mindspore import nn, ops, Parameter, train
>>> from mindspore.communication import init, get_rank
>>> from mindspore.common.initializer import initializer, HeUniform
>>> from mindspore.train import Model, TrainFaultTolerance
>>> from mindspore import dataset as ds
>>> ms.set_context(mode=ms.GRAPH_MODE, jit_level='O2')
>>> ms.set_auto_parallel_context(parallel_mode=ms.ParallelMode.SEMI_AUTO_PARALLEL, pipeline_
↳stages=2)
>>> init()
>>> ms.set_seed(1)
>>> ms.set_auto_parallel_context(strategy_ckpt_config={"save_file":
...                                     "./src_pipeline_strategy/src_strategy_{}.ckpt".format(get_
↳rank())})
>>> class MatMulCell(nn.Cell):
...     def __init__(self, param=None, shape=None):
...         super().__init__()
...         if shape is None:
...             shape = [28 * 28, 512]
...         weight_init = HeUniform(math.sqrt(5))
...         self.param = Parameter(initializer(weight_init, shape), name="param")
...         if param is not None:
...             self.param = param
...         self.print = ops.Print()
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         out = self.matmul(x, self.param)
...         self.print("out is:", out)
...         return out
>>>
>>> class Network(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.flatten = nn.Flatten()
...         self.layer1 = MatMulCell()
...         self.relu1 = nn.ReLU()
...         self.layer2 = nn.Dense(512, 512)
...         self.relu2 = nn.ReLU()
...         self.layer3 = nn.Dense(512, 10)
...
...     def construct(self, x):
...         x = self.flatten(x)
...         x = self.layer1(x)
...         x = self.relu1(x)
...         x = self.layer2(x)
...         x = self.relu2(x)
...         logits = self.layer3(x)
...         return logits
```

(下页继续)

(续上页)

```

>>>
>>> net = Network()
>>> net.layer1.pipeline_stage = 0
>>> net.relu1.pipeline_stage = 0
>>> net.layer2.pipeline_stage = 0
>>> net.relu2.pipeline_stage = 1
>>> net.layer3.pipeline_stage = 1
>>>
>>> def create_dataset(batch_size):
...     dataset_path = os.getenv("DATA_PATH")
...     dataset = ds.MnistDataset(dataset_path)
...     image_transforms = [
...         ds.vision.Rescale(1.0 / 255.0, 0),
...         ds.vision.Normalize(mean=(0.1307,), std=(0.3081,)),
...         ds.vision.HWC2CHW()
...     ]
...     label_transform = ds.transforms.TypeCast(ms.int32)
...     dataset = dataset.map(image_transforms, 'image')
...     dataset = dataset.map(label_transform, 'label')
...     dataset = dataset.batch(batch_size)
...     return dataset
>>>
>>> dataset = create_dataset(32)
>>>
>>> optimizer = nn.SGD(net.trainable_params(), 1e-2)
>>> optimizer_wrapper = nn.OptTFTWrapper(optimizer)
>>> loss_fn = nn.CrossEntropyLoss()
>>>
>>> net_with_loss = nn.Pipeline(nn.WithLossCell(net, loss_fn), 4)
>>> net_with_loss.set_train()
>>> model = Model(net_with_loss, optimizer=optimizer_wrapper)
>>> tft_cb = TrainFaultTolerance()
>>> loss_cb = train.LossMonitor(1)
>>> model.train(1, dataset, callbacks=[tft_cb, loss_cb])

```

end(*run_context*)

训练结束，解注册 MindIO TTP。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

classmethod get_optimizer_wrapper(*origin_opt_cls*)

使用 TFT 功能时的优化器类封装函数。

参数：

- **origin_opt_cls** (Class) - 原优化器类。

on_train_begin(*run_context*)

训练开始时，向 MindIO TTP 注册训练时参数。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_step_end(*run_context*)

每个 step 完成时，进行 MindIO TTP 的上报。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

7.2.15 mindspore.train.TimeMonitor

class mindspore.train.**TimeMonitor** (*data_size=None, data_time=False*)

监控训练或推理的时间。

参数：

- **data_size** (int) - 表示每隔多少个 step 打印一次信息。如果程序在训练期间获取到 Model 的 *batch_num*，则将 *data_size* 设为 *batch_num* 的值，否则将直接使用 *data_size* 的值。如果程序在训练期间没有获取到 Model 的 *batch_num* 且没有设置 *data_size*，程序将会报错。默认值：None。
- **data_time** (bool) - 表示是否打印在 Host 侧获取数据的时间。请注意：在非数据集下沉模式时，数据获取和网络计算是按同步执行的，而在数据集下沉模式时，它们是异步执行的。默认值：False。

异常：

- **ValueError** - *data_size* 不是正整数。
- **TypeError** - *data_time* 不是布尔类型。

样例：

```
>>> from mindspore import nn
>>> from mindspore.train import Model, TimeMonitor
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
>>> optim = nn.Momentum(net.trainable_params(), 0.01, 0.9)
>>> model = Model(net, loss_fn=loss, optimizer=optim)
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> time_monitor = TimeMonitor()
>>> model.train(10, dataset, callbacks=time_monitor)
```

epoch_begin (*run_context*)

在 epoch 开始时记录时间。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

epoch_end (*run_context*)

在 epoch 结束时打印 epoch 的耗时。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_step_begin (*run_context*)

在 step 开始时记录时间。

参数：

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考[mindspore.train.RunContext](#)。

on_train_step_end (*run_context*)

在 step 结束时记录时间。

参数:

- **run_context** (RunContext) - 包含模型的相关信息。详情请参考 `mindspore.train.RunContext`。

7.3 评价指标

接口名	概述	支持平台
<code>mindspore.train.Accuracy</code>	计算数据分类的正确率，包括二分类和多分类。	Ascend GPU CPU
<code>mindspore.train.BleuScore</code>	计算 BLEU 分数。	Ascend GPU CPU
<code>mindspore.train.ConfusionMatrix</code>	计算混淆矩阵 (Confusion Matrix)，通常用于评估分类模型的性能，包括二分类和多分类场景。	Ascend GPU CPU
<code>mindspore.train.ConfusionMatrixMetric</code>	计算与混淆矩阵相关的度量。	Ascend GPU CPU
<code>mindspore.train.CosineSimilarity</code>	计算余弦相似度。	Ascend GPU CPU
<code>mindspore.train.Dice</code>	集合相似性度量。	Ascend GPU CPU
<code>mindspore.train.F1</code>	计算 F1 评分。	Ascend GPU CPU
<code>mindspore.train.Fbeta</code>	计算 Fbeta 评分。	Ascend GPU CPU
<code>mindspore.train.HausdorffDistance</code>	计算 Hausdorff 距离。	Ascend GPU CPU
<code>mindspore.train.Loss</code>	计算 loss 的平均值。	Ascend GPU CPU
<code>mindspore.train.MAE</code>	计算平均绝对误差 MAE (Mean Absolute Error)。	Ascend GPU CPU
<code>mindspore.train.MeanSurfaceDistance</code>	计算从 y_{pred} 到 y 的平均表面距离。	Ascend GPU CPU
<code>mindspore.train.Metric</code>	用于计算评估指标的基类。	Ascend GPU CPU
<code>mindspore.train.MSE</code>	测量均方差 MSE (Mean Squared Error)。	Ascend GPU CPU
<code>mindspore.train.OcclusionSensitivity</code>	用于计算神经网络对给定图像的遮挡灵敏度 (Occlusion Sensitivity)，表示了图像的哪些部分对神经网络的分类决策最重要。	Ascend GPU CPU
<code>mindspore.train.Perplexity</code>	计算困惑度 (perplexity)。	Ascend GPU CPU
<code>mindspore.train.Precision</code>	计算数据分类的精度，包括单标签场景和多标签场景。	Ascend GPU CPU
<code>mindspore.train.Recall</code>	计算数据分类的召回率，包括单标签场景和多标签场景。	Ascend GPU CPU
<code>mindspore.train.ROC</code>	计算 ROC 曲线。	Ascend GPU CPU
<code>mindspore.train.RootMeanSquareDistance</code>	计算从 y_{pred} 到 y 的均方根表面距离。	Ascend GPU CPU
<code>mindspore.train.Top1CategoricalAccuracy</code>	计算 top-1 分类正确率。	Ascend GPU CPU
<code>mindspore.train.Top5CategoricalAccuracy</code>	计算 top-5 分类正确率。	Ascend GPU CPU
<code>mindspore.train.TopKCategoricalAccuracy</code>	计算 top-k 分类正确率。	Ascend GPU CPU

7.3.1 mindspore.train.Accuracy

class mindspore.train.**Accuracy** (*eval_type='classification'*)

计算数据分类的正确率，包括二分类和多分类。

此类创建两个局部变量，预测正确的样本数和总样本数，用于计算 y_{pred} 和 y 的匹配率。此匹配率即为 accuracy。

$$\text{accuracy} = \frac{\text{true\_positive} + \text{true\_negative}}{\text{true\_positive} + \text{true\_negative} + \text{false\_positive} + \text{false\_negative}}$$

参数：

- **eval_type** (str) - 评估的数据集的类型，支持 'classification' 和 'multilabel'。'classification' 为单标签分类场景，'multilabel' 为多标签分类场景。默认值：'classification'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.train import Accuracy
>>>
>>> y_pred = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]), mindspore.float32)
>>> y_true = Tensor(np.array([1, 0, 1]), mindspore.float32)
>>> metric = Accuracy('classification')
>>> metric.clear()
>>> metric.update(y_pred, y_true)
>>> accuracy = metric.eval()
>>> print(accuracy)
0.6666666666666666
```

clear()

内部评估结果清零。

eval()

计算正确率。

返回：

np.float64，计算的正确率结果。

异常：

- **RuntimeError** - 样本量为 0。

update(*inputs)

更新局部变量。计算预测值 y_{pred} 和标签 y 的匹配频率。

对于 'classification'，如果预测的最大值的索引匹配真实的标签，预测正确；对于 'multilabel'，如果预测值与真实标签匹配，预测正确。

参数：

- **inputs** - 预测值 y_{pred} 和真实标签 y ， y_{pred} 和 y 支持 Tensor、list 或 numpy.ndarray 类型。
 - 对于 'classification' 情况， y_{pred} 在大多数情况下由范围 [0,1] 中的浮点数组成，shape 为 (N, C) ，其中 N 是样本数， C 是类别数。 y 由整数值组成，如果是 one_hot 编码格式，shape 是 (N, C) ；如果是类别索引，shape 是 $(N,)$ 。

- 对于 'multilabel' 情况, y_{pred} 和 y 只能是值为 0 或 1 的 one-hot 编码格式, 其中值为 1 的索引表示正类别。 y_{pred} 和 y 的 shape 都是 (N, C) 。

异常:

- **ValueError** - $inputs$ 的数量不等于 2。
- **ValueError** - 当前输入的 y_{pred} 和历史 y_{pred} 类别数不匹配。

7.3.2 mindspore.train.BleuScore

class mindspore.train.BleuScore ($n\_gram=4$, $smooth=False$)

计算 BLEU 分数。BLEU 指的是具有一个或多个引用的机器翻译文本的 metric。

参数:

- **n_gram** (int) - 取值范围为 1~4。默认值: 4。
- **smooth** (bool) - 是否采用平滑计算的方式。默认值: False。

异常:

- **ValueError** - n_gram 的取值范围不在 1~4 之间。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore.train import BleuScore
>>>
>>> candidate_corpus = [['i', 'have', 'a', 'pen', 'on', 'my', 'desk']]
>>> reference_corpus = [['i', 'have', 'a', 'pen', 'in', 'my', 'desk'],
...                     ['there', 'is', 'a', 'pen', 'on', 'the', 'desk']]
>>> metric = BleuScore()
>>> metric.clear()
>>> metric.update(candidate_corpus, reference_corpus)
>>> bleu_score = metric.eval()
>>> print(bleu_score)
0.5946035575013605
```

clear()

重置评估结果。

eval()

计算 BLEU 分数。

返回:

numpy.ndarray, numpy 类型的 BLEU 分数。

异常:

- **RuntimeError** - 调用该方法前没有先调用 update 方法。

update(*inputs)

使用输入的内容更新内部评估结果。

参数:

- **inputs** (iterator) - 输入的元组。第一个输入是机器翻译语料库列表 (candidate_corpus), 第二个输入是引用语料库列表 (reference_corpus)。

异常:

- **ValueError** - 输入参数的数量不等于 2。
- **ValueError** - *candidate_corpus* 的长度与 *reference_corpus* 不同。

7.3.3 mindspore.train.ConfusionMatrix

class mindspore.train.ConfusionMatrix (num_classes, normalize='no_norm', threshold=0.5)

计算混淆矩阵 (Confusion Matrix)，通常用于评估分类模型的性能，包括二分类和多分类场景。

如果只想使用混淆矩阵，请使用该类。如果想计算“PPV”、“TPR”、“TNR”等，请使用 *mindspore.train.ConfusionMatrixMetric* 类。

参数:

- **num_classes** (int) - 数据集中的类别数量。
- **normalize** (str) - 计算 Confusion Matrix 的参数支持四种归一化模式，默认值: "no_norm"。
 - "no_norm": 不使用标准化。
 - "target": 基于目标值的标准化。
 - "prediction": 基于预测值的标准化。
 - "all": 整个矩阵的标准化。
- **threshold** (float) - 阈值，用于与输入 Tensor 进行比较。默认值: 0.5。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import ConfusionMatrix
>>>
>>> x = Tensor(np.array([1, 0, 1, 0]))
>>> y = Tensor(np.array([1, 0, 0, 1]))
>>> metric = ConfusionMatrix(num_classes=2, normalize='no_norm', threshold=0.5)
>>> metric.clear()
>>> metric.update(x, y)
>>> output = metric.eval()
>>> print(output)
[[1. 1.]
 [1. 1.]]
```

clear()

重置评估结果。

eval()

计算混淆矩阵。

返回:

numpy.ndarray, 计算的结果。

update(*inputs)

使用 *y_pred* 和 *y* 更新内部评估结果。

参数:

- ***inputs** (tuple) - 输入 y_{pred} 和 y 。 y_{pred} 和 y 是 *Tensor*、列表或数组。 y_{pred} 是预测值, y 是真实值, y_{pred} 的 shape 是 $(N, C, \dots)$ 或 $(N, \dots)$, y 的 shape 是 $(N, \dots)$ 。

异常:

- **ValueError** - 输入参数的数量不等于 2。
- **ValueError** - 如果预测值 y_{pred} 和真实值 y 的维度不一致。

7.3.4 mindspore.train.ConfusionMatrixMetric

```
class mindspore.train.ConfusionMatrixMetric (skip_channel=True, metric_name='sensitivity',
                                             calculation_method=False, decrease='mean')
```

计算与混淆矩阵相关的度量。

该计算基于全尺度张量, 并收集批处理平均值、类通道数和迭代数。此函数支持计算参数 `metric_name` 的描述中列出的所有度量名称。

- 如果要使用混淆矩阵计算, 如“PPV”、“TPR”、“TNR”, 请使用此类。
- 如果只想计算混淆矩阵, 请使用 `mindspore.train.ConfusionMatrix`。

参数:

- **skip_channel** (bool) - 是否跳过预测输出的第一个通道的度量计算。默认值: `True`。
- **metric_name** (str) - 建议采用如下指标。当然, 也可以为这些指标设置通用别名。取值范围: `["sensitivity", "specificity", "precision", "negative predictive value", "miss rate", "fall out", "false discovery rate", "false omission rate", "prevalence threshold", "threat score", "accuracy", "balanced accuracy", "f1 score", "matthews correlation coefficient", "fowlkes mallows index", "informedness", "markedness"]`。默认值: `"sensitivity"`。
- **calculation_method** (bool) - 如果为 `True`, 则计算每个样本的度量值。如果为 `False`, 则累积所有样本的混淆矩阵。对于分类任务, `calculation_method` 应为 `False`。默认值: `False`。
- **decrease** (str) - 定义减少一批数据计算结果的模式。仅当 `calculation_method` 为 `True` 时, 才生效。取值范围: `["none", "mean", "sum", "mean_batch", "sum_batch", "mean_channel", "sum_channel"]`。默认值: `"mean"`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import ConfusionMatrixMetric
>>>
>>> metric = ConfusionMatrixMetric(skip_channel=True, metric_name="tpr",
...                               calculation_method=False, decrease="mean")
>>> metric.clear()
>>> x = Tensor(np.array([[0], [1]], [[1], [0]]))
>>> y = Tensor(np.array([[0], [1]], [[0], [1]]))
>>> metric.update(x, y)
>>> avg_output = metric.eval()
>>> print(avg_output)
[0.5]
```

clear()
重置评估结果。

eval()

计算混淆矩阵度量。

返回:

numpy.ndarray, 计算的结果。

update(*inputs)

使用预测值和目标值更新状态。

参数:

- **inputs** (tuple) - y_pred 和 y 。 y_pred 和 y 是 *Tensor*、列表或数组。
 - **y_pred** (ndarray) - 待计算的输入数据。格式必须为 one-hot, 且第一个维度是 batch。 y_pred 的 shape 是 $(N, C, \dots)$ 或 $(N, \dots)$ 。对于分类任务, y_pred 的 shape 应为 (B, N) , 其中 N 大于 1。对于分割任务, shape 应为 (B, N, H, W) 或 (B, N, H, W, D) 。
 - **y** (ndarray) - 计算度量值的真实值。格式必须为 one-hot, 且第一个维度是 batch。 y 的 shape 是 $(N, C, \dots)$ 。

异常:

- **ValueError** - 输入参数的数量不等于 2。

7.3.5 mindspore.train.CosineSimilarity

class mindspore.train.CosineSimilarity (*similarity='cosine', reduction='none', zero_diagonal=True*)

计算余弦相似度。

参数:

- **similarity** (str) - 计算逻辑。 'cosine' 表示相似度计算逻辑, 'dot' 表示矩阵点乘计算逻辑。默认值: 'cosine'。
- **reduction** (str) - 规约计算方式。支持 'none'、'sum' 或 'mean'。默认值: 'none'。
- **zero_diagonal** (bool) - 如果为 True, 则对角线将设置为零。默认值: True。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore.train import CosineSimilarity
>>>
>>> test_data = np.array([[1, 3, 4, 7], [2, 4, 2, 5], [3, 1, 5, 8]])
>>> metric = CosineSimilarity()
>>> metric.clear()
>>> metric.update(test_data)
>>> square_matrix = metric.eval()
>>> print(square_matrix)
[[0.  0.94025615  0.95162452]
 [0.94025615  0.  0.86146098]
 [0.95162452  0.86146098  0.]]
```

clear()

重置评估结果。

eval()

计算 Cosine Similarity 矩阵。

返回:

numpy.ndarray, 相似度矩阵。

异常:

- **RuntimeError** - 如果没有先调用 update 方法。

update(*inputs)

使用 *inputs* 更新内部评估结果。

参数:

- **inputs** (Union[Tensor, list, numpy.ndarray]) - 输入的矩阵, 包含 *y_pred* 和 *y*。

7.3.6 mindspore.train.Dice

class mindspore.train.Dice (*smooth=1e-5*)

集合相似性度量。

用于计算两个样本之间的相似性。当分割结果最好时, Dice 系数的值为 1; 当分割结果最差时, Dice 系数的值为 0。Dice 系数表示预测值与真实值交集同预测值和真实值并集之间的比值。

$$dice = \frac{2 * (pred \cap true)}{pred \cup true}$$

参数:

- **smooth** (float) - 在计算过程中直接添加到分母里做加法, 用于提高数值稳定性, 取值需大于 0。默认值: 1e-5。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import Dice
>>>
>>> x = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]))
>>> y = Tensor(np.array([[0, 1], [1, 0], [0, 1]]))
>>> metric = Dice(smooth=1e-5)
>>> metric.clear()
>>> metric.update(x, y)
>>> dice = metric.eval()
>>> print(dice)
0.20467791371802546
```

clear()

重置评估结果。

eval()

计算 Dice 系数。

返回:

float, 计算的结果。

异常:

- **RuntimeError** - 样本数为 0。

update (*inputs)

更新内部评估结果 y_{pred} 和 y 。

参数:

- **inputs** (tuple) - 输入 y_{pred} 和 y 。 y_{pred} 和 y 是 tensor、列表或 `numpy.ndarray`。 y_{pred} 是预测值, y 是真实值。 y_{pred} 和 y 的 shape 都是 $(N, \dots)$ 。

异常:

- **ValueError** - 输入参数的数量不等于 2。
- **ValueError** - 如果预测值和标签 shape 不一致。

7.3.7 mindspore.train.F1

class `mindspore.train.F1`

计算 F1 评分。F1 是 Fbeta 的特殊情况, 即 β 为 1。有关更多详细信息, 请参阅类 `mindspore.train.Fbeta`。

$$F_1 = \frac{2 \cdot \text{true\_positive}}{2 \cdot \text{true\_positive} + \text{false\_negative} + \text{false\_positive}}$$

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import F1
>>>
>>> x = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]))
>>> y = Tensor(np.array([1, 0, 1]))
>>> metric = F1()
>>> metric.update(x, y)
>>> result = metric.eval()
>>> print(result)
[0.66666667 0.66666667]
```

7.3.8 mindspore.train.Fbeta

class `mindspore.train.Fbeta` (β)

计算 Fbeta 评分。

Fbeta 评分是精度 (Precision) 和召回率 (Recall) 的加权平均值。

$$F_\beta = \frac{(1 + \beta^2) \cdot \text{true\_positive}}{(1 + \beta^2) \cdot \text{true\_positive} + \beta^2 \cdot \text{false\_negative} + \text{false\_positive}}$$

参数:

- **beta** (`Union[float, int]`) - F-measure 中的 β 系数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import Fbeta
>>>
>>> x = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]))
>>> y = Tensor(np.array([1, 0, 1]))
>>> metric = Fbeta(1)
>>> metric.clear()
>>> metric.update(x, y)
>>> fbeta = metric.eval()
>>> print(fbeta)
[0.66666667 0.66666667]
```

clear()

内部评估结果清零。

eval (*average=False*)

计算 Fbeta 结果。

参数:

- **average** (bool) - 是否计算 Fbeta 平均值。默认值: False。

返回:

numpy.ndarray 或 numpy.float64, 计算的 Fbeta 评分结果。

update (**inputs*)

使用预测值 *y_pred* 和真实标签 *y* 更新内部评估结果。

参数:

- **inputs** - *y_pred* 和 *y*。 *y_pred* 和 *y* 支持 Tensor、list 或 numpy.ndarray 类型。通常情况下, *y_pred* 是 [0,1] 之间的浮点数列表, shape 为 (N,C), 其中 N 是样本数, C 是类别数。 *y* 是整数, 如果使用 one-hot 编码, 则 shape 为 (N,C); 如果使用类别索引, shape 是 (N,)。

异常:

- **ValueError** - 当前输入的 *y_pred* 和历史 *y_pred* 类别数不匹配。
- **ValueError** - 预测值和真实值包含的类别不同。

7.3.9 mindspore.train.HausdorffDistance

class mindspore.train.HausdorffDistance (*distance_metric='euclidean', percentile=None, directed=False, crop=True*)

计算 Hausdorff 距离。Hausdorff 距离是两个点集之间, 两点的最小距离的最大值, 度量了两个点集间的最大不匹配程度。

给定两个集合 A 和 B, A 和 B 之间的 Hausdorff 距离定义如下:

$$\begin{aligned} H(A, B) &= \max[h(A, B), h(B, A)] \\ h(A, B) &= \max_{a \in A} \{\min_{b \in B} \|a - b\|\} \\ h(B, A) &= \max_{b \in B} \{\min_{a \in A} \|b - a\|\} \end{aligned}$$

其中 $h(A, B)$ 表示, 对 A 中的每个点 a 找到 B 集合里的最近点, 这些最短距离的最大值为从 A 到 B 的单向 Hausdorff 距离。同理, $h(B, A)$ 为集合 B 到集合 A 中最近点的最大距离。Hausdorff 距离是有方向性的, 通常情况下 $h(A, B)$ 不等于 $h(B, A)$ 。 $H(A, B)$ 为双向 Hausdorff 距离。

参数:

- **distance_metric** (str) - 支持如下三种距离计算方法: "euclidean" (欧式距离)、"chessboard" (棋盘距离、切比雪夫距离) 或 "taxicab" (出租车距离、曼哈顿距离)。默认值: "euclidean"。
- **percentile** (float) - 0 到 100 之间的浮点数。指定最终返回的 Hausdorff 距离的百分位数。默认值: None。
- **directed** (bool) - 如果为 True, 为单向 Hausdorff 距离, 只计算 $h(y\_pred, y)$ 距离; 如果为 False, 为双向 Hausdorff 距离, 计算 $\max(h(y\_pred, y), h(y, y\_pred))$ 。默认值: False。
- **crop** (bool) - 是否裁剪输入图像, 仅保留目标区域。为了保证 y_pred 和 y 的 shape 匹配, 使用 $(y\_pred | y)$, 即两图像的并集来确定 bounding box。默认值: True。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import HausdorffDistance
>>>
>>> x = Tensor(np.array([[3, 0, 1], [1, 3, 0], [1, 0, 2]]))
>>> y = Tensor(np.array([[0, 2, 1], [1, 2, 1], [0, 0, 1]]))
>>> metric = HausdorffDistance()
>>> metric.clear()
>>> metric.update(x, y, 0)
>>> mean_average_distance = metric.eval()
>>> print(mean_average_distance)
1.4142135623730951
```

clear()

内部评估结果清零。

eval()

计算定向或非定向 Hausdorff 距离。

返回:

numpy.float64, 计算得到的 Hausdorff 距离。

异常:

- **RuntimeError** - 如果没有先调用 update 方法。

update(*inputs)

使用 y_pred 、 y 和 $label\_idx$ 更新内部评估结果。

参数:

- **inputs** - y_pred 、 y 和 $label\_idx$ 。 y_pred 和 y 为 Tensor、list 或 numpy.ndarray。 y_pred 是预测的二值图像, y 是实际的二值图像。 $label\_idx$ 的数据类型为 int 或 float, 表示像素点的类别值。

异常:

- **ValueError** - 输入的数量不等于 3。
- **TypeError** - $label\_idx$ 的数据类型不是 int 或 float。
- **ValueError** - $label\_idx$ 的值不在 y_pred 或 y 中。

- **ValueError** - y_{pred} 和 y 的 shape 不同。

7.3.10 mindspore.train.Loss

class mindspore.train.Loss

计算 loss 的平均值。如果每 n 次迭代调用一次 *update* 方法，则计算结果为：

$$loss = \frac{\sum_{k=1}^n loss_k}{n}$$

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.train import Loss
>>>
>>> x = Tensor(np.array(0.2), mindspore.float32)
>>> loss = Loss()
>>> loss.clear()
>>> loss.update(x)
>>> result = loss.eval()
>>> print(result)
0.20000000298023224
```

clear()

内部评估结果清零。

eval()

计算 loss 的平均值。

返回：

float, loss 的平均值。

异常：

- **RuntimeError** - 样本总数为 0。

update(*inputs)

更新内部评估结果。

参数：

- **inputs** - 输入只包含一个元素，且该元素为 loss。loss 的维度必须为 0 或 1。

异常：

- **ValueError** - *inputs* 的长度不为 1。
- **ValueError** - *inputs* 的维度不为 0 或 1。

7.3.11 mindspore.train.MAE

class mindspore.train.MAE

计算平均绝对误差 MAE (Mean Absolute Error)。

计算输入 y_{pred} 和目标 y 各元素之间的平均绝对误差。

$$MAE = \frac{\sum_{i=1}^n \|y_{pred_i} - y_i\|}{n}$$

这里, n 是 batch size。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.train import MAE
>>>
>>> x = Tensor(np.array([0.1, 0.2, 0.6, 0.9]), mindspore.float32)
>>> y = Tensor(np.array([0.1, 0.25, 0.7, 0.9]), mindspore.float32)
>>> error = MAE()
>>> error.clear()
>>> error.update(x, y)
>>> result = error.eval()
>>> print(result)
0.037499990314245224
```

clear()

内部评估结果清零。

eval()

计算平均绝对误差 (MAE)。

返回:

numpy.float64, 计算的 MAE 的结果。

异常:

- **RuntimeError** - 样本总数为 0。

update(*inputs)

使用预测值 y_{pred} 和真实值 y 更新局部变量。

参数:

- **inputs** - 输入 y_{pred} 和 y 来计算 MAE, 其中 y_{pred} 和 y 的 shape 都是 N-D, 它们的 shape 相同。

异常:

- **ValueError** - *inputs* 的数量不等于 2。

7.3.12 mindspore.train.MeanSurfaceDistance

class mindspore.train.MeanSurfaceDistance (symmetric=False, distance_metric='euclidean')

计算从 y_{pred} 到 y 的平均表面距离。通常情况下，用来衡量分割任务中，预测情况和真实情况之间的差异度。

给定两个集合 A 和 B ， $S(A)$ 表示 A 的表面像素，任意 v 到 $S(A)$ 的最短距离定义为：

$$\text{dis}(v, S(A)) = \min_{s_A \in S(A)} \|v - s_A\|$$

从集合 B 到集合 A 的平均表面距离（Average Surface Distance）为：

$$\text{AvgSurDis}(B \rightarrow A) = \frac{\sum_{s_B \in S(B)} \text{dis}(s_B, S(A))}{|S(B)|}$$

其中， $\|*\|$ 表示距离度量， $|*|$ 表示元素的数量。

从集合 B 到集合 A ，以及从集合 A 到集合 B 的表面距离平均值为：

$$\text{MeanSurDis}(A \leftrightarrow B) = \frac{\sum_{s_A \in S(A)} \text{dis}(s_A, S(B)) + \sum_{s_B \in S(B)} \text{dis}(s_B, S(A))}{|S(A)| + |S(B)|}$$

参数：

- **distance_metric** (str) - 支持如下三种距离计算方法："euclidean"（欧式距离）、"chessboard"（棋盘距离、切比雪夫距离）或 "taxicab"（出租车距离、曼哈顿距离）。默认值："euclidean"。
- **symmetric** (bool) - 是否计算 y_{pred} 和 y 之间的对称平均平面距离。如果为 False，计算方式为 $\text{AvgSurDis}(y_{pred} \rightarrow y)$ ；如果为 True，计算方式为 $\text{MeanSurDis}(y_{pred} \leftrightarrow y)$ 。默认值：False。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import MeanSurfaceDistance
>>> x = Tensor(np.array([[3, 0, 1], [1, 3, 0], [1, 0, 2]]))
>>> y = Tensor(np.array([[0, 2, 1], [1, 2, 1], [0, 0, 1]]))
>>> metric = MeanSurfaceDistance(symmetric=False, distance_metric="euclidean")
>>> metric.clear()
>>> metric.update(x, y, 0)
>>> mean_average_distance = metric.eval()
>>> print(mean_average_distance)
0.8047378541243649
```

clear()

内部评估结果清零。

eval()

计算平均表面距离。

返回：

numpy.float64，计算得到的平均表面距离值。

异常：

- **RuntimeError** - 如果没有先调用 update 方法。

update (*inputs)

使用 *y_pred*、*y* 和 *label_idx* 更新内部评估结果。

参数:

- **inputs** - *y_pred*、*y* 和 *label_idx*。*y_pred* 和 *y* 为 Tensor、list 或 numpy.ndarray。*y_pred* 是预测的二值图像。*y* 是实际的二值图像。*label_idx* 数据类型为 int 或 float，表示像素点的类别值。

异常:

- **ValueError** - 输入的数量不等于 3。
- **TypeError** - *label_idx* 的数据类型不是 int 或 float。
- **ValueError** - *label_idx* 的值不在 *y_pred* 或 *y* 中。
- **ValueError** - *y_pred* 和 *y* 的 shape 不同。

7.3.13 mindspore.train.Metric

class mindspore.train.Metric

用于计算评估指标的基类。

在计算评估指标时，需要调用 *clear*、*update* 和 *eval* 三个方法，在继承该类自定义评估指标时，也需要实现这三个方法。其中，*update* 用于计算中间过程的内部结果，*eval* 用于计算最终评估结果，*clear* 用于重置中间结果。

请勿直接使用该类，需使用子类如 *mindspore.train.MAE*、*mindspore.train.Recall* 等。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>>
>>> class MyMAE(ms.train.Metric):
...     def __init__(self):
...         super(MyMAE, self).__init__()
...         self.clear()
...
...     def clear(self):
...         self._abs_error_sum = 0
...         self._samples_num = 0
...
...     def update(self, *inputs):
...         y_pred = inputs[0].asnumpy()
...         y = inputs[1].asnumpy()
...         abs_error_sum = np.abs(y - y_pred)
...         self._abs_error_sum += abs_error_sum.sum()
...         self._samples_num += y.shape[0]
...
...     def eval(self):
...         return self._abs_error_sum / self._samples_num
>>>
>>> x = ms.Tensor(np.array([[0.1, 0.2, 0.6, 0.9], [0.1, 0.2, 0.6, 0.9]]), ms.float32)
>>> y = ms.Tensor(np.array([[0.1, 0.1, 0.1, 0.1], [0.1, 0.1, 0.1, 0.1]]), ms.float32)
>>> y2 = ms.Tensor(np.array([[0.1, 0.25, 0.7, 0.9], [0.1, 0.25, 0.7, 0.9]]), ms.float32)
>>> metric = MyMAE().set_indexes([0, 2])
```

(下页继续)

(续上页)

```
>>> metric.clear()
>>> # indexes is [0, 2], using x as logits, y2 as label.
>>> metric.update(x, y, y2)
>>> accuracy = metric.eval()
>>> print(accuracy)
1.399999976158142
>>> print(metric.indexes)
[0, 2]
```

abstract clear()
清除内部评估结果。

说明：所有子类都必须重写此接口。

abstract eval()
计算最终评估结果。

说明：所有子类都必须重写此接口。

property indexes

获取当前的 *indexes* 值。默认为 None，调用 *set_indexes* 方法可修改 *indexes* 值。

set_indexes(indexes)

该接口用于重排 *update* 的输入。

给定 (label0, label1, logits) 作为 *update* 的输入，将 *indexes* 设置为 [2, 1]，则最终使用 (logits, label1) 作为 *update* 的真实输入。

说明：在继承该类自定义评估函数时，需要用装饰器 `mindspore.train.rearrange_inputs()` 修饰 *update* 方法，否则配置的 *indexes* 值不生效。

参数：

- **indexes** (List(int)) - logits 和标签的目标顺序。

输出：

`Metric`，类实例本身。

异常：

- **ValueError** - 如果输入的 *index* 类型不是 list，或其元素类型不全为 int。

abstract update(*inputs)
更新内部评估结果。

说明：所有子类都必须重写此接口。

参数：

- **inputs** - 可变长度输入参数列表。通常是预测值和对应的真实标签。

7.3.14 mindspore.train.MSE

class mindspore.train.MSE

测量均方差 MSE (Mean Squared Error)。

计算输入 y_{pred} 和目标 y 各元素之间的平均平方误差。

$$\text{MSE}(y_{pred}, y) = \frac{\sum_{i=1}^n (y_{pred_i} - y_i)^2}{n}$$

其中, n 为 batch size。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.train import MSE
>>>
>>> x = Tensor(np.array([0.1, 0.2, 0.6, 0.9]), mindspore.float32)
>>> y = Tensor(np.array([0.1, 0.25, 0.5, 0.9]), mindspore.float32)
>>> error = MSE()
>>> error.clear()
>>> error.update(x, y)
>>> result = error.eval()
>>> print(result)
0.0031250009778887033
```

clear()

清除内部评估结果。

eval()

计算均方差 (MSE)。

返回:

numpy.float64, 计算的 MSE 的结果。

异常:

- **RuntimeError** - 样本数为 0。

update(*inputs)

使用预测值 y_{pred} 和真实值 y 更新局部变量。

参数:

- **inputs** - 输入 y_{pred} 和 y 用于计算 MSE。其中 y_{pred} 和 y 的 shape 都为 N-D, 它们的 shape 相同。

异常:

- **ValueError** - *inputs* 的数量不等于 2。

7.3.15 mindspore.train.OcclusionSensitivity

class mindspore.train.OcclusionSensitivity (pad_val=0.0, margin=2, n_batch=128, b_box=None)

用于计算神经网络对给定图像的遮挡灵敏度 (Occlusion Sensitivity)，表示了图像的哪些部分对神经网络的分类决策最重要。

遮挡灵敏度是指神经网络对图像的类别预测概率如何随着图像被遮挡部分的变化而变化。遮挡灵敏度值越高，意味着遮挡后模型对类别预测的概率值下降越大，说明遮挡区域在神经网络的分类决策过程中越重要。

参数：

- **pad_val** (float) - 图像中被遮挡部分的填充值。默认值：0.0。
- **margin** (Union[int, Sequence]) - 在要遮挡的像素点周围设置的长方体/立方体，表示被遮挡多少个像素点。默认值：2。
- **n_batch** (int) - 一个 batch 中样本的数量。默认值：128。
- **b_box** (Sequence) - 执行分析的目标区域的边界框 (Bounding box)，其大小与输出图像的大小相匹配。如果没有设置此入参，Bounding box 将与输入图像的大小相同；如果设置了此入参，输入图像将被裁剪为此大小，此设置值应形如：[min1, max1, min2, max2, ...]，分别对应除 batch size 外各维度的最大最小值。默认值：None。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import nn, Tensor
>>> from mindspore.train import OcclusionSensitivity
>>>
>>> class DenseNet(nn.Cell):
...     def __init__(self):
...         super(DenseNet, self).__init__()
...         w = np.array([[0.1, 0.8, 0.1, 0.1], [1, 1, 1, 1]]).astype(np.float32)
...         b = np.array([0.3, 0.6]).astype(np.float32)
...         self.dense = nn.Dense(4, 2, weight_init=Tensor(w), bias_init=Tensor(b))
...
...     def construct(self, x):
...         return self.dense(x)
>>>
>>> model = DenseNet()
>>> test_data = np.array([[0.1, 0.2, 0.3, 0.4]]).astype(np.float32)
>>> label = np.array(1).astype(np.int32)
>>> metric = OcclusionSensitivity()
>>> metric.clear()
>>> metric.update(model, test_data, label)
>>> score = metric.eval()
>>> print(score)
[0.29999995 0.6 1. 0.9]
```

clear()

内部评估结果清零。

eval()

计算遮挡灵敏度。

返回：

numpy ndarray。计算得到的遮挡灵敏度值。

异常:

- **RuntimeError** - 如果没有先调用 `update` 方法, 则会报错。

update (*inputs)

更新 inputs, 包括 *model*、*y_pred* 和 *label*。

参数:

- **inputs** - *y_pred* 和 *label* 为 Tensor、list 或 numpy.ndarray。*y_pred* 是要测试的图像, 一般为 2D 或 3D; *label* 是用于检测神经网络预测值变化的类别标签, 通常情况下为真实标签。*model* 为神经网络模型。

异常:

- **ValueError** - 输入数量不是 3。
- **RuntimeError** - *y_pred.shape[0]* 不是 1。
- **RuntimeError** - 标签数量与 batch 数量不同。

7.3.16 mindspore.train.Perplexity

class mindspore.train.Perplexity (ignore_label=None)

计算困惑度 (perplexity)。困惑度是衡量一个概率分布或语言模型好坏的标准。低困惑度表明语言模型可以很好地预测样本。计算方式如下:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

其中 *w* 代表语料库中的单词。根号内是句子概率的倒数, 句子越好 (概率大), 困惑度越小。

参数:

- **ignore_label** (Union[int, None]) - 计数时要忽略的无效标签的索引。如果设置为 None, 它将包括所有条目。
默认值: None。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import Perplexity
>>> x = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]))
>>> y = Tensor(np.array([1, 0, 1]))
>>> metric = Perplexity(ignore_label=None)
>>> metric.clear()
>>> metric.update(x, y)
>>> perplexity = metric.eval()
>>> print(perplexity)
2.231443166940565
```

clear()

内部评估结果清零。

eval()

返回当前评估结果。

返回:

numpy.float64, 计算得到的困惑度结果。

异常:

- **RuntimeError** - 样本量为 0。

update (*inputs)

使用 *preds* 和 *labels* 更新内部评估结果。

参数:

- **inputs** - 输入 *preds* 和 *labels*。 *preds* 和 *labels* 是 Tensor、list 或 numpy.ndarray。 *preds* 是预测值, *labels* 是数据的标签。 *preds* 和 *labels* 的 shape 都是 (N, C) 。

异常:

- **ValueError** - 输入数量不是 2。
- **RuntimeError** - 预测值和标签的长度不同。
- **RuntimeError** - 预测值和标签的 shape 不同。

7.3.17 mindspore.train.Precision

class mindspore.train.Precision (eval_type='classification')

计算数据分类的精度, 包括单标签场景和多标签场景。

此函数创建两个局部变量 true_positive 和 false_positive, 用于计算精度。计算方式如下:

$$\text{precision} = \frac{\text{true\_positive}}{\text{true\_positive} + \text{false\_positive}}$$

参数:

- **eval_type** (str) - 支持 'classification' 和 'multilabel'。具体作用见下方的 update 方法。默认值: 'classification'。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import Precision
>>>
>>> x = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]))
>>> y = Tensor(np.array([1, 0, 1]))
>>> metric = Precision('classification')
>>> metric.clear()
>>> metric.update(x, y)
>>> precision = metric.eval()
>>> print(precision)
[0.5 1. ]
```

clear ()

内部评估结果清零。

eval (average=False)

计算精度。

参数:

- **average** (bool) - 指定是否计算平均精度。默认值: `False`。

返回:

`numpy.float64`, 计算结果。

update (*inputs)

使用预测值 `y_pred` 和真实标签 `y` 更新局部变量。在多标签情况下, `y` 和 `y_pred` 的元素必须为 0 或 1。

参数:

- **inputs** - 输入 `y_pred` 和 `y`。 `y_pred` 和 `y` 支持 `Tensor`、`list` 或 `numpy.ndarray` 类型。
 - 对于 'classification' 情况, `y_pred` 在大多数情况下由范围 `[0, 1]` 中的浮点数组成, `shape` 为 `(N, C)`, 其中 `N` 是样本数, `C` 是类别数。 `y` 由整数值组成, 如果是 `one-hot` 编码格式, `shape` 是 `(N, C)`; 如果是类别索引, `shape` 是 `(N,)`。
 - 对于 'multilabel' 情况, `y_pred` 和 `y` 只能是值为 0 或 1 的 `one-hot` 编码格式, 其中值为 1 的索引表示正类别。 `y_pred` 和 `y` 的 `shape` 都是 `(N, C)`。

异常:

- **ValueError** - inputs 数量不是 2。

7.3.18 mindspore.train.Recall

class mindspore.train.Recall (eval_type='classification')

计算数据分类的召回率, 包括单标签场景和多标签场景。

Recall 类创建两个局部变量 `true_positive` 和 `false_negative` 用于计算召回率。计算方式为:

$$\text{recall} = \frac{\text{true\_positive}}{\text{true\_positive} + \text{false\_negative}}$$

说明: 在多标签情况下, `y` 和 `y_pred` 的元素必须为 0 或 1。

参数:

- **eval_type** (str) - 支持 'classification' (单标签分类) 和 'multilabel' (多标签分类)。默认值: 'classification'。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import Recall
>>>
>>> x = Tensor(np.array([[0.2, 0.5], [0.3, 0.1], [0.9, 0.6]]))
>>> y = Tensor(np.array([1, 0, 1]))
>>> metric = Recall('classification')
>>> metric.clear()
>>> metric.update(x, y)
```

(下页继续)

(续上页)

```
>>> recall = metric.eval()
>>> print(recall)
[1. 0.5]
```

clear()

内部评估结果清零。

eval (*average=False*)

计算召回率。

参数:

- **average** (bool) - 指定是否计算平均召回率。默认值: `False`。

返回:

`numpy.float64`, 计算结果。

update (**inputs*)

使用预测值 `y_pred` 和真实标签 `y` 更新局部变量。

参数:

- **inputs** - 输入 `y_pred` 和 `y`。 `y_pred` 和 `y` 支持 `Tensor`、`list` 或 `numpy.ndarray` 类型。
 - 对于 'classification' 场景, `y_pred` 在大多数情况下由范围 `[0, 1]` 中的浮点数组成, `shape` 为 `(N, C)`, 其中 `N` 是样本数, `C` 是类别数。 `y` 由整数值组成, 如果是 `one_hot` 编码格式, `shape` 是 `(N, C)`; 如果是类别索引, `shape` 是 `(N,)`。
 - 对于 'multilabel' 场景, `y_pred` 和 `y` 只能是值为 0 或 1 的 `one-hot` 编码格式, 其中值为 1 的索引表示正类别。 `y_pred` 和 `y` 的 `shape` 都是 `(N, C)`。

异常:

- **ValueError** - `inputs` 数量不是 2。

7.3.19 mindspore.train.ROC

class `mindspore.train.ROC` (*class_num=None, pos_label=None*)

计算 ROC 曲线。适用于求解二分类和多分类问题。在多分类的情况下, 将基于 `one-vs-the-rest` 的方法进行计算。

参数:

- **class_num** (int) - 类别数。对于二分类问题, 此入参可以不设置。默认值: `None`。
- **pos_label** (int) - 正类的类别值。二分类问题中, 用户可以自行设置正类的类别值为其他值, 如果不设置此入参即 `pos_label` 为 `None` 时, 正类的类别值默认为 1。多分类问题中, 用户不应设置此参数, 因为它将在 `[0, num_classes-1]` 范围内迭代更改。默认值: `None`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import ROC
>>>
>>> # 1) binary classification example
```

(下页继续)

(续上页)

```

>>> x = Tensor(np.array([0.28, 0.55, 0.15, 0.05]))
>>> y = Tensor(np.array([0, 1, 2, 3]))
>>> metric = ROC(pos_label=2)
>>> metric.clear()
>>> metric.update(x, y)
>>> fpr, tpr, thresholds = metric.eval()
>>> print(fpr)
[0.          0.33333333 0.66666667 0.66666667 1.          ]
>>> print(tpr)
[0. 0. 0. 1. 1.]
>>> print(thresholds)
[1.55 0.55 0.28 0.15 0.05]
>>>
>>> # 2) multiclass classification example
>>> x = Tensor(np.array([[0.28, 0.55, 0.15, 0.05], [0.10, 0.20, 0.05, 0.05], [0.20, 0.05, 0.
↪15, 0.05],
...                      [0.05, 0.05, 0.05, 0.75]]))
>>> y = Tensor(np.array([0, 1, 2, 3]))
>>> metric = ROC(class_num=4)
>>> metric.clear()
>>> metric.update(x, y)
>>> fpr, tpr, thresholds = metric.eval()
>>> print(fpr)
[array([0., 0., 0.33333333, 0.66666667, 1.]), array([0., 0.33333333, 0.33333333, 1.]),
array([0., 0.33333333, 1.]), array([0., 0., 1.])]
>>> print(tpr)
[array([0., 1., 1., 1., 1.]), array([0., 0., 1., 1.]), array([0., 1., 1.]), array([0., 1., 1.
↪.])]
>>> print(thresholds)
[array([1.28, 0.28, 0.2, 0.1, 0.05]), array([1.55, 0.55, 0.2, 0.05]), array([1.15, 0.15, 0.
↪05]),
array([1.75, 0.75, 0.05])]

```

clear()

内部评估结果清零。

eval()

计算 ROC 曲线。

返回：

tuple, 由 *fpr*、*tpr* 和 *thresholds* 组成。

- **fpr** (np.array) - 假正率 (false positive rate)。二分类情况下, 返回不同阈值下的 fpr; 多分类情况下, 则返回 fpr 的列表, 列表的每个元素代表一个类别。
- **tpr** (np.array) - 真正率 (true positive rate)。二分类情况下, 返回不同阈值下的 tpr; 多分类情况下, 则返回 tpr 的列表, 列表的每个元素代表一个类别。
- **thresholds** (np.array) - 用于计算假正率和真正率的阈值。

异常：

- **RuntimeError** - 如果没有先调用 `update` 方法, 则会报错。

update(*inputs)

使用 *y_pred* 和 *y* 更新内部评估结果。

参数:

- **inputs** - 输入 y_pred 和 y 。 y_pred 和 y 是 Tensor、list 或 `numpy.ndarray`。 y_pred 一般情况下是范围为 $[0, 1]$ 的浮点数列表, `shape` 为 (N, C) , 其中 N 是用例数, C 是类别数。 y 为整数, 如果为 one-hot 格式, `shape` 为 (N, C) ; 如果是类别索引, `shape` 为 $(N,)$ 。

7.3.20 mindspore.train.RootMeanSquareDistance

class mindspore.train.RootMeanSquareDistance (*symmetric=False, distance_metric='euclidean'*)

计算从 y_pred 到 y 的均方根表面距离。

给定两个集合 A 和 B , $S(A)$ 表示 A 的表面像素, 任意 v 到 $S(A)$ 的最短距离定义为:

$$\text{dis}(v, S(A)) = \min_{s_A \in S(A)} \|v - s_A\|$$

从集合 B 到集合 A 的均方根表面距离 (Root Mean Square Surface Distance) 为:

$$\text{RmsSurDis}(B \rightarrow A) = \sqrt{\frac{\sum_{s_B \in S(B)} \text{dis}^2(s_B, S(A))}{|S(B)|}}$$

其中, $\|*\|$ 表示距离度量, $|*|$ 表示元素的数量。

从集合 B 到集合 A , 以及从集合 A 到集合 B 的表面距离平均值为:

$$\text{RmsSurDis}(A \leftrightarrow B) = \sqrt{\frac{\sum_{s_A \in S(A)} \text{dis}(s_A, S(B))^2 + \sum_{s_B \in S(B)} \text{dis}(s_B, S(A))^2}{|S(A)| + |S(B)|}}$$

参数:

- **distance_metric** (str) - 支持如下三种距离计算方法: "euclidean" (欧式距离)、"chessboard" (棋盘距离、切比雪夫距离) 或 "taxicab" (出租车距离、曼哈顿距离)。默认值: "euclidean"。
- **symmetric** (bool) - 是否计算 y_pred 和 y 之间的对称平均平面距离。如果为 False, 计算方式为 $\text{RmsSurDis}(y\_pred, y)$; 如果为 True, 计算方式为 $\text{RmsSurDis}(y\_pred \leftrightarrow y)$ 。默认值: False。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import RootMeanSquareDistance
>>>
>>> x = Tensor(np.array([[3, 0, 1], [1, 3, 0], [1, 0, 2]]))
>>> y = Tensor(np.array([[0, 2, 1], [1, 2, 1], [0, 0, 1]]))
>>> metric = RootMeanSquareDistance(symmetric=False, distance_metric="euclidean")
>>> metric.clear()
>>> metric.update(x, y, 0)
>>> root_mean_square_distance = metric.eval()
>>> print(root_mean_square_distance)
1.0000000000000002
```

clear()

内部评估结果清零。

eval()

计算均方根表面距离。

返回:

`numpy.float64`, 计算得到的均方根表面距离值。

异常:

- **RuntimeError** - 如果没有先调用 `update` 方法, 则会报错。

update (*inputs)

使用 `y_pred`、`y` 和 `label_idx` 更新内部评估结果。

参数:

- **inputs** - `y_pred`、`y` 和 `label_idx`。`y_pred` 和 `y` 为 `Tensor`、`list` 或 `numpy.ndarray`。`y_pred` 是预测的二值图像。`y` 是实际的二值图像。`label_idx` 数据类型为 `int` 或 `float`, 表示像素点的类别值。

异常:

- **ValueError** - 输入的数量不等于 3。
- **TypeError** - `label_idx` 的数据类型不是 `int` 或 `float`。
- **ValueError** - `label_idx` 的值不在 `y_pred` 或 `y` 中。
- **ValueError** - `y_pred` 和 `y` 的 `shape` 不同。

7.3.21 mindspore.train.Top1CategoricalAccuracy

class `mindspore.train.Top1CategoricalAccuracy`

计算 top-1 分类正确率。此类是 `TopKCategoricalAccuracy` 的特殊类。有关更多详细信息, 请参阅 `mindspore.train.TopKCategoricalAccuracy`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.train import Top1CategoricalAccuracy
>>>
>>> x = Tensor(np.array([[0.2, 0.5, 0.3, 0.6, 0.2], [0.1, 0.35, 0.5, 0.2, 0.],
...                      [0.9, 0.6, 0.2, 0.01, 0.3]]), mindspore.float32)
>>> y = Tensor(np.array([2, 0, 1]), mindspore.float32)
>>> topk = Top1CategoricalAccuracy()
>>> topk.clear()
>>> topk.update(x, y)
>>> output = topk.eval()
>>> print(output)
0.0
```

7.3.22 mindspore.train.Top5CategoricalAccuracy

class mindspore.train.Top5CategoricalAccuracy

计算 top-5 分类正确率。此类是 TopKCategoricalAccuracy 的特殊类。有关更多详细信息, 请参阅[mindspore.train.TopKCategoricalAccuracy](#)。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.train import Top5CategoricalAccuracy
>>>
>>> x = Tensor(np.array([[0.2, 0.5, 0.3, 0.6, 0.2], [0.1, 0.35, 0.5, 0.2, 0.],
...                      [0.9, 0.6, 0.2, 0.01, 0.3]]), mindspore.float32)
>>> y = Tensor(np.array([2, 0, 1]), mindspore.float32)
>>> topk = Top5CategoricalAccuracy()
>>> topk.clear()
>>> topk.update(x, y)
>>> output = topk.eval()
>>> print(output)
1.0
```

7.3.23 mindspore.train.TopKCategoricalAccuracy

class mindspore.train.TopKCategoricalAccuracy(*k*)

计算 top-k 分类正确率。

参数:

- **k** (int) - 计算正确率使用的 Top 类别数。

异常:

- **TypeError** - *k* 不是 int。
- **ValueError** - *k* 小于 1。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.train import TopKCategoricalAccuracy
>>>
>>> x = Tensor(np.array([[0.2, 0.5, 0.3, 0.6, 0.2], [0.1, 0.35, 0.5, 0.2, 0.],
...                      [0.9, 0.6, 0.2, 0.01, 0.3]]), mindspore.float32)
>>> y = Tensor(np.array([2, 0, 1]), mindspore.float32)
>>> topk = TopKCategoricalAccuracy(3)
>>> topk.clear()
>>> topk.update(x, y)
```

(下页继续)

(续上页)

```
>>> output = topk.eval()
>>> print(output)
0.6666666666666666
```

clear()
内部评估结果清零。

eval()
计算 top-k 分类正确率。
返回：
numpy.float64，计算结果。

update(*inputs)
使用预测值 *y_pred* 和真实标签 *y* 更新局部变量。
参数：

- **inputs** - 输入 *y_pred* 和 *y*。 *y_pred* 和 *y* 支持 Tensor、list 或 numpy.ndarray 类型。 *y_pred* 在大多数情况下由范围 [0, 1] 中的浮点数组成， shape 为 (N, C)， 其中 N 是样本数， C 是类别数。 *y* 由整数值组成。 如果使用 one-hot 编码， 则 shape 为 (N, C)； 如果使用类别索引， shape 是 (N,)。

说明: *update* 方法需要接收满足 (*y_pred*, *y*) 格式的输入。 如果某些样本具有相同的正确率， 则将选择第一个样本。

7.4 工具

接口名	概述	支持平台
<code>mindspore.train.auc</code>	使用梯形法则计算曲线下面积 AUC (Area Under the Curve, AUC)。	Ascend GPU CPU
<code>mindspore.train.get_metric_fn</code>	根据输入的 <i>name</i> 获取 <i>metric</i> 的方法。	Ascend GPU CPU
<code>mindspore.train.names</code>	获取所有 <i>metric</i> 的名称。	Ascend GPU CPU
<code>mindspore.train.rearrange_inputs</code>	此装饰器用于根据类的 <i>indexes</i> 属性对输入重新排列。	Ascend GPU CPU

7.4.1 mindspore.train.auc

`mindspore.train.auc(x, y, reorder=False)`
使用梯形法则计算曲线下面积 AUC (Area Under the Curve, AUC)。 这是一个一般函数， 给定曲线上的点， 用于计算 ROC (Receiver Operating Curve, ROC) 曲线下的面积。

参数：

- **x** (Union[np.array, list]) - 从 ROC 曲线 (False Positive Rate, FPR) 来看， np.array 具有假阳性率。 如果是多类， 则为 np.array 列表。 Shape 为 (N)。
- **y** (Union[np.array, list]) - 从 ROC 曲线 (True Positive Rate, TPR) 来看， np.array 具有假阳性率。 如果是多类， 则为 np.array 列表。 Shape 为 (N)。
- **reorder** (bool) - 如果为 False， 那么 *x* 必须是单调上升或下降的； 如果为 True， 那么 *x* 将会按照升序排序。 默认值: False。

返回：

float, 曲线下面积的值 AUC。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore.train import ROC, auc
>>>
>>> y_pred = np.array([[3, 0, 1], [1, 3, 0], [1, 0, 2]])
>>> y = np.array([[0, 2, 1], [1, 2, 1], [0, 0, 1]])
>>> metric = ROC(pos_label=2)
>>> metric.clear()
>>> metric.update(y_pred, y)
>>> fpr, tpr, thre = metric.eval()
>>> output = auc(fpr, tpr)
>>> print(output)
0.5357142857142857
```

7.4.2 mindspore.train.get_metric_fn

`mindspore.train.get_metric_fn(name, *args, **kwargs)`

根据输入的 *name* 获取 metric 的方法。

参数：

- **name** (str) - metric 的方法名，可以通过 `mindspore.train.names()` 接口获取。
- **args** - metric 函数的参数。
- **kwargs** - metric 函数的关键字参数。

返回：

metric 对象，metric 方法的类实例。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore.train import get_metric_fn
>>> metric = get_metric_fn('precision', eval_type='classification')
```

7.4.3 mindspore.train.names

`mindspore.train.names()`

获取所有 metric 的名称。

返回：

List, 所有 metric 的名称列表。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> ms.train.names()
['F1', 'acc', 'accuracy', 'auc', 'bleu_score', 'confusion_matrix', 'confusion_matrix_metric',
'cosine_similarity', 'dice', 'hausdorff_distance', 'loss', 'mae', 'mean_surface_distance',
↪ 'mse',
'occlusion_sensitivity', 'perplexity', 'precision', 'recall', 'roc', 'root_mean_square_
↪ distance',
'top_1_accuracy', 'top_5_accuracy', 'topk']
```

7.4.4 mindspore.train.rearrange_inputs

`mindspore.train.rearrange_inputs` (*func*)

此装饰器用于根据类的 *indexes* 属性对输入重新排列。

此装饰器目前用于 `mindspore.train.Metric` 类的 *update* 方法。

参数:

- **func** (Callable) - 要装饰的候选函数，其输入将被重新排列。

返回:

Callable，用于在函数之间调换输入。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore.train import rearrange_inputs
>>> class RearrangeInputsExample:
...     def __init__(self):
...         self._indexes = None
...
...     @property
...     def indexes(self):
...         return getattr(self, '_indexes', None)
...
...     def set_indexes(self, indexes):
...         self._indexes = indexes
...         return self
...
...     @rearrange_inputs
...     def update(self, *inputs):
...         return inputs
>>>
>>> rearrange_inputs_example = RearrangeInputsExample().set_indexes([1, 0])
>>> outs = rearrange_inputs_example.update(5, 9)
>>> print(outs)
(9, 5)
```


`mindspore.parallel` 提供了大量的自动并行接口，包括并行基础配置、模型加载与转换、函数式并行切分等。
模块导入方法如下：

```
from mindspore import parallel
```

8.1 并行基础配置

<code>mindspore.parallel.auto_parallel.AutoParallel</code>	对顶层 <code>Cell</code> 或函数进行封装，实现单个网络的静态图并行。
<code>mindspore.parallel.nn.GradAccumulation</code>	实现静态图并行梯度累加功能。
<code>mindspore.parallel.nn.MicroBatchInterleaved</code>	实现静态图并行多副本拆分功能，使得计算及通信能并发。
<code>mindspore.parallel.nn.Pipeline</code>	指定流水线并行的 <code>micro_batch</code> 个数及 <code>stage</code> 的划分规则。
<code>mindspore.parallel.nn.PipelineGradReducer</code>	函数式训练场景下，实现流水线并行的梯度规约及累加。

8.1.1 `mindspore.parallel.auto_parallel.AutoParallel`

class `mindspore.parallel.auto_parallel.AutoParallel` (*network*, *parallel_mode*='semi_auto')
对顶层 `Cell` 或函数进行封装，实现单个网络的静态图并行。

说明：

- 使用 `Model` 高阶接口进行训练或推理时，传入 `Model` 的 `network` 必须用 `AutoParallel` 进行封装。
- 使用函数式训练或推理接口时，必须在最外层进行 `AutoParallel` 进行封装，即对 `AutoParallel` 进行编译。
- 使用函数式训练或推理接口时，暂不支持数据下沉场景。

参数:

- **network** (Union[Cell, Function]) - 待封装的前向网络的顶层 Cell 或函数, 定义了将被并行化的核心网络结构。
- **parallel_mode** (str, 可选) - 并行模式, 指定并行策略, 可选项: "semi_auto"、"sharding_propagation"、"recursive_programming"。默认为 "semi_auto", 支持:
 - "semi_auto": 半自动化并行模式, 支持数据并行、算子级并行、优化器并行和流水线并行场景, 默认开启该模式。
 - "sharding_propagation": 策略传播搜索模式。目前不支持动态 shape。
 - "recursive_programming": 递归编程搜索模式。

支持平台:

Ascend

样例:

说明: 需要使用 msrun 进行启动。

```
>>> import os
>>> import mindspore as ms
>>> import mindspore.dataset as ds
>>> from mindspore import nn, ops
>>> from mindspore.communication import init, get_rank
>>> from mindspore.common.initializer import initializer
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> from mindspore.train import Model
>>> from mindspore.train import LossMonitor
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> ms.set_seed(1)
>>>
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>>
>>> def create_dataset(batch_size):
...     dataset_path = os.getenv("DATA_PATH")
...     dataset = ds.MnistDataset(dataset_path)
...     image_transforms = [
...         ds.vision.Rescale(1.0 / 255.0, 0),
...         ds.vision.Normalize(mean=(0.1307,), std=(0.3081,)),
...         ds.vision.HWC2CHW()
...     ]
...     label_transform = ds.transforms.TypeCast(ms.int32)
...     dataset = dataset.map(image_transforms, 'image')
...     dataset = dataset.map(label_transform, 'label')
...     dataset = dataset.batch(batch_size)
...     return dataset
>>>
>>> dataset = create_dataset(32)
>>>
>>> from mindspore import nn, ops, Parameter
>>> from mindspore.common.initializer import initializer, HeUniform
```

(下页继续)

(续上页)

```

>>> import math
>>>
>>> class MatMulCell(nn.Cell):
...     def __init__(self, param=None, shape=None):
...         super().__init__()
...         if shape is None:
...             shape = [28 * 28, 512]
...         weight_init = HeUniform(math.sqrt(5))
...         self.param = Parameter(initializer(weight_init, shape), name="param")
...         if param is not None:
...             self.param = param
...         self.print = ops.Print()
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         out = self.matmul(x, self.param)
...         self.print("out is:", out)
...         return out
>>>
>>> class Network(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.flatten = nn.Flatten()
...         self.layer1 = MatMulCell()
...         self.relu1 = nn.ReLU()
...         self.layer2 = nn.Dense(512, 512)
...         self.relu2 = nn.ReLU()
...         self.layer3 = nn.Dense(512, 10)
...
...     def construct(self, x):
...         x = self.flatten(x)
...         x = self.layer1(x)
...         x = self.relu1(x)
...         x = self.layer2(x)
...         x = self.relu2(x)
...         logits = self.layer3(x)
...         return logits
>>>
>>> import mindspore as ms
>>> from mindspore import nn, ops
>>> from mindspore.parallel.nn import Pipeline, PipelineGradReducer
>>> from mindspore.nn.utils import no_init_parameters
>>>
>>> with no_init_parameters():
>>>     net = Network()
>>>     optimizer = nn.SGD(net.trainable_params(), 1e-2)
>>>     pp_grad_reducer = PipelineGradReducer(optimizer.parameters, opt_shard=False)
>>>
>>> loss_fn = nn.CrossEntropyLoss()
>>> net_with_loss = Pipeline(nn.WithLossCell(net, loss_fn), 4, stage_config={"_backbone.
    ↳flatten":0,
>>>     "_backbone.layer1": 0, "_backbone.relu1": 0, "_backbone.layer2": 1,
>>>     "_backbone.relu2": 1, "_backbone.layer3": 1})
>>> parallel_net = AutoParallel(net_with_loss, parallel_mode="semi_auto")
>>> parallel_net.hsdp()
>>> parallel_net.pipeline(stages=2)

```

(下页继续)

(续上页)

```

>>> parallel_net.dataset_strategy("data_parallel")
>>> parallel_net.save_param_strategy_file(f"/tmp/param_{get_rank()}.ckpt")
>>> parallel_net.set_group_ckpt_save_file(f"/tmp/comm_group_{get_rank()}.ckpt")
>>> parallel_net.dump_local_norm(f"/tmp/local_norm_{get_rank()}.ckpt")
>>> parallel_net.disable_strategy_file_only_for_trainable_params()
>>> parallel_net.enable_fp32_communication()
>>> parallel_net.enable_device_local_norm()
>>> parallel_net.enable_gradients_mean()
>>> parallel_net.disable_gradient_fp32_sync()
>>> parallel_net.disable_loss_repeated_mean()
>>>
>>> loss_monitor = LossMonitor(per_print_times=1)
>>> model = Model(network=parallel_net, optimizer=optimizer)
>>> model.train(epoch=2, train_dataset=dataset, callbacks=[loss_monitor])

```

comm_fusion (config)

用于设置并行通信算子的融合配置。

参数:

- **config** (dict) - 输入格式为 {"通信类型": {"mode": str, "config": None int 或者 list}}, 每种通信算子的融合配置有两个键: "mode" 和 "config"。支持以下通信类型的融合类型和配置:
 - **openstate**: 是否开启通信融合功能。通过 True 或 False 来开启或关闭通信融合功能。默认值: True, 开启通信融合功能。
 - **allreduce**: 进行 AllReduce 算子的通信融合。"mode" 包含 auto、size 和 index。在 auto 模式下, 融合梯度变量的大小, 默认值阈值为"64"MB, "config" 对应的值为 None。在 size 模式下, 需要用户在 config 的字典中指定梯度大小阈值, 这个值必须大于"0"MB。在"mode" 为 index 时, 它与"all_reduce_fusion_config" 相同, 用户需要给"config" 传入一个列表, 里面每个值表示梯度的索引。
 - **allgather**: 进行 AllGather 算子的通信融合。"mode" 包含 auto、size。在 auto 模式下, AllGather 融合由梯度值决定, 其默认融合配置阈值为'64' MB。在 size 模式下, 手动设置 AllGather 算子融合的梯度阈值, 并且其融合阈值必须大于'0' MB。
 - **reducescatter**: 进行 ReduceScatter 算子的通信融合。"mode" 包含 auto、size, auto 和 size 模式的配置方式与 allgather 相同。

异常:

- **TypeError** - 配置项非字典类型。

样例:

```

>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> parallel_net = AutoParallel(net, parallel_mode="semi_auto")
>>> comm_config = {"openstate": True, "allreduce": {"mode": "auto", "config": None}}
>>> net.comm_fusion(config=comm_config)

```

dataset_strategy (config)

设置数据集分片策略。

参数:

- **config** (Union[str, tuple(tuple), tuple(Layout)]) - 数据集切分策略配置。默认模式: "data_parallel", 若需将数据集拆分到不同设备上并行处理, 可设置此策略。若需要加载全批量数据集, 则必须将数据集分片策略设为"full_batch"。若通过模型并行策略 (如 ds_stra ((1, 8), (1, 8))) 将数据集加载到网络中, 需调用 AutoParallel.dataset_strategy(ds_stra) 方法。此外, 数据集策略也支持 Layout 形式。

异常:

- **TypeError** - config 不是字符串或元组类型。
- **TypeError** - config 是元组类型时, 元组中的元素不是"元组类型" 或者"Layout 类型" 中的一个。
- **TypeError** - config 是元组类型时, 且元组中的元素是元组类型时, 子元组的元素类型不是"int"。
- **ValueError** - 输入 config 为空。
- **ValueError** - config 为字符串类型时, 取值不是"full_batch" 或"data_parallel" 中的一个。

disable_gradient_fp32_sync()

开启后, 关闭梯度间的 fp32 通信。

disable_loss_repeated_mean()

开启后, loss 在多卡重复计算时, 均值运算符不会向后执行。

disable_strategy_file_only_for_trainable_params()

默认情况, MindSpore 仅保存/加载可训练参数的策略信息, 调用此接口后, 可支持加载和保存模型的非可训练参数。

dump_local_norm(file_path)

指定 local norm 值的保存路径。

参数:

- **file_path** (str) - 保存路径, 默认值为 ""

异常:

- **TypeError** - 文件路径类型非字符串

enable_device_local_norm()

开启后打印 device local norm 值。

enable_fp32_communication()

开启之后, 通信期间 Reduce 类算子 (AllReduce、ReduceScatter) 强制使用 fp32 数据类型进行通信。

enable_gradients_mean()

开启后, 在并行模式下, 对梯度执行 allreduce 操作后的 mean 操作。

hsdp(shard_size=-1, threshold=64, optimizer_level='level1')

设置优化器并行配置。

参数:

- **shard_size** (int, 可选) - 指定优化器权重跨设备切分通信域的大小, 数值范围可为 (0, 设备数量]。默认值: -1, 表明优化器权重分片组大小将采用每个参数的数据并行组。
- **threshold** (int, 可选) - 切分参数时, 要求目标参数所占内存的最小值, 小于该阈值的参数不会在设备间进行分片。Parameter size = shape[0] * ... * shape[n] * size(dtype)。取值范围: 非负数, 单位: KB。默认值: 64。
- **optimizer_level** (str, 可选) - 配置用于指定优化器切分的切分级别, 静态图下的优化器分片实现与动态图 (如 Megatron) 不一致, 但内存优化效果相同。默认为 "level1"。
 - "level1": 对权重、优化器状态进行切分。
 - "level2": 对权重、优化器状态以及梯度进行切分。
 - "level3": 对权重、优化器状态、梯度进行切分, 并且在反向开始前会对权重额外展开一次 *allgather* 通信, 以释放前向 *allgather* 的显存。

异常:

- **ValueError** - *shard_size* 不是正整数或-1。
- **ValueError** - *threshold* 不是正整数或 0。
- **ValueError** - *optimizer_level* 取值不是"level1", "level2" 或"level3" 中的一个。

load_operator_strategy_file (*file_path*)

在使用策略传播模式时, 设置加载策略 JSON 文件的路径。

说明:

- 只在策略传播并行模式下生效。
- 在分布式训练场景, 用户可以首先在单卡下用 Dryrun 设置策略, 然后再加载策略进行分布式训练。

警告:

- 实验性接口, 未来可能变更或移除。
- 暂不支持加载策略时使用 Layout 格式。

参数:

- **file_path** (str) - 加载并行策略 JSON 文件的路径, 必须是绝对路径。

异常:

- **TypeError** - 文件路径类型非字符串。
- **KeyError** - 文件路径非绝对路径。
- **KeyError** - 文件路径非 JSON 文件后缀结尾。

样例:

```
>>> import math
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import nn, ops
>>> from mindspore.communication.management import init
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> from mindspore.common.initializer import initializer, HeUniform
>>>
>>> class ParallelNetwork(nn.Cell):
...     def __init__(self, strategy=None):
...         super().__init__()
...         self.flatten = ops.Flatten()
...         self.fc1_weight = ms.Parameter(initializer(HeUniform(math.sqrt(5))), shape=[
...             16, 10], dtype=ms.float32), name="fc1")
...         self.matmul1 = ops.MatMul().shard(strategy)
...         self.relu1 = ops.ReLU()
...
...     def construct(self, x):
...         x = self.flatten(x)
...         x = self.matmul1(x, self.fc1_weight)
...         x = self.relu1(x)
...         return x
>>>
```

(下页继续)

(续上页)

```
>>> init(backend_name='hccl')
>>> strategy = ((1, 1), (1, 2))
>>> net = ParallelNetwork(strategy)
>>> parallel_net = AutoParallel(net, parallel_mode='sharding_propagation')
>>> parallel_net.load_operator_strategy_file("/tmp/strategy.json")
```

load_param_strategy_file(file_path)

设置加载并行策略 checkpoint 的路径，默认仅加载可训练参数的策略信息。

参数：

- **file_path** (str) - 加载路径。

异常：

- **TypeError** - 文件路径类型非字符串。

样例：

```
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> parallel_net = AutoParallel(net)
>>> parallel_net.load_param_strategy_file(file_path="./train_strategy.ckpt")
```

no_init_parameters_in_compile()

开启后，在编译过程中，不进行模型权重参数初始化。

警告：实验性接口，未来可能变更或移除。

样例：

```
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> parallel_net = AutoParallel(net, parallel_mode="semi_auto")
>>> parallel_net.no_init_parameters_in_compile()
```

pipeline(stages=1, output_broadcast=False, interleave=False, scheduler='1f1b')

配置流水线阶段的数量，stage 的结果是否广播，是否启用 interleave 调度，配置流水线并行时配置调度策略。

参数：

- **stages** (int, 可选) - 设置流水线并行的阶段信息。默认值：1。
- **output_broadcast** (bool, 可选) - 在执行流水线并行推理时，是否将最后阶段的结果广播到其他阶段。默认值：False。
- **interleave** (bool, 可选) - 是否启用交错调度。默认值：False。
- **scheduler** (str, 可选) - 调度器的类型。默认值：1f1b。

异常：

- **TypeError** - stages 的类型非 int。
- **ValueError** - stages <= 0。
- **TypeError** - output_broadcast 的类型非 bool。
- **TypeError** - interleave 的类型非 bool。

- **TypeError** - *scheduler* 的类型非 str。
- **ValueError** - *scheduler* 的类型非支持。

print_local_norm()

开启后打印 local norm 值。

样例:

```
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> parallel_net = AutoParallel(net, parallel_mode="semi_auto")
>>> parallel_net.print_local_norm()
```

save_operator_strategy_file(file_path)

在使用策略传播模式时，设置保存策略 JSON 文件的路径。

说明:

- 只在策略传播并行模式下生效。
- 在分布式训练场景，用户可以首先在单卡下用 Dryrun 设置策略，然后再加载策略进行分布式训练。

警告:

- 实验性接口，未来可能变更或移除。
- 暂不支持加载策略时使用 Layout 格式。

参数:

- **file_path** (str) - 保存并行策略 JSON 文件的路径，必须是绝对路径。

异常:

- **TypeError** - 文件路径类型非字符串。
- **KeyError** - 文件路径非绝对路径。
- **KeyError** - 文件路径非 JSON 文件后缀结尾。

样例:

```
>>> import math
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import nn, ops
>>> from mindspore.communication.management import init
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> from mindspore.common.initializer import initializer, HeUniform
>>>
>>> class ParallelNetwork(nn.Cell):
...     def __init__(self, strategy=None):
...         super().__init__()
...         self.flatten = ops.Flatten()
...         self.fc1_weight = ms.Parameter(initializer(HeUniform(math.sqrt(5))), shape=[
```

(下页继续)

(续上页)

```

...         16, 10], dtype=ms.float32), name="fc1")
...     self.matmul1 = ops.MatMul().shard(strategy)
...     self.relu1 = ops.ReLU()
...
...     def construct(self, x):
...         x = self.flatten(x)
...         x = self.matmul1(x, self.fc1_weight)
...         x = self.relu1(x)
...         return x
>>>
>>> init(backend_name='hccl')
>>> strategy = ((1, 1), (1, 2))
>>> net = ParallelNetwork(strategy)
>>> parallel_net = AutoParallel(net, parallel_mode='sharding_propagation')
>>> parallel_net.save_operator_strategy_file("/tmp/strategy.json")

```

save_param_strategy_file (*file_path*)

设置保存并行策略 checkpoint 的路径，默认仅保存可训练参数的策略信息。

参数：

- **file_path** (str) - 保存路径。

异常：

- **TypeError** - 文件路径类型非字符串。

set_group_ckpt_save_file (*file_path*)

指定图编译过程中所创建 group 的保存路径。

参数：

- **file_path** (str) - 保存路径。

异常：

- **TypeError** - 文件路径类型非字符串。

transformer_opt (*file_path*)

并行加速配置文件，配置项可以参考 [parallel_speed_up.json](#)。当设置为 None 时，表示不启用。

参数：

- **file_path** (Union[str, None]) - 并行加速配置文件，配置项可以参考 [parallel_speed_up.json](#)。当设置为 None 时，表示不启用。
 - **recomputation_communication_overlap** (bool): 为 True 时表示开启反向重计算和通信掩盖。默认值：False。
 - **grad_matmul_communication_overlap** (bool): 为 True 时表示开启反向 Matmul 和通信掩盖。默认值：False。
 - **grad_fa_allgather_overlap** (bool): 为 True 时表示在序列并行和开启 FlashAttentionScoreGrad 算子时，开启重计算以掩盖重复的 AllGather。默认值：False。
 - **enable_communication_fusion** (bool): 为 True 时表示开启通信融合进行通信算子 task 数量优化。默认值：False。
 - **grad_computation_allreduce_overlap** (bool): 为 True 时表示开启梯度 dx 计算与数据并行梯度通信的掩盖，暂时不支持 O2 编译模式下开启。注意在数据并行梯度通信和计算掩盖良好的情况下，开启该选项后性能不一定有提升，请根据实际场景确定是否开启。默认值：False。

- **computation_allgather_overlap** (bool): 为 True 时表示开启正向计算与优化器并行的 AllGather 通信的掩盖, 暂时不支持 O2 编译模式下开启。注意在权重聚合通信和计算掩盖良好的情况下, 开启该选项后性能不一定有提升, 请根据实际场景确定是否开启。默认值: False。
- **enable_concat_eliminate_opt** (bool): 为 True 时表示开启 Concat 消除优化, 当前在开启细粒度双副本优化时有收益。默认值: False。
- **enable_begin_end_inline_opt** (bool): 为 True 时表示开启首尾 micro_batch 子图的内联, 用于半自动并行子图模式, 流水线并行场景, 一般需要和其他通信计算掩盖优化一起使用。默认值: False。
- **computation_communication_fusion_level** (int): 控制通算融合的级别。默认值: 0。注: 需要配套 Ascend Training Solution 24.0.RC2 以上版本使用。该参数属于实验性质参数, 未来可能变更或移除。
 - * 0: 不启用通算融合。
 - * 1: 仅对前向节点使能通算融合。
 - * 2: 仅对反向节点使能通算融合。
 - * 3: 对所有节点使能通算融合。
- **dataset_broadcast_opt_level** (int): 数据集读取的优化级别, 目前只支持 O0/O1 模式, O2 模式下不生效。默认值: 0。
 - * 0: 不启用数据集读取优化。
 - * 1: 优化流水线并行中, Stage 间的数据读取。
 - * 2: 优化模型并行维度数据的读取。
 - * 3: 同时优化场景 1 和 2。
- **allreduce_and_biasadd_swap** (bool): 为 True 时表示开启 matmul-add 结构下, 通信算子与 Add 算子执行顺序互换。当前仅支持 bias 为一维的情况。默认值: False。
- **enable_allreduce_slice_to_reducescatter** (bool): 为 True 时, 表示开启 AllReduce 优化。在 batchmatmul 模型并行引入 AllReduce 的场景中, 如果后续节点是配置了模型并行的 StridedSlice 算子, 在已识别可优化的模式中, 将 AllReduce 优化为 ReduceScatter。典型的用在开启了 groupwise alltoall 的 MoE 模块。默认值: False。
- **enable_interleave_split_concat_branch** (bool): 为 True 时, 表示针对带 enable_interleave 属性的 Split 和 Concat 算子形成的分支, 开启通信计算并行优化。典型的使用场景为 MoE 模块并行场景, 对输入数据进行 split 后, 对各切片数据进行 MoE 模块运算, 再对分支结果进行 Concat, 开启后各分支的 MoE 模块进行通信计算并行优化。默认值: False。
- **enable_interleave_parallel_branch** (bool): 为 True 时, 表示针对可并行的分支, 如果分支汇聚点带 parallel_branch 属性, 开启通信计算并行优化。典型的使用场景为 MoE 模块带路由专家和共享专家分支的并行场景, 开启后并行分支进行通信计算并行优化。默认值: False。

样例:

```
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>>
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> net = AutoParallel(net, parallel_mode="semi_auto")
>>> net.transformer_opt("./parallel_speed_up.json")
```


8.1.2 mindspore.parallel.nn.GradAccumulation

class mindspore.parallel.nn.GradAccumulation(*network*, *micro_size*)
实现静态图并行梯度累加功能。

参数：

- **network** (Cell) - 将进行梯度累加的网络。
- **micro_size** (int) - MicroBatchSize。

异常：

- **TypeError** - *network* 不是 cell 类型输入。
- **TypeError** - *micro_size* 不是整数类型。
- **ValueError** - *micro_size* 值异常，为 0 或者负数。

支持平台：

Ascend

样例：

```
>>> from mindspore.parallel.nn import GradAccumulation
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> net = GradAccumulation(net, 4)
```

8.1.3 mindspore.parallel.nn.MicroBatchInterleaved

class mindspore.parallel.nn.MicroBatchInterleaved(*network*, *interleave_num*=2)

实现静态图并行多副本拆分功能，使得计算及通信能并发。使用场景：当在半自动模式以及网络中存在模型并行时，第 1 份切片数据前向计算的同时，第 2 份数据将会进行模型并行的通信，以此来达到通信计算并发的性能加速。

参数：

- **network** (Cell) - 需要封装的网络。
- **interleave_num** (int, 可选) - batch size 的拆分份数，默认值：2。

输入：

tuple[Tensor]，与传入的 *network* 的输入一致。

输出：

被封装后的网络。传入的 *network* 的输出只能是单个 Tensor。

支持平台：

Ascend

样例：

```
>>> import mindspore.nn as nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> net = nn.MicroBatchInterleaved(net, 2)
```

8.1.4 mindspore.parallel.nn.Pipeline

class mindspore.parallel.nn.**Pipeline** (*network, micro_size, stage_config=None*)
指定流水线并行的 *micro_batch* 个数及 *stage* 的划分规则。

说明: *micro_size* 必须大于等于 *pipeline_stages*。

参数:

- **network** (Cell) - 将进行 pp 并行的网络。
- **micro_size** (int) - MicroBatchsize。
- **stage_config** (dict, 可选) - 流水线并行对于每个 cell 的 stage 配置。默认值: None。

异常:

- **TypeError** - *net* 不是 cell 类型输入。
- **TypeError** - *micro_size* 不是整数类型。
- **ValueError** - *micro_size* 值异常, 为 0 或者负数。
- **KeyError** - *dict* cell 名称匹配异常, 遍历当前 *net* 下所有 *cell* 仍有剩余的配置项。

支持平台:

Ascend

样例:

```
>>> from mindspore.parallel.nn import Pipeline
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> net = Pipeline(net, 4, stage_config={"cell_name_0": 0, "cell_name_1": 1})
```

8.1.5 mindspore.parallel.nn.PipelineGradReducer

class mindspore.parallel.nn.**PipelineGradReducer** (*parameters, scale_sense=1.0, opt_shard=None*)
函数式训练场景下, 实现流水线并行的梯度规约及累加。

参数:

- **parameters** (list) - 将进行 pp 并行的网络。
- **scale_sense** (float, 可选) - 梯度的尺度感知。默认值: 1.0。
- **opt_shard** (bool, 可选) - 如果使用优化器, 需要配置为 True。默认值: None。

异常:

- **RuntimeError** - *mode* 不是图模式。

支持平台:

Ascend GPU

样例:

说明: 在运行以下示例之前, 您需要配置通信环境变量。对于 Ascend 设备, 用户需要准备 rank 表, 并设置 rank_id 和 device_id。请参阅 [rank table 启动](#) 以获取更多详细信息。此示例需要在多设备上运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import nn, ops, Tensor
>>> from mindspore.communication import init
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> ms.reset_auto_parallel_context()
>>> init()
>>> ms.set_seed(1)
>>>
>>> class Network(nn.Cell):
...     def __init__(self, in_features, out_features, sens=1.0):
...         super().__init__()
...         self.layer1 = nn.Dense(in_features, 16)
...         self.relu1 = nn.ReLU()
...         self.layer2 = nn.Dense(16, 16)
...         self.relu2 = nn.ReLU()
...         self.layer3 = nn.Dense(16, out_features)
...
...     def construct(self, x):
...         x = self.layer1(x)
...         x = self.relu1(x)
...         x = self.layer2(x)
...         x = self.relu2(x)
...         logits = self.layer3(x)
...         return logits
>>>
>>> size, in_features, out_features = 16, 32, 10
>>> net = Network(in_features, out_features)
>>> net.layer1.pipeline_stage = 0
>>> net.relu1.pipeline_stage = 0
>>> net.layer2.pipeline_stage = 0
>>> net.relu2.pipeline_stage = 1
>>> net.layer3.pipeline_stage = 1
>>> loss_fn = nn.CrossEntropyLoss()
>>> optimizer = nn.SGD(net.trainable_params(), 1e-2)
>>> net_with_loss = nn.Pipeline(nn.WithLossCell(net, loss_fn), 2)
>>> net_with_loss.set_train()
>>> def forward_fn(inputs, target):
...     loss = net_with_loss(inputs, target)
...     return loss
>>>
>>> grad_fn = ops.value_and_grad(forward_fn, None, net_with_loss.trainable_params())
>>> pp_grad_reducer = nn.PipelineGradReducer(optimizer.parameters)
>>>
>>> @ms.jit
>>> def train_one_step(inputs, target):
...     loss, grads = grad_fn(inputs, target)
...     grads = pp_grad_reducer(grads)
...     optimizer(grads)
...     return loss, grads
```

(下页继续)

(续上页)

```
>>>
>>> parallel_net = AutoParallel(train_one_step, parallel_mode="semi_auto")
>>> parallel_net.pipeline(stages=2)
>>> inputs = Tensor(np.ones([size, in_features]).astype(np.float32))
>>> label = Tensor(np.ones([size, out_features]).astype(np.float32))
>>> loss, _ = train_one_step(inputs, label)
>>> print(loss)
46.36721
```

8.2 模型加载与转换

<code>mindspore.parallel.convert_checkpoints</code>	将一个分布式网络的 Checkpoint 由源切分策略转换到目标切分策略。
<code>mindspore.parallel.convert_checkpoint_by_rank</code>	将一个分布式网络的 Checkpoint 由源切分策略转换到目标切分策略，对特定一个 rank 进行转换。
<code>mindspore.parallel.load_distributed_checkpoint</code>	加载分布式 checkpoint 参数到网络，用于分布式推理。
<code>mindspore.parallel.load_segmented_checkpoints</code>	加载指定路径下所有 checkpoint 文件。
<code>mindspore.parallel.rank_list_for_convert</code>	在对分布式 Checkpoint 转换的过程中，获取目标 rank 的 Checkpoint 文件所需的源 Checkpoint 文件 rank 列表。
<code>mindspore.parallel.unified_safetensors</code>	将多个 safetensors 文件合并为一系列统一的 safetensors 文件。

8.2.1 mindspore.parallel.convert_checkpoints

`mindspore.parallel.convert_checkpoints` (*src_checkpoints_dir*, *dst_checkpoints_dir*, *ckpt_prefix*,
src_strategy_file=None, *dst_strategy_file*=None, *process_num*=1,
output_format='ckpt')

将一个分布式网络的 Checkpoint 由源切分策略转换到目标切分策略。

说明: *src_checkpoints_dir* 目录必须设置为“src_checkpoints_dir/rank_0/a.ckpt”这样的目录结构，rank 号必须作为子目录并且该 rank 的 Checkpoint 必须放置于该子目录下。如果多个文件存在于一个 rank 目录下，将会选择名字的字典序最高的文件。多进程设置数量与主机规模有关，不推荐设置太大，否则容易导致卡死。

参数:

- **src_checkpoints_dir** (str) - 源 Checkpoint 文件所在的目录。
- **dst_checkpoints_dir** (str) - 目标 Checkpoint 文件存储的目录。
- **ckpt_prefix** (str) - 目标 Checkpoint 前缀名。
- **src_strategy_file** (str, 可选) - 源切分策略 proto 文件名，由 `mindspore.parallel.auto_parallel.AutoParallel.save_param_strategy_file()` 接口存储下来的文件。当其为 None 时，表示切分策略为不切分。默认值: None。

- **dst_strategy_file** (str, 可选) - 目标切分策略 proto 文件名, 由 `mindspore.parallel.auto_parallel.AutoParallel.save_param_strategy_file()` 接口存储下来的文件。当其为 `None` 时, 表示切分策略为不切分。默认值: `None`。
- **process_num** (int, 可选) - 控制并行处理的进程数量。默认值: 1。
- **output_format** (str, 可选) - 控制转换后输出的 checkpoint 格式。可以设置为 "ckpt" 或 "safetensors"。默认值: "ckpt"。

异常:

- **ValueError** - `src_strategy_file` 或者 `dst_strategy_file` 不是正确的切分策略 proto 文件。
- **NotADirectoryError** - `src_checkpoints_dir` 或者 `dst_checkpoints_dir` 不是一个目录。
- **ValueError** - `src_checkpoints_dir` 中缺失了 Checkpoint 文件。
- **TypeError** - `src_strategy_file` 或者 `dst_strategy_file` 不是字符串。

支持平台:

Ascend

样例:

```
>>> from mindspore.parallel import convert_checkpoints
>>> convert_checkpoints(src_checkpoints_dir, dst_checkpoints_dir, "dst_checkpoint",
...                     "./src_strategy.ckpt", "./dst_strategy.ckpt")
```

8.2.2 mindspore.parallel.convert_checkpoint_by_rank

`mindspore.parallel.convert_checkpoint_by_rank(rank_id, checkpoint_files_map, save_checkpoint_file_name, src_strategy_file=None, dst_strategy_file=None)`

将一个分布式网络的 Checkpoint 由源切分策略转换到目标切分策略, 对特定一个 rank 进行转换。

参数:

- **rank_id** (int) - 待转换得到的 Checkpoint 的 rank 号。
- **checkpoint_files_map** (dict) - 源 Checkpoint 字典, 其 key 为 rank 号, value 为该 rank 号对应的 Checkpoint 文件路径。
- **save_checkpoint_file_name** (str) - 目标 Checkpoint 路径以及名称。
- **src_strategy_file** (str) - 源切分策略 proto 文件名 (由 `AutoParallel.save_param_strategy_file(strategy_ckpt_save_file)` 生成)。当其为 `None` 时, 表示切分策略为不切分。默认值: `None`。
- **dst_strategy_file** (str) - 目标切分策略 proto 文件名 (由 `AutoParallel.save_param_strategy_file(strategy_ckpt_save_file)` 生成)。当其为 `None` 时, 表示切分策略为不切分。默认值: `None`。

异常:

- **ValueError** - `src_strategy_file` 或者 `dst_strategy_file` 不是正确的切分策略 proto 文件。
- **ValueError** - `checkpoint_files_map` 内的元素不是正确的 Checkpoint 文件。
- **ValueError** - `save_checkpoint_file_name` 不以 ".ckpt" 结尾。
- **TypeError** - `checkpoint_files_map` 不是一个字典。
- **TypeError** - `src_strategy_file` 或者 `dst_strategy_file` 不是字符串。
- **TypeError** - `rank_id` 不是一个整数。

- **TypeError** - `save_checkpoint_file_name` 不是字符串。

支持平台：

Ascend

样例：

```
>>> from mindspore.parallel import rank_list_for_convert, convert_checkpoint_by_rank
>>> dst_device_num = 8
>>> for rank_id in range(dst_device_num):
...     rank_list = rank_list_for_convert(rank_id, "./src_strategy.ckpt", "./dst_strategy.
    ↪ckpt")
...     checkpoint_files_map = {}
...     for rank in rank_list:
...         checkpoint_files_map[rank] = "./origin_checkpoint_rank{}/pangu{}-100_2.ckpt".
    ↪format(rank)
...     save_checkpoint_file_name = "./new_checkpoint_rank{}/pangu{}-100_2.ckpt".format(rank_
    ↪id)
...     convert_checkpoint_by_rank(rank_id, checkpoint_files_map, save_checkpoint_file_name,
...                               "./src_strategy.ckpt", "./dst_strategy.ckpt")
```

8.2.3 mindspore.parallel.load_distributed_checkpoint

`mindspore.parallel.load_distributed_checkpoint` (*network, checkpoint_filenames=None, predict_strategy=None, train_strategy_filename=None, strict_load=False, dec_key=None, dec_mode='AES-GCM', format='ckpt', unified_safetensors_dir=None, dst_safetensors_dir=None, rank_id=None, output_format='safetensors', name_map=None, max_process_num=64, return_param_dict=False*)

加载分布式 checkpoint 参数到网络，用于分布式推理。

说明：只有 `format` 设置为 `safetensors` 并且 `network` 为 `None` 时，`output_format` 才会生效。

参数：

- **network** (Cell) - 分布式预测网络，`format` 为 `safetensors` 时，`network` 入参可以不传递或传递为 `None`，此时接口执行保存模式。
- **checkpoint_filenames** (list[str]) - checkpoint 文件的名称，按 rank id 顺序排列。默认值：None。
- **predict_strategy** (Union[dict, str]) - 预测时参数的切分策略或者策略文件。默认值：None。
- **train_strategy_filename** (str) - 训练策略 proto 文件名。默认值：None。
- **strict_load** (bool) - 表示是否严格加载参数到网络。如果值为 `False`，则当 checkpoint 文件中参数名称的后缀与网络中的参数相同时，加载参数到网络。当类型不一致时，对相同类型的参数进行类型转换，如从 `float32` 到 `float16`。默认值：`False`。
- **dec_key** (Union[None, bytes]) - 用于解密的字节类型 key。如果 `value` 为 `None`，则不需要解密。默认值：None。
- **dec_mode** (str) - 指定解密模式，目前支持 'AES-GCM'，'AES-CBC' 和 'SM4-CBC'。仅当 `dec_key` 不设为 `None` 时，该参数有效。默认值：'AES-GCM'。
- **format** (str) - 待加载进网络的输入权重格式。可以设置为“ckpt”或“safetensors”。默认值：“ckpt”。
- **unified_safetensors_dir** (str) - 待加载进网络的输入权重文件目录。默认值：None。

- **dst_safetensors_dir** (str) - 保存模式场景下，权重的保存目录。
- **rank_id** (int) - 卡的逻辑序号。非保存模式下，通过初始化网络全局自动获取；保存模式下，按传入序号保存文件，若未传入，则全量保存。
- **output_format** (str, 可选) - 控制转换后输出的 checkpoint 格式。可以设置为“ckpt”或“safetensors”。默认值：“safetensors”。
- **name_map** (dict) - 权重映射字典，切分完的权重加载到网络或保存之前，会按照映射字典修改权重名字。默认值：None。
- **max_process_num** (int) - 最大进程数。默认值：64。
- **return_param_dict** (bool) - 是否返回 *param_dict*。默认值：False。

异常：

- **TypeError** - 输入类型不符合要求。
- **ValueError** - 无法加载 checkpoint 文件到网络。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import os
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.dataset as ds
>>> from mindspore import nn, ops, train
>>> from mindspore.communication import init
>>> from mindspore.parallel import load_distributed_checkpoint
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> from mindspore.nn.utils import no_init_parameters
>>> from mindspore.common.initializer import initializer, One
>>>
>>> step_per_epoch = 4
>>>
>>> # Define the network structure.
>>> class Net(nn.Cell):
...     def __init__(self, matmul_size, strategy=None):
...         super().__init__()
...         self.matmul_weight = ms.Parameter(initializer(One(), matmul_size, ms.float32))
...         self.matmul = ops.MatMul()
...         self.neg = ops.Neg()
...         if strategy is not None:
...             self.matmul.shard(strategy)
...
...     def construct(self, inputs):
...         x = self.matmul(inputs, self.matmul_weight)
...         x = self.neg(x)
...         return x
>>>
>>> # Create dataset.
```

(下页继续)

(续上页)

```

>>> def get_dataset(*inputs):
...     def generate():
...         for _ in range(step_per_epoch):
...             yield inputs
...     return generate
>>>
>>> # Train network and save distributed checkpoint.
>>> def train_net():
...     ms.set_context(mode=ms.GRAPH_MODE)
...     init()
...     np.random.seed(1)
...     input_data = np.random.rand(16, 96).astype(np.float32)
...     label_data = np.random.rand(16, 16).astype(np.float32)
...     fake_dataset = get_dataset(input_data, label_data)
...     dataset = ds.GeneratorDataset(fake_dataset, ["input", "label"])
...
...     # Set parallel strategy.
...     strategy = ((1, 4), (4, 1))
...     with no_init_parameters():
...         network = Net(matmul_size=(96, 16), strategy=strategy)
...         net_opt = nn.Momentum(network.trainable_params(), 0.01, 0.9)
...
...     net_loss = nn.SoftmaxCrossEntropyWithLogits(reduction="mean")
...     network = AutoParallel(network, parallel_mode="semi_auto")
...     network.save_param_strategy_file(file_path="./train_strategy.ckpt")
...     model = ms.Model(network=network, loss_fn=net_loss, optimizer=net_opt)
...     ckpt_config = train.CheckpointConfig(keep_checkpoint_max=1, integrated_save=False)
...     global_rank_id = int(os.getenv("RANK_ID"))
...     ckpt_path = "./rank_{}/_ckpt".format(global_rank_id)
...     ckpt_callback = train.ModelCheckpoint(prefix="parallel", directory=ckpt_path,
...     ↪config=ckpt_config)
...     model.train(epoch=2, train_dataset=dataset, callbacks=[ckpt_callback], dataset_sink_
...     ↪mode=False)
>>>
>>> # Load distributed checkpoint and test.
>>> def load_model():
...     ms.set_context(mode=ms.GRAPH_MODE)
...     init()
...     predict_data = ms.Tensor(np.random.randn(128, 96).astype(np.float32))
...     with no_init_parameters():
...         network = Net(matmul_size=(96, 16))
...         network = AutoParallel(network, parallel_mode="semi_auto")
...         network.dataset_strategy(config="full_batch")
...         train_strategy_file = "./train_strategy.ckpt"
...         network.save_param_strategy_file(file_path=train_strategy_file)
...     model = ms.Model(network)
...     predict_layout = model.infer_predict_layout(ms.Tensor(predict_data))
...     ckpt_file_list = ["/rank_{}/_ckpt/parallel-2_4.ckpt".format(i) for i in range(0,
...     ↪device_num)]
...     load_distributed_checkpoint(network, ckpt_file_list, predict_layout, None)
...     predict_result = model.predict(predict_data)
...     print(predict_result)
>>>
>>> train_net()
>>> load_model()
[[-7.3259363 -7.497216 -7.398196 ... -7.374962 -7.204874 -7.234935 ]

```

(下页继续)

(续上页)

```
[ 3.362938    3.3535435   3.3832688   ...   3.4263954   3.279045    3.3202887]
...
[ 1.6067538   1.6244187   1.5384722   ...   1.5449994   1.6195512   1.6176052]]
```

8.2.4 mindspore.parallel.load_segmented_checkpoints

`mindspore.parallel.load_segmented_checkpoints` (*ckpt_file_dir*, *net=None*, *strict_load=False*, *filter_prefix=None*, *dec_key=None*, *dec_mode='AES-GCM'*, *specify_prefix=None*, *choice_func=None*)

加载指定路径下所有 checkpoint 文件。当 *ckpt_file_dir* 指定路径下存在多个 checkpoint 文件时，会逐个读取各个 checkpoint 文件并返回合并后的字典。

说明：

- *specify_prefix* 和 *filter_prefix* 参数已被弃用，推荐使用 *choice_func* 代替。*specify_prefix* 和 *filter_prefix* 的功能相互之间没有影响，并且使用这两个参数中的任何一个都将覆盖 *choice_func*。
- 如果发现没有参数被成功加载，将会报 `ValueError`。

参数：

- **ckpt_file_dir** (str) - checkpoint 的文件夹路径。
- **net** (Cell) - 加载 checkpoint 参数的网络。默认值：None。
- **strict_load** (bool) - 加载参数到网络中时，是否根据参数名进行全词匹配以及数据类型自动转换。如果是 `False`，它将根据相同的后缀名将参数字典中的参数加载到网络中，并会在精度不匹配时，进行强制精度转换，比如将 `float32` 转换为 `float16`。默认值：`False`。
- **filter_prefix** (Union[str, list[str], tuple[str]]) - 废弃（请参考参数 *choice_func*）。以 *filter_prefix* 开头的参数将不会被加载。默认值：None。
- **dec_key** (Union[None, bytes]) - 用于解密的字节类型密钥，如果值为 None，则不需要解密。默认值：None。
- **dec_mode** (str) - 该参数仅当 *dec_key* 不为 None 时有效。指定解密模式，目前支持 "AES-GCM"，"AES-CBC" 和 "SM4-CBC"。默认值："AES-GCM"。
- **specify_prefix** (Union[str, list[str], tuple[str]]) - 废弃（请参考参数 *choice_func*）。以 *specify_prefix* 开头的参数将会被加载。默认值：None。
- **choice_func** (Union[None, function]) - 函数的输入值为字符串类型的 Parameter 名称，并且返回值是一个布尔值。如果返回 `True`，则匹配自定义条件的 Parameter 将被加载。如果返回 `False`，则匹配自定义条件的 Parameter 将被删除。默认值：None。

返回：

字典，key 是参数名称，value 是 Parameter 类型。当使用 `mindspore.save_checkpoint()` 的 *append_dict* 参数和 `mindspore.train.CheckpointConfig` 的 *append_info* 参数保存 checkpoint，*append_dict* 和 *append_info* 是 dict 类型，且它们的值 value 是 string 时，加载 checkpoint 得到的返回值是 string 类型，其他情况返回值均是 Parameter 类型。

异常：

- **TypeError** - *ckpt_file_dir* 入参非字符串类型。
- **ValueError** - checkpoint 文件夹路径不存在或传入路径非文件夹。
- **ValueError** - checkpoint 文件格式不正确。

- **ValueError** - 没有一个参数被成功加载。
- **TypeError** - *specify_prefix* 或者 *filter_prefix* 的数据类型不正确。

支持平台:

Ascend

8.2.5 mindspore.parallel.rank_list_for_convert

`mindspore.parallel.rank_list_for_convert(rank_id, src_strategy_file=None, dst_strategy_file=None)`

在对分布式 Checkpoint 转换的过程中, 获取目标 rank 的 Checkpoint 文件所需的源 Checkpoint 文件 rank 列表。

参数:

- **rank_id** (int) - 待转换的 Checkpoint 的 rank 号。
- **src_strategy_file** (str) - 源切分策略 proto 文件名, 由 *AutoParallel.save_param_strategy_file(strategy_ckpt_save_file)* 接口存储下来的文件。当其为 None 时, 表示切分策略为不切分。默认值: None。
- **dst_strategy_file** (str) - 目标切分策略 proto 文件名, 由 *AutoParallel.save_param_strategy_file(strategy_ckpt_save_file)* 接口存储下来的文件。当其为 None 时, 表示切分策略为不切分。默认值: None。

返回:

转换得到 rank_id 的分布式 Checkpoint 所需要的卡号列表。

异常:

- **ValueError** - *src_strategy_file* 或者 *dst_strategy_file* 不是正确的切分策略 proto 文件。
- **TypeError** - *src_strategy_file* 或者 *dst_strategy_file* 不是字符串。
- **TypeError** - *rank_id* 不是一个整数。

支持平台:

Ascend

样例:

```
>>> from mindspore.parallel import rank_list_for_convert
>>> rank_id = 0
>>> rank_list = rank_list_for_convert(rank_id, "./src_strategy.ckpt", "./dst_strategy.ckpt")
>>> checkpoint_files_map = {}
>>> for rank in rank_list:
...     checkpoint_files_map[rank] = "./pangu{ }-100_2.ckpt".format(rank)
```

8.2.6 mindspore.parallel.unified_safetensors

`mindspore.parallel.unified_safetensors(src_dir, src_strategy_file, dst_dir, merge_with_redundancy=True, file_suffix=None, max_process_num=64, choice_func=None, split_dst_file=())`

将多个 safetensors 文件合并为一系列统一的 safetensors 文件。

参数:

- **src_dir** (str) - 源权重保存目录。
- **src_strategy_file** (str) - 源权重切分策略文件。

- **dst_dir** (str) - 目标保存目录。
- **merge_with_redundancy** (bool, 可选) - 合并源权重文件是否是去冗余保存的 safetensors 文件。默认值是：True，合并的源权重文件是完整的。
- **file_suffix** (str, 可选) - 指定合并 safetensors 的文件名后缀。默认值是：None，合并源权重目录下所有的 safetensors 文件。
- **max_process_num** (int, 可选) - 最大进程数。默认值：64。
- **choice_func** (callable, 可选) - 可调用的函数，用于筛选参数或者修改参数名，函数的返回值必须为 str 或者 bool 类型。默认值：None。
- **split_dst_file** (tuple, 可选) - 手动将任务切分为多个子任务执行，以元组形式表示，元组中包含两个元素：第一个元素表示当前子任务编号，第二个元素表示任务的总数量。该参数支持在单台机器上多次切分所执行的任务，也支持在多台机器上分别执行不同的子任务。默认值：()。

异常：

- **ValueError** - 如果某个 rank 的 safetensors 文件丢失。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> src_dir = "/usr/safetensors/llama31B/4p_safetensors/"
>>> src_strategy_file = "/usr/safetensors/llama31B/strategy_4p.ckpt"
>>> dst_dir = "/usr/safetensors/llama31B/merge_llama31B_4p/"
>>> ms.parallel.unified_safetensors(src_dir, src_strategy_file, dst_dir)
```

8.3 函数式并行切分

<i>mindspore.parallel.function.reshape</i>	将张量从一种分布式排布转换成另一种分布式排布。
<i>mindspore.parallel.Layout</i>	描述集群设备的拓扑抽象，用于张量分片在集群上的放置。
<i>mindspore.parallel.shard</i>	指定一个 Cell 或函数的输入、输出切分策略。

8.3.1 mindspore.parallel.function.reshape

`mindspore.parallel.function.reshape` (*tensor, layout*)
将张量从一种分布式排布转换成另一种分布式排布。其中，传入的 `layout` 需要为 `mindspore.parallel.Layout` 类型，可参考：[*mindspore.parallel.Layout*](#) 的描述。

说明：

- 在图模式下，可以利用此方法设置某个张量的并行切分策略，未设置的会自动通过策略传播方式配置。
- 在 PyNative 模式下，可以利用此方法，对某个以图模式并行执行的 Cell（即 PyNative 模式下使用了 `Cell.shard/F.shard` 的 Cell）中的张量进行排布指定。

参数：

- **tensor** (Tensor) - 待设置切分策略的张量。
- **layout** (Layout) - 指定精准排布的方案，包括描述设备的排布（`device_matrix`）和设备矩阵的映射别名（`alias_name`）。

返回：

Tensor，与输入的 tensor 数学等价。

异常：

- **TypeError** - Reshard 的第一个输入参数需要是 Tensor 类型，但是当前为 `type(tensor)` 类型。
- **TypeError** - Reshard 只支持输入 `mindspore.parallel.Layout` 类型作为 layout 参数，但当前为 `type(layout)` 类型。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 8 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import ops, nn, Tensor, context, Layout
>>> from mindspore.parallel.function import reshard
>>> from mindspore.nn.utils import no_init_parameters
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> from mindspore.communication import init
>>> context.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> class Network(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.matmul = ops.MatMul()
...         self.relu = ops.ReLU()
...     def construct(self, x, layout):
...         x = self.relu(x)
...         x_reshard = reshard(x, layout)
...         y = Tensor(np.ones(shape=(128, 128)), dtype=ms.float32)
...         x = self.matmul(x_reshard, y)
...         return x
>>> layout = Layout((4, 2), ("dp", "mp"))
>>> input_layout = layout("dp", "mp")
>>> with no_init_parameters():
...     net = Network()
>>> parallel_net = AutoParallel(net, parallel_mode='sharding_propagation')
>>> tensor = Tensor(np.ones(shape=(128, 128)), dtype=ms.float32)
>>> out = parallel_net(tensor, input_layout)
```

8.3.2 mindspore.parallel.Layout

class mindspore.parallel.**Layout** (*device_matrix, alias_name, rank_list=None*)

描述集群设备的拓扑抽象，用于张量分片在集群上的放置。

说明：

- 仅在半自动并行或自动并行模式下有效。
- *device_matrix* 的累乘结果必须等于一个 *pipeline stage* 中的设备数。
- 当 *Layout* 来构建切分策略时，每个别名只允许用于一次张量的切分。

参数：

- **device_matrix** (tuple) - 描述设备排列的形状，其元素类型为 int。
- **alias_name** (tuple) - *device_matrix* 的每个轴的别名，其元素类型为字符串。使用 *interleaved_parallel* 作为别名时，会在其对应的切分维度将该算子在单卡内拆分为多个副本。
- **rank_list** (list, 可选) - 数据根据 *rank_list* 排布在设备上。默认 None。

异常：

- **TypeError** - *device_matrix* 不是 tuple 类型。
- **TypeError** - *alias_name* 不是 tuple 类型。
- **TypeError** - *rank_list* 不是 list 类型。
- **ValueError** - *device_matrix* 长度不等于 *alias_name* 长度。
- **TypeError** - *device_matrix* 的元素不是 int 类型。
- **TypeError** - *alias_name* 的元素不是 str 类型。
- **TypeError** - *rank_list* 的元素不是 int 类型。
- **ValueError** - *alias_name* 的元素是一个空的 str。
- **ValueError** - *alias_name* 的元素为 None。
- **ValueError** - *alias_name* 包含重复的元素。

支持平台：

Ascend

样例：

```
>>> from mindspore.parallel import Layout
>>> layout = Layout((2, 2, 2), ("dp", "sp", "mp"))
>>> layout0 = layout("dp", "mp")
>>> print(layout0.to_dict())
{"device_matrix": (2, 2, 2), "tensor_map": (2, 0), "interleaved_parallel": False,
'alias_name': {'dp', 'sp', 'mp'}, "rank_list": [0, 1, 2, 3, 4, 1, 6, 7]}
>>> # Total device num is 4, but split the tensor in local device into two copies.
>>> layout = Layout((2, 2, 2), ("dp", "sp", "interleaved_parallel"))
>>> layout1 = layout(("dp", "interleaved_parallel"), "sp")
```

to_dict()

将 Layout 转换为词典。

8.3.3 mindspore.parallel.shard

`mindspore.parallel.shard(fn, in_strategy, out_strategy=None, parameter_plan=None, device='Ascend', level=0)`

指定一个 `Cell` 或函数的输入、输出切分策略。在 `PyNative` 模式下，可以利用此方法指定某个 `Cell` 以图模式进行分布式执行。在图模式下，可以利用此方法设置某个模块的分布式切分策略，未设置的会自动通过策略传播方式配置。`in_strategy` 和 `out_strategy` 需要为元组类型，其中的每一个元素指定对应的输入/输出的 `Tensor` 分布策略，可参考：[mindspore.ops.Primitive.shard\(\)](#) 的描述。也可以设置为 `None`，会默认以数据并行执行。其余算子的并行策略由输入输出指定的策略推导得到。

说明：

- 调用该方法后，并行模式 (`parallel_mode`) 会自动设置为“`auto_parallel`”且搜索模式 (`search_mode`) 自动设置为“`sharding_propagation`”。
- 如果输入含有 `Parameter`，其对应的策略应该在 `in_strategy` 里设置。
- 该方法目前不支持动态 `shape`。

参数：

- **fn** (`Union[Cell, Function]`) - 待通过分布式并行执行的函数，它的参数和返回值类型应该均为 `Tensor`。如果 `fn` 是 `Cell` 类型且含有参数，则 `fn` 必须是一个实例化的对象，否则无法访问到其内部参数。
- **in_strategy** (`tuple`) - 指定各输入的切分策略，输入元组的每个元素可以为整数元组或 `mindspore.parallel.Layout` 的元组。元组即具体指定输入每一维的切分策略。
- **out_strategy** (`Union[tuple, None]`，可选) - 指定各输出的切分策略，用法同 `in_strategy`，目前未使能。默认值：`None`。
- **parameter_plan** (`Union[dict, None]`，可选) - 指定各参数的切分策略，传入字典时，格式为“参数名: 布局”。键是 `str` 类型的参数名，值是一维整数 `tuple` 或一维 `mindspore.parallel.Layout` 的 `tuple` 表示相应的切分策略。如果参数名错误或对应参数已经设置了切分策略，该参数的设置会被跳过。该参数只支持对 `cell` 类型且含有参数的 `fn` 设置。默认值：`None`。
- **device** (`str`，可选) - 指定执行设备，可以为 [`"CPU"`, `"GPU"`, `"Ascend"`] 中任意一个，目前未使能。默认值：`Ascend`。
- **level** (`int`，可选) - 指定搜索切分策略的目标函数，即是最大化计算通信比、最小化内存消耗、最大化执行速度等。可以为 [`0`, `1`, `2`] 中任意一个，默认值：`0`。目前仅支持最大化计算通信比，其余模式未使能。

返回：

`Function`，返回一个在自动并行流程下执行的函数。

异常：

- **AssertionError** - 如果并行模式不是“`auto_parallel`”或“`semi_auto_parallel`”。
- **AssertionError** - 如果后端不是“`Ascend`”或“`GPU`”。
- **TypeError** - 如果 `in_strategy` 不是 `tuple`。
- **TypeError** - 如果 `out_strategy` 不是 `tuple` 或 `None`。
- **TypeError** - 如果 `in_strategy` 里的任何一个元素不是 `tuple(int)` 或者 `tuple(mindspore.parallel.Layout)`。
- **TypeError** - 如果 `out_strategy` 里的任何一个元素不是 `tuple(int)` 或者 `tuple(mindspore.parallel.Layout)`。
- **TypeError** - 如果 `parameter_plan` 不是 `dict` 或 `None`。
- **TypeError** - 如果 `parameter_plan` 里的任何一个键值类型不是 `str`。
- **TypeError** - 如果 `parameter_plan` 里的任何一个值类型不是 `tuple(int)` 或者 `tuple(mindspore.parallel.Layout)`。

- **TypeError** - 如果 *device* 不是 str。
- **TypeError** - 如果 *level* 不是 int。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, nn, ops
>>> from mindspore.communication import init
>>> from mindspore.parallel import shard
>>> from mindspore.parallel import Layout
>>> from mindspore.nn.utils import no_init_parameters
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>>
>>> # Case 1: cell uses functional
>>> class BasicBlock(nn.Cell):
>>>     def __init__(self):
>>>         super(BasicBlock, self).__init__()
>>>         self.dense1 = nn.Dense(64, 64)
>>>         self.gelu = nn.GELU()
>>>         def my_add(x, y):
>>>             x = ops.abs(x)
>>>             return x + y
>>>         # shard a function with tuple(int) strategies
>>>         self.shard_my_add = shard(my_add, in_strategy=((2, 2), (1, 4)), out_strategy=((4,
↳ 1),))
>>>
>>>     def construct(self, x, u):
>>>         x = self.gelu(x)
>>>         y = self.gelu(u)
>>>         y = x * y
>>>         x = self.dense1(x)
>>>         x = self.shard_my_add(x, y)
>>>         return x
>>>
>>> class NetForward(nn.Cell):
>>>     def __init__(self):
>>>         super(NetForward, self).__init__()
>>>         self.block1 = BasicBlock()
>>>         self.block2 = BasicBlock()
>>>         self.matmul = ops.MatMul()
>>>
>>>     def construct(self, x, y):
>>>         x = self.matmul(x, y)
>>>         x = self.block1(x, x)
>>>         x = self.block2(x, x)
>>>         return x
>>>
>>> class Net(nn.Cell):
>>>     def __init__(self):
>>>         super(Net, self).__init__()
>>>         # setting cell sharding strategy and parameter_plan by tuple(int)
```

(下页继续)

(续上页)

```

>>> self.layer_net1 = NetForward()
>>> self.layer_net1_shard = shard(self.layer_net1, in_strategy=((4, 2), (2, 1)),
...                               parameter_plan={"self.layer_net1.block1.weight":
↪ (4, 1)})
>>>
>>> # setting cell sharding strategy and parameter_plan by tuple(ms.Layout)
>>> self.layer_net2 = NetForward()
>>> layout = Layout((4, 2, 1), ("dp", "mp", "sp"))
>>> in_layout = (layout("dp", "mp"), layout("mp", "sp"))
>>> param_layout = layout("dp", "sp")
>>> self.layer_net2_shard = shard(self.layer_net2, in_strategy=in_layout,
...                               parameter_plan={"self.layer_net2.block2.weight":
↪ param_layout})
>>> self.flatten = nn.Flatten()
>>> self.layer1 = nn.Dense(64, 64)
>>> self.layer2 = nn.Dense(64, 32)
>>> self.add = ops.Add()
>>> self.matmul = ops.MatMul()
>>>
>>> def construct(self, x, y):
>>>     x = self.flatten(x)
>>>     y = self.flatten(y)
>>>     x = self.layer1(x)
>>>     x = self.layer_net1_shard(x, y)
>>>     x = self.layer_net2_shard(x, y)
>>>     x = self.layer2(x)
>>>     x = self.matmul(x, Tensor(np.ones(shape=(32, 32)), dtype=ms.float32))
>>>     return x
>>>
>>> with no_init_parameters():
>>>     net = Net()
>>> x = Tensor(np.ones(shape=(64, 1, 8, 8)), dtype=ms.float32)
>>> y = Tensor(np.ones(shape=(64, 1, 8, 8)), dtype=ms.float32)
>>> parallel_net = AutoParallel(net, parallel_mode='sharding_propagation')
>>> parallel_net(x, y)
>>>
>>> # Case 2: function uses functional sharding
>>> def test_shard(x, y):
...     return x + y
>>> x = Tensor(np.ones(shape=(32, 10)), dtype=ms.float32)
>>> y = Tensor(np.ones(shape=(32, 10)), dtype=ms.float32)
>>> output = shard(test_shard, in_strategy=((4, 2), (4, 2)))(x, y)
>>> print(output.shape)
(32, 10)

```

8.4 其他

<code>mindspore.parallel.build_searched_strategy</code>	从策略文件中提取网络中每个参数的切分策略，用于分布式推理的场景。
<code>mindspore.parallel.merge_pipeline_strategys</code>	汇聚所有流水线并行子图的切分策略文件到目的文件。

[下页继续](#)

表 4 - 续上页

<code>mindspore.parallel.parameter_broadcast</code>	在数据并行维度，将参数广播到其他卡上。
<code>mindspore.parallel.restore_group_info_list</code>	从通信域文件中提取 rank list 信息。
<code>mindspore.parallel.sync_pipeline_shared_parameters</code>	在流水线并行推理场景下，对 stage 间的共享权重进行同步。

8.4.1 mindspore.parallel.build_searched_strategy

`mindspore.parallel.build_searched_strategy(strategy_filename)`
 从策略文件中提取网络中每个参数的切分策略，用于分布式推理的场景。

参数：

- **strategy_filename** (str) - 策略文件的名称。

返回：

Dict, key 为参数名，value 为该参数的切片策略。

异常：

- **ValueError** - 策略文件不正确。
- **TypeError** - `strategy_filename` 不是字符串类型。

支持平台：

Ascend

样例：

```
>>> from mindspore.parallel import build_searched_strategy
>>> strategy = build_searched_strategy("./strategy_train.ckpt")
```

8.4.2 mindspore.parallel.merge_pipeline_strategys

`mindspore.parallel.merge_pipeline_strategys(src_strategy_dirs, dst_strategy_file)`
 汇聚所有流水线并行子图的切分策略文件到目的文件。

说明：src_strategy_dirs 必须包含所有流水线并行的子图的切分策略文件。

参数：

- **src_strategy_dirs** (str) - 包含所有流水线并行的子图的切分策略文件的目录，切分策略文件由 `mindspore.parallel.auto_parallel.AutoParallel.save_param_strategy_file()` 接口存储得到。
- **dst_strategy_file** (str) - 保存汇聚后的切分策略的文件路径。

异常：

- **NotADirectoryError** - `src_strategy_dirs` 不是一个目录。

样例：

```
>>> import mindspore as ms
>>> # src_strategy_dir/stra0.ckpt, src_strategy_dir/stra1.ckpt ... src_strategy_dir/stra127.
    ↪ckpt
>>> ms.parallel.merge_pipeline_strategys("./src_strategy_dir", "./dst_strategy.ckpt")
```

8.4.3 mindspore.parallel.parameter_broadcast

`mindspore.parallel.parameter_broadcast` (*net, layout, cur_rank=0, initial_rank=0*)

在数据并行维度，将参数广播到其他卡上。

警告：实验性接口，未来可能变更或移除。

参数：

- **net** (Cell) - 将进行参数广播的网络。
- **layout** (Dict) - 参数 `layout` 字典，来自于函数 `mindspore.nn.Cell.parameter_layout_dict()`；也可以从文件中读取。例如，通过 `mindspore.parallel.auto_parallel.AutoParallel.save_param_strategy_file()` 保存下来的 `strategy.ckpt`。该字典的 `key` 是参数名称，`value` 是参数的 `Layout`。
- **cur_rank** (int, 可选) - 当前的 `rankID`。默认值：0。
- **initial_rank** (int, 可选) - 每个流水线并行阶段的起始 `rankID`。默认值：0。

异常：

- **ValueError** - `cur_rank` 不是当前 `rank` 的 `rankID`。
- **ValueError** - `initial_rank` 不是当前流水线并行阶段的起始 `rankID`。
- **ValueError** - `layout` 中的参数名称不在函数 `mindspore.nn.Cell.parameters_dict()`。

支持平台：

Ascend

样例：

```
>>> import os
>>> import mindspore as ms
>>> import mindspore.dataset as ds
>>> from mindspore import nn, ops
>>> from mindspore.communication import init
>>> from mindspore.common.initializer import initializer
>>> from mindspore.train import Model
>>> from mindspore.train.serialization import load_checkpoint, load_param_into_net
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> from mindspore.parallel import parameter_broadcast
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> ms.runtime.set_memory(max_size="28GB")
>>> init()
>>> ms.set_seed(1)
>>> class Network(nn.Cell):
...     def __init__(self):
...         super().__init__()
```

(下页继续)

(续上页)

```

...     self.flatten = ops.Flatten()
...     self.fc1_weight = ms.Parameter(initializer("normal", [28*28, 512], ms.float32))
...     self.fc2_weight = ms.Parameter(initializer("normal", [512, 512], ms.float32))
...     self.fc3_weight = ms.Parameter(initializer("normal", [512, 10], ms.float32))
...     self.matmul1 = ops.MatMul()
...     self.relu1 = ops.ReLU()
...     self.matmul2 = ops.MatMul()
...     self.relu2 = ops.ReLU()
...     self.matmul3 = ops.MatMul()
...     def construct(self, x):
...         x = self.flatten(x)
...         x = self.matmul1(x, self.fc1_weight)
...         x = self.relu1(x)
...         x = self.matmul2(x, self.fc2_weight)
...         x = self.relu2(x)
...         logits = self.matmul3(x, self.fc3_weight)
...         return logits
>>> net = Network()
>>> net.matmul1.shard(((2, 4), (4, 1)))
>>> net.relu1.shard(((4, 1),))
>>> net.matmul2.shard(((1, 8), (8, 1)))
>>> net.relu2.shard(((8, 1),))
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> optim = nn.SGD(net.trainable_params(), 1e-2)
>>> loss = nn.CrossEntropyLoss()
>>> parallel_net = AutoParallel(net)
>>> model = Model(parallel_net, loss_fn=loss, optimizer=optim)
>>> model.train(1, dataset)
>>> ms.save_checkpoint(net, "./simple.ckpt", False)
>>> layout = model.train_network.parameter_layout_dict
>>> param_dict = load_checkpoint("./simple.ckpt")
>>> load_param_into_net(net, param_dict)
>>> rank_id = os.environ["RANK_ID"]
>>> parameter_broadcast(model.train_network, layout, int(rank_id), 0)
>>> class LossCallBack(Callback):
...     def step_end(self, run_context):
...         cb_params = run_context.original_args()
...         print("step end, cur step num: ", cb_params.cur_step_num, flush=True)
>>> model.train(1, dataset, callbacks=[LossCallBack()])

```

8.4.4 mindspore.parallel.restore_group_info_list

`mindspore.parallel.restore_group_info_list(group_info_file_name)`

从通信域文件中提取 rank list 信息。通过配置环境变量 `GROUP_INFO_FILE` 以在编译阶段存储下该通信域信息，例如“`export GROUP_INFO_FILE=/data/group_info.pb`”。

参数：

- **group_info_file_name** (str) - 保存通信域的文件的名字。

返回：

List，通信域列表。

异常：

- **ValueError** - 通信域文件格式不正确。
- **TypeError** - `group_info_file_name` 不是字符串。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore.parallel import restore_group_info_list
>>> ms.restore_list = restore_group_info_list("./group_info.pb")
```

8.4.5 mindspore.parallel.sync_pipeline_shared_parameters

`mindspore.parallel.sync_pipeline_shared_parameters` (*net*)

在流水线并行推理场景下，对 stage 间的共享权重进行同步。例如 *embedding table* 被 *VocabEmbedding* 和 *LMHead* 两层共享，这两层通常会被切分到不同的 stage 上。在流水线并行推理时，*embedding table* 变更后，有必要在 stage 之间进行权重同步。

说明：网络需要先编译，再执行 stage 之间权重同步。

参数：

- **net** (Cell) - 推理网络。

异常：

- **TypeError** - *net* 不是 *Cell* 的类型。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend 设备，用户需要编写动态组网启动脚本，详见 [动态组网启动](#)。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication.management as D
>>> from mindspore import lazy_inline, context, nn, ops, Parameter, Tensor
>>> from mindspore.parallel.auto_parallel import AutoParallel
>>> context.set_context(mode=context.GRAPH_MODE)
>>> class Embedding(nn.Cell):
...     def __init__(self, shape):
...         super().__init__()
...         self.w = Parameter(Tensor(np.ones(shape), ms.float32), name='w')
...         self.matmul = ops.MatMul().shard(((1, 1), (1, 1)))
...     def construct(self, x):
...         return self.matmul(x, self.w), self.w
... 
```

(下页继续)

(续上页)

```

>>> class LMHead(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.matmul = ops.MatMul(transpose_b=True).shard(((1, 1), (1, 1)))
...     def construct(self, x, w):
...         return self.matmul(x, w)
...
>>> class Network(nn.Cell):
...     @lazy_inline
...     def __init__(self):
...         super().__init__()
...         shape = (4, 4)
...         self.word_embedding = Embedding(shape)
...         self.lm_head = LMHead()
...         self.word_embedding.pipeline_stage = 0
...         self.lm_head.pipeline_stage = 1
...     def construct(self, x):
...         x, embed = self.word_embedding(x)
...         return self.lm_head(x, embed)
...
>>> class PipelineCellInference(nn.Cell):
...     def __init__(self, network, micro_batch_num):
...         super().__init__()
...         self.network = network
...         self.micro_batch_num = micro_batch_num
...         self.concat = ops.Concat()
...     def construct(self, x):
...         ret = ()
...         for i in range(self.micro_batch_num):
...             micro_batch_size = x.shape[0] // self.micro_batch_num
...             start = micro_batch_size * i
...             end = micro_batch_size * (i + 1)
...             micro_input = x[start:end]
...             y = self.network(micro_input)
...             ret = ret + (y,)
...         ret = self.concat(ret)
...         return ret
>>> D.init()
>>> net = Network()
>>> net = PipelineCellInference(net, 2)
>>> net.set_train(False)
>>> x = Tensor(np.ones((2, 4)), ms.float32)
>>> net.compile(x)
>>> pp_net = AutoParallel(net, parallel_mode="semi_auto")
>>> pp_net.full_batch = True
>>> pp_net.pipeline(stages=2, scheduler="1f1b")
>>> ms.parallel.sync_pipeline_shared_parameters(pp_net)
>>> print(pp_net.network.network.word_embedding.w.asnumpy())
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

```


运行时封装了执行、内存、流、事件的接口。MindSpore 从不同的后端抽象出相应的模块，允许用户在 Python 层调度硬件资源。

9.1 执行

<code>mindspore.runtime.set_cpu_affinity</code>	使能线程级绑核功能，给 MindSpore 的主要模块（主线程、pynative、runtime、minddata）分配特定的 CPU 核，防止 MindSpore 线程抢占 CPU 导致性能不稳定的情况。
<code>mindspore.runtime.launch_blocking</code>	表示启动设备同步执行，会降低程序的执行性能。
<code>mindspore.runtime.dispatch_threads_num</code>	设置运行时要使用的线程数。
<code>mindspore.runtime.set _kernel_launch_group</code>	O0 模式支持算子批量并行下发接口，支持开启并行下发，并配置并行数。

9.1.1 mindspore.runtime.set_cpu_affinity

`mindspore.runtime.set_cpu_affinity(enable_affinity, affinity_cpu_list=None)`

使能线程级绑核功能，给 MindSpore 的主要模块（主线程、pynative、runtime、minddata）分配特定的 CPU 核，防止 MindSpore 线程抢占 CPU 导致性能不稳定的情况。

说明：

- 提供两种绑核模式：
 1. 依据环境上可用的 CPU、NUMA 节点和设备资源，自动化生成绑核策略，进行线程级绑核。
 2. 依据 `affinity_cpu_list` 传入的自定义绑核策略，进行线程级绑核。
- 自动化生成绑核策略场景会调用系统命令去获取环境上的 CPU、NUMA 节点和设备资源，由于环境差异，一些命令无法成功执行；按照环境上可获取资源，生成的自动化绑核策略会有所差异：

1. `cat /sys/fs/cgroup/cpuset/cpuset.cpus`, 获取环境上可用的 CPU 资源; 若执行该命令失败, 绑核功能不会生效。
2. `npu-smi info -m`, 获取环境上可用的 NPU 资源; 若执行该命令失败, 仅根据可用 CPU 资源去生成绑核策略, 不考虑设备亲和性。
3. `npu-smi info -t board -i {NPU_ID} -c {CHIP_ID}`, 根据设备逻辑 ID 去获取 NPU 的详细信息; 若执行该命令失败, 仅根据可用 CPU 资源去生成绑核策略, 不考虑设备亲和性。
4. `lspci -s {PCIe_No} -vvv`, 获取环境上设备的硬件信息; 若执行该命令失败, 仅根据可用 CPU 资源去生成绑核策略, 不考虑设备亲和性。
5. `lscpu`, 获取环境上 CPU 与 NUMA 节点的信息; 若执行该命令失败, 仅根据可用 CPU 资源去生成绑核策略, 不考虑设备亲和性。

参数:

- **enable_affinity** (bool) - 开关线程级绑核功能。
- **affinity_cpu_list** (dict, 可选) - 指定自定义的绑核策略。传入字典的 **key** 需要为字符串 "deviceX" 格式, **value** 需要为列表 ["cpuidX-cpuidY"] 格式。默认值: None, 即使用依据环境自动化生成的绑核策略。允许传入空字典 {}, 这种情况下会依据环境自动化生成的绑核策略。

异常:

- **TypeError** - 参数 `enable_affinity` 不是 bool。
- **TypeError** - 参数 `affinity_cpu_list` 既不是字典, 也不是 None。
- **ValueError** - 参数 `affinity_cpu_list` 的 **key** 不是字符串。
- **ValueError** - 参数 `affinity_cpu_list` 的 **key** 不符合 "deviceX" 格式。
- **ValueError** - 参数 `affinity_cpu_list` 的 **value** 不是列表。
- **ValueError** - 参数 `affinity_cpu_list` 的 **value** 中的元素不是字符串。
- **ValueError** - 参数 `affinity_cpu_list` 的 **value** 中的元素不符合 ["cpuidX-cpuidY"] 格式。
- **RuntimeError** - 自动化生成绑核策略或者自定义指定绑核策略场景, 分配给每个设备的 CPU 核数量小于 7 个。
- **RuntimeError** - 自定义指定绑核策略场景, 分配给某个设备的 CPU 在环境中不可用。
- **RuntimeError** - 重复调用了 `mindspore.runtime.set_cpu_affinity()` 接口。

样例:

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
>>> ms.runtime.set_cpu_affinity(True)
>>>
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
>>> ms.runtime.set_cpu_affinity(True, {"device0": ["0-9"], "device1": ["10-15", "20-29"], "device2": ["35-45"]})
```


9.1.2 mindspore.runtime.launch_blocking

`mindspore.runtime.launch_blocking()`

表示启动设备同步执行，会降低程序的执行性能。

- 未调用此接口的初始状态下，算子会在设备上以异步方式执行。此时，当算子执行出错时，将无法定位特定错误脚本代码的位置。
- 调用此接口后，算子会在设备上以同步方式执行。此时，当算子执行出错时，可以根据错误的调用栈来定位错误脚本代码的位置。

样例：

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
>>> ms.runtime.launch_blocking()
```

9.1.3 mindspore.runtime.dispatch_threads_num

`mindspore.runtime.dispatch_threads_num(threads_num)`

设置运行时要使用的线程数。

框架默认运行时的线程数为 5。

参数：

- **threads_num** (int) - 要设置的线程数。

样例：

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
>>> ms.runtime.dispatch_threads_num(6)
```

9.1.4 mindspore.runtime.set_kernel_launch_group

`mindspore.runtime.set_kernel_launch_group(thread_num=2, kernel_group_num=8)`

O0 模式支持算子批量并行下发接口，支持开启并行下发，并配置并行数。

参数：

- **thread_num** (int, 可选) - 并发线程数，一般不建议增加。和现有接口 `mindspore.runtime.dispatch_threads_num` 配置的线程数相互独立。默认值：2。
- **kernel_group_num** (int, 可选) - 算子分组总数量，每线程 `kernel_group_num/thread_num` 个组，默认值：8。

样例：

```
>>> import mindspore as ms
>>> ms.runtime.set_kernel_launch_group(thread_num=2, kernel_group_num=8)
```

9.2 内存

<code>mindspore.runtime.max_memory_allocated</code>	返回从进程启动开始，内存池实际被 Tensor 占用的内存大小的峰值。
<code>mindspore.runtime.max_memory_reserved</code>	返回从进程启动开始，内存池管理的内存总量的峰值。
<code>mindspore.runtime.memory_allocated</code>	返回当前实际被 Tensor 占用的内存大小。
<code>mindspore.runtime.memory_reserved</code>	返回内存池当前管理的内存总量。
<code>mindspore.runtime.memory_stats</code>	返回从内存池查询到的状态信息。
<code>mindspore.runtime.memory_summary</code>	返回可读的内存池状态信息。
<code>mindspore.runtime.reset_max_memory_reserved</code>	重置被内存池所管理的内存峰值。
<code>mindspore.runtime.reset_max_memory_allocated</code>	重置内存池实际被 Tensor 占用的内存大小的峰值。
<code>mindspore.runtime.reset_peak_memory_stats</code>	重置内存池中的全部峰值。
<code>mindspore.runtime.empty_cache</code>	清理内存池中的内存碎片，优化内存排布。
<code>mindspore.runtime.set_memory</code>	设置使用内存池实现的运行时设备内存管理的内存参数。

9.2.1 mindspore.runtime.max_memory_allocated

`mindspore.runtime.max_memory_allocated()`
返回从进程启动开始，内存池实际被 **Tensor** 占用的内存大小的峰值。

- 说明：
- 对于 *CPU* 硬件，固定返回 0。

返回：
`int`，单位为 `Byte`。

支持平台：
Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.max_memory_allocated())
1536
```

9.2.2 mindspore.runtime.max_memory_reserved

`mindspore.runtime.max_memory_reserved()`
 返回从进程启动开始，内存池管理的内存总量的峰值。

说明：

- 对于 *CPU* 硬件，固定返回 0。

返回：

int，单位为 Byte。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.max_memory_reserved())
1073741824
```

9.2.3 mindspore.runtime.memory_allocated

`mindspore.runtime.memory_allocated()`
 返回当前实际被 Tensor 占用的内存大小。

说明：

- 对于 *CPU* 硬件，固定返回 0。

返回：

int，单位为 Byte。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
```

(下页继续)

(续上页)

```
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.memory_allocated())
1024
```

9.2.4 mindspore.runtime.memory_reserved

`mindspore.runtime.memory_reserved()`

返回内存池当前管理的内存总量。

说明:

- 对于 *CPU* 后端, 固定返回 0。
-

返回:

`int`, 单位为 `Byte`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.memory_reserved())
1073741824
```

9.2.5 mindspore.runtime.memory_stats

`mindspore.runtime.memory_stats()`

返回从内存池查询到的状态信息。

说明: 对于 *CPU* 后端, 固定返回数据为空的字典。

返回:

`dict`, 查询到的内存信息。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.memory_stats())
{'total_reserved_memory': 1073741824, 'total_allocated_memory': 1024, 'total_idle_memory': 1073740800,
 'total_eager_free_memory': 0, 'max_reserved_memory': 1073741824, 'max_allocated_memory': 1536,
 'common_mem_pool_stats': {'block_unit_size': 1073741824, 'block_counts': 1, 'blocks_info':
 {<capsule object NULL at 0x7f7e8c27b030>: {'block_stream_id': 0, 'block_memory_size': 1073741824}}},
 'persistent_mem_pool_stats': {'block_unit_size': 1073741824, 'block_counts': 0, 'blocks_info': {}}}
```

9.2.6 mindspore.runtime.memory_summary

`mindspore.runtime.memory_summary()`

返回可读的内存池状态信息。

返回:

str, 表格形式的可读内存池状态信息。

支持平台:

Ascend GPU CPU

9.2.7 mindspore.runtime.reset_max_memory_reserved

`mindspore.runtime.reset_max_memory_reserved()`

重置被内存池所管理的内存峰值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.max_memory_reserved())
1073741824
>>> ms.runtime.reset_max_memory_reserved()
>>> print(ms.runtime.max_memory_reserved())
0
```

9.2.8 mindspore.runtime.reset_max_memory_allocated

`mindspore.runtime.reset_max_memory_allocated()`

重置内存池实际被 Tensor 占用的内存大小的峰值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.max_memory_allocated())
1536
>>> ms.runtime.reset_max_memory_allocated()
>>> print(ms.runtime.max_memory_allocated())
0
```

9.2.9 mindspore.runtime.reset_peak_memory_stats

`mindspore.runtime.reset_peak_memory_stats()`

重置内存池中的全部峰值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.runtime.max_memory_reserved())
1073741824
>>> print(ms.runtime.max_memory_allocated())
1536
>>> ms.runtime.reset_peak_memory_stats()
>>> print(ms.runtime.max_memory_reserved())
0
>>> print(ms.runtime.max_memory_allocated())
0
```

9.2.10 mindspore.runtime.empty_cache

`mindspore.runtime.empty_cache()`
清理内存池中的内存碎片，优化内存排布。

说明:

- 目前 MindSpore 内存池没有清空内存碎片的功能，此为预留接口，实现为空方法。使用时，会通过日志方式提示。

9.2.11 mindspore.runtime.set_memory

`mindspore.runtime.set_memory (init_size='2GB', increase_size='2GB', max_size='1024GB', optimize_level='00', huge_page_reserve_size='0GB')`

设置使用内存池实现的运行时设备内存管理的内存参数。

框架默认设置所有参数，如下所示。

参数:

- **init_size** (str) - 内存池初始大小。格式为 `xxGB`，默认 `2GB`。
- **increase_size** (str) - 内存池大小的增量。若当前内存池没有足够的内存的时候，内存池将按此值扩展。格式为 `xxGB`，默认 `2GB`。
- **max_size** (str) - 内存池可用的最大内存。实际使用的内存大小是设备和最大设备内存的可用内存的最小值。格式为 `xxGB`，默认为设备的最大可用内存，表示为 `1024GB`。
- **optimize_level** (str) - 内存优化级别。该值必须在 `[00, 01]` 中，默认 `00`。
- **huge_page_reserve_size** (str) - 大页内存预留大小。格式为 `xxGB`，默认 `0GB`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
>>> ms.runtime.set_memory("10GB", "2GB", "60GB", "01", "0GB")
```

9.3 流

<code>mindspore.runtime.communication_stream</code>	返回此设备上的通信流。
<code>mindspore.runtime.current_stream</code>	返回此设备上正在使用的流。
<code>mindspore.runtime.default_stream</code>	返回此设备上的默认流。
<code>mindspore.runtime.set_cur_stream</code>	设置当前流，这是用于设置流的包装器 API。
<code>mindspore.runtime.synchronize</code>	同步当前设备上的所有流。
<code>mindspore.runtime.Stream</code>	基于设备流的封装器。
<code>mindspore.runtime.StreamCtx</code>	上下文管理器，用于选择给定的流。

9.3.1 mindspore.runtime.communication_stream

`mindspore.runtime.communication_stream()`

返回此设备上的通信流。

返回:

Stream, 此设备上的通信流。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 0)
>>> ms.runtime.communication_stream()
Stream(device_name=Ascend, device_id=0, stream id:1)
```

9.3.2 mindspore.runtime.current_stream

`mindspore.runtime.current_stream()`

返回此设备上正在使用的流。

返回:

Stream, 此设备上正在使用的流。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 0)
>>> cur_stream = ms.runtime.current_stream()
>>> assert cur_stream == ms.runtime.default_stream()
```

9.3.3 mindspore.runtime.default_stream

`mindspore.runtime.default_stream()`

返回此设备上的默认流。

返回:

Stream, 此设备上的默认流。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 0)
>>> cur_stream = ms.runtime.current_stream()
>>> assert cur_stream == ms.runtime.default_stream()
```


9.3.4 mindspore.runtime.set_cur_stream

`mindspore.runtime.set_cur_stream(stream)`

设置当前流，这是用于设置流的包装器 API。

不建议使用此函数，建议使用 *StreamCtx* 上下文管理器。

参数：

- **stream** (Stream) - 指定的流。如果是 None，表示此上下文管理器无操作。

异常：

- **TypeError** - 参数 *stream* 既不是 `mindspore.runtime.Stream` 也不是 None。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 0)
>>> cur_stream = ms.runtime.current_stream()
>>> assert cur_stream == ms.runtime.default_stream()
>>> s1 = ms.runtime.Stream()
>>> ms.runtime.set_cur_stream(s1)
>>> assert ms.runtime.current_stream() == s1
>>> ms.runtime.set_cur_stream(ms.runtime.default_stream())
```

9.3.5 mindspore.runtime.synchronize

`mindspore.runtime.synchronize()`

同步当前设备上的所有流。（每个 MindSpore 进程只占用一个设备）

支持平台：

Ascend GPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1024, 2048]), ms.float32)
>>> b = Tensor(np.ones([2048, 4096]), ms.float32)
>>> s1 = ms.runtime.Stream()
>>> with ms.runtime.StreamCtx(s1):
...     c = ops.matmul(a, b)
>>> ms.runtime.synchronize()
>>> assert s1.query()
```

9.3.6 mindspore.runtime.Stream

class mindspore.runtime.Stream(priority=0, **kwargs)

基于设备流的封装器。设备流是属于特定设备的线性执行序列，流之间相互独立。

参数：

- **priority** (int, 可选) - 流的优先级，较低的数字表示较高的优先级。默认情况下，流的优先级为 0。
- **kwargs** (dict) - 关键字参数字典。

支持平台：

Ascend GPU

query()

检查所有提交的工作是否已完成。

返回：

bool，指示该流中的所有算子是否已执行完成。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1024, 2048]), ms.float32)
>>> b = Tensor(np.ones([2048, 4096]), ms.float32)
>>> s1 = ms.runtime.Stream()
>>> with ms.runtime.StreamCtx(s1):
...     c = ops.matmul(a, b)
>>> s1.synchronize()
>>> assert s1.query()
```

record_event (event=None)

记录一个事件。

参数：

- **event** (Event, 可选) - 要记录的事件。如果输入为 None，将分配一个新的事件。默认值：None。

返回：

Event，记录的事件。

异常：

- **TypeError** - 参数 *event* 既不是一个 *mindspore.runtime.Event* 也不是一个 None。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
```

(下页继续)

(续上页)

```

>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([3, 3]), ms.float32)
>>> b = Tensor(np.ones([3, 3]), ms.float32)
>>> s1 = ms.runtime.Stream()
>>> with ms.runtime.StreamCtx(s1):
...     c = a + b
...     event = s1.record_event()
...     d = a * b
>>> cur_stream = ms.runtime.current_stream()
>>> cur_stream.wait_event(event)
>>> e = c + 3
>>> print(e)
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]

```

synchronize()

等待此流中的所有算子执行完成。

支持平台：

Ascend GPU

样例：

```

>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1024, 2048]), ms.float32)
>>> b = Tensor(np.ones([2048, 4096]), ms.float32)
>>> s1 = ms.runtime.Stream()
>>> with ms.runtime.StreamCtx(s1):
...     c = ops.matmul(a, b)
>>> s1.synchronize()
>>> assert s1.query()

```

wait_event(event)

使提交到流的所有未来工作等待本事件。

参数：

- **event** (Event) - 等待的事件。

异常：

- **TypeError** - 参数 *event* 不是一个 `mindspore.runtime.Event`。

支持平台：

Ascend GPU

样例：

```

>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([3, 3]), ms.float32)
>>> b = Tensor(np.ones([3, 3]), ms.float32)

```

(下页继续)

(续上页)

```

>>> s1 = ms.runtime.Stream()
>>> with ms.runtime.StreamCtx(s1):
...     c = a + b
...     event = s1.record_event()
...     d = a * b
>>> cur_stream = ms.runtime.current_stream()
>>> cur_stream.wait_event(event)
>>> e = c + 3
>>> print(e)
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]

```

wait_stream(stream)

与另一个流同步。

所有提交到本流的未来工作都将等待，直到所有算子都提交至给定流并执行完成。

参数：

- **stream** (Stream) - 需要同步的流。

异常：

- **TypeError** - 参数 *stream* 不是一个 `mindspore.runtime.Stream`。

支持平台：

Ascend GPU

样例：

```

>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> s1 = ms.runtime.Stream()
>>> s2 = ms.runtime.Stream()
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([2, 2]), ms.float32)
>>> with ms.runtime.StreamCtx(s1):
...     c = ops.matmul(a, b)
>>> with ms.runtime.StreamCtx(s2):
...     s2.wait_stream(s1)
...     d = ops.matmul(c, b)
>>> ms.runtime.synchronize()
>>> print(d)
[[4. 4.]]

```

9.3.7 mindspore.runtime.StreamCtx

class mindspore.runtime.**StreamCtx** (*ctx_stream*)

上下文管理器，用于选择给定的流。

上下文范围内的所有算子，都将在指定流上执行。

参数：

- **ctx_stream** (Stream) - 指定的流。如果是 *None*，这个上下文管理器无操作。

异常：

- **TypeError** - 参数 *ctx_stream* 即不是一个 *mindspore.runtime.Stream* 也不是一个 *None*。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1024, 2048]), ms.float32)
>>> b = Tensor(np.ones([2048, 4096]), ms.float32)
>>> s1 = ms.runtime.Stream()
>>> with ms.runtime.StreamCtx(s1):
...     c = ops.matmul(a, b)
>>> ms.runtime.synchronize()
>>> assert s1.query()
```

9.4 事件

mindspore.runtime.Event

设备事件的封装器。

9.4.1 mindspore.runtime.Event

class mindspore.runtime.**Event** (*enable_timing=False, blocking=False*)

设备事件的封装器。

设备事件是同步标记，可用于监视设备的执行进度、准确计时和同步设备流。

当事件首次被记录时，底层设备事件才会被初始化。

参数：

- **enable_timing** (bool, 可选) - 事件是否应计时。(默认值: *False*)
- **blocking** (bool, 可选) - 如果为 *True*，*wait* 函数将是阻塞的。(默认值: *False*)

支持平台：

Ascend GPU

样例：

```

>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> start = ms.runtime.Event(enable_timing=True)
>>> end = ms.runtime.Event(enable_timing=True)
>>> s1 = ms.runtime.Stream()
>>> s2 = ms.runtime.Stream()
>>> a = Tensor(np.ones([2, 2]), ms.float32)
>>> b = Tensor(np.ones([2, 2]), ms.float32)
>>> c = Tensor(np.ones([2, 2]), ms.float32)
>>> with ms.runtime.StreamCtx(s1):
...     d = ops.matmul(a, b)
...     start.record()
>>> c += 2
>>> end.record()
>>> with ms.runtime.StreamCtx(s2):
...     start.synchronize()
...     end.synchronize()
...     e = c + d
>>> ms.runtime.synchronize()
>>> print(e)
[[5. 5.]
 [5. 5.]]
>>> elapsed_time = start.elapsed_time(end)

```

elapsed_time(end_event)

返回记录事件之后到记录 `end_event` 之前所用的时间（以毫秒为单位）。

参数：

- **end_event** (Event) - 结束事件。

返回：

float, 经过的时间（以毫秒为单位）。

异常：

- **TypeError** - 参数 `end_event` 不是一个 `mindspore.runtime.Event`。

query()

检查事件当前捕获的所有工作是否已完成。

返回：

bool, 指示事件当前捕获的所有工作是否都已完成。

支持平台：

Ascend GPU

样例：

```

>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> a = Tensor(np.ones([1024, 2048]), ms.float32)
>>> b = Tensor(np.ones([2048, 4096]), ms.float32)
>>> s1 = ms.runtime.Stream()

```

(下页继续)

(续上页)

```
>>> with ms.runtime.StreamCtx(s1):
...     c = ops.matmul(a, b)
...     ev = s1.record_event()
>>> s1.synchronize()
>>> assert ev.query()
```

record (*stream=None*)

在给定的流中记录事件。

如果未指定 *stream*, 将使用 `mindspore.runtime.current_stream()`。*stream* 的设备必须与 *event* 的设备匹配。

参数:

- **stream** (Stream, 可选) - 需要记录的流。如果输入为 None, 将使用当前流。默认值: None。

异常:

- **TypeError** - 参数 *stream* 既不是一个 `mindspore.runtime.Stream` 也不是一个 None。

synchronize ()

等待事件完成。

等待直到完成当前此事件捕获的所有工作。这将阻止 CPU 线程继续进行, 直到事件完成。

wait (*stream=None*)

使提交给给定流的所有未来工作等待此事件。

如果未指定 *stream*, 将使用 `mindspore.runtime.current_stream()`。

参数:

- **stream** (Stream, 可选) - 需要等待的流。如果输入为 None, 将使用当前流。默认值: None。

异常:

- **TypeError** - 参数 *stream* 既不是一个 `mindspore.runtime.Stream` 也不是一个 None。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.set_device("Ascend", 0)
>>> event = ms.runtime.Event()
>>> s1 = ms.runtime.Stream()
>>> s2 = ms.runtime.Stream()
>>> a = Tensor(np.ones([2, 2]), ms.float32)
>>> b = Tensor(np.ones([2, 2]), ms.float32)
>>> with ms.runtime.StreamCtx(s1):
...     c = ops.matmul(a, b)
...     event.record()
>>> event.wait()
>>> d = c + 2
>>> ms.runtime.synchronize()
>>> print(d)
[[4. 4.]
 [4. 4.]]
```

mindspore.device_context

`device_context` 中封装了设备数量查询与当前指定后端是否可用的接口。

10.1 CPU 设备后端管理

<code>mindspore.device_context.cpu.device_count</code>	返回可用的 CPU 数量。
<code>mindspore.device_context.cpu.is_available</code>	返回 CPU 设备是否可用的指示。
<code>mindspore.device_context.cpu.op_tuning.threads_num</code>	设置 CPU 算子使用的线程数。

10.1.1 mindspore.device_context.cpu.device_count

`mindspore.device_context.cpu.device_count()`

返回可用的 CPU 数量。

说明:

- 对于 "CPU" 硬件, 固定返回 1。
-

返回:

int, 可用的 CPU 数量。

样例:

```
>>> import mindspore as ms
>>> print(ms.device_context.cpu.device_count())
1
```

10.1.2 mindspore.device_context.cpu.is_available

`mindspore.device_context.cpu.is_available()`
返回 CPU 设备是否可用的指示。

返回：
bool, 指示 CPU 设备是否可用。

样例：

```
>>> import mindspore as ms
>>> print(ms.device_context.cpu.is_available())
True
```

10.1.3 mindspore.device_context.cpu.op_tuning.threads_num

`mindspore.device_context.cpu.op_tuning.threads_num(num)`
设置 CPU 算子使用的线程数。

框架默认设置的线程数为 25 。

参数：
• **num** (int) - CPU 算子使用的线程数。

样例：

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 1)
>>> ms.device_context.cpu.op_tuning.threads_num(10)
```

10.2 GPU 设备后端管理

<code>mindspore.device_context.gpu.device_count</code>	返回可用的 GPU 数量。
<code>mindspore.device_context.gpu.is_available</code>	返回 GPU 设备是否可用的指示。
<code>mindspore.device_context.gpu.op_precision.conv_allow_tf32</code>	是否开启卷积在 cuDNN 下的 TF32 张量核计算。
<code>mindspore.device_context.gpu.op_precision.matmul_allow_tf32</code>	是否开启矩阵乘在 CUBLAS 下的 TF32 张量核计算。
<code>mindspore.device_context.gpu.op_tuning.conv_dgrad_algo</code>	指定 cuDNN 的卷积输入数据的反向算法。
<code>mindspore.device_context.gpu.op_tuning.conv_fprop_algo</code>	指定 cuDNN 的卷积前向算法。
<code>mindspore.device_context.gpu.op_tuning.conv_wgrad_algo</code>	指定 cuDNN 的卷积输入卷积核的反向算法。

10.2.1 mindspore.device_context.gpu.device_count

`mindspore.device_context.gpu.device_count()`

返回可用的 GPU 数量。

返回:

int, 可用的 GPU 数量。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.gpu.device_count()
```

10.2.2 mindspore.device_context.gpu.is_available

`mindspore.device_context.gpu.is_available()`

返回 GPU 设备是否可用的指示。

返回:

bool, 指示 GPU 设备是否可用。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.gpu.device.is_available()
```

10.2.3 mindspore.device_context.gpu.op_precision.conv_allow_tf32

`mindspore.device_context.gpu.op_precision.conv_allow_tf32(value)`

是否开启卷积在 cuDNN 下的 TF32 张量核计算。详细信息请查看 [nvidia 关于 CUBLAS_COMPUTE_32F_FAST_TF32 的说明](#)。

参数:

- **value** (bool) - 是否开启卷积在 cuDNN 下的 TF32 张量核计算。如未配置, 框架默认为 True。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.gpu.op_precision.conv_allow_tf32(False)
```

10.2.4 mindspore.device_context.gpu.op_precision.matmul_allow_tf32

`mindspore.device_context.gpu.op_precision.matmul_allow_tf32(value)`

是否开启矩阵乘在 CUBLAS 下的 TF32 张量核计算。详细信息请查看 [nvidia 关于 CUBLAS_COMPUTE_32F_FAST_TF32 的说明](#)。

参数:

- **value** (bool) - 是否开启矩阵乘在 CUBLAS 下的 TF32 张量核计算。如未配置, 框架默认为 False。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.gpu.op_precision.matmul_allow_tf32(True)
```

10.2.5 mindspore.device_context.gpu.op_tuning.conv_dgrad_algo

`mindspore.device_context.gpu.op_tuning.conv_dgrad_algo(mode)`

指定 cuDNN 的卷积输入数据的反向算法。详细信息请查看 [NVIDIA cuDNN 关于 cudnnConvolutionBackwardData 的说明](#)。

参数：

- **mode** (str) - cuDNN 的卷积输入数据的反向算法。如未配置，框架默认为 `normal`。其值范围如下：
 - `normal`：使用 cuDNN 自带的启发式搜索算法，会根据卷积形状和类型快速选择合适的卷积算法。该参数不保证性能最优。
 - `performance`：使用 cuDNN 自带的试运行搜索算法，会根据卷积形状和类型试运行所有卷积算法，然后选择最优算法。该参数保证性能最优。
 - `algo_0`：该算法将卷积表示为矩阵乘积的和，而没有实际显式地形成保存输入张量数据的矩阵。求和使用原子加法操作完成，因此结果是不确定的。
 - `algo_1`：该算法将卷积表示为矩阵乘积，而没有实际显式地形成保存输入张量数据的矩阵。结果是确定的。
 - `fft`：该算法利用快速傅里叶变换完成卷积计算。需要额外申请较大内存空间，保存中间结果。结果是确定的。
 - `fft_tiling`：该算法利用快速傅里叶变换完成卷积计算，但是需要对输入进行分块。同样需要额外申请内存空间，保存中间结果，但是对大尺寸的输入，所需内存空间小于 `fft` 算法。结果是确定的。
 - `winograd`：该算法利用 Winograd 变换完成卷积计算。需要额外申请合理的内存空间，保存中间结果。结果是确定的。
 - `winograd_nonfused`：该算法利用 Winograd 变形算法完成卷积计算。需要额外申请较大的内存空间，保存中间结果。结果是确定的。

样例：

```
>>> import mindspore as ms
>>> ms.device_context.gpu.op_tuning.conv_dgrad_algo("performance")
```

10.2.6 mindspore.device_context.gpu.op_tuning.conv_fprop_algo

`mindspore.device_context.gpu.op_tuning.conv_fprop_algo(mode)`

指定 cuDNN 的卷积前向算法。详细信息请查看 [NVIDIA cuDNN 关于 cudnnConvolutionForward 的说明](#)。

参数：

- **mode** (str) - cuDNN 的卷积前向算法。如未配置，框架默认为 `normal`。其值范围如下：
 - `normal`：使用 cuDNN 自带的启发式搜索算法，会根据卷积形状和类型快速选择合适的卷积算法。该参数不保证性能最优。
 - `performance`：使用 cuDNN 自带的试运行搜索算法，会根据卷积形状和类型试运行所有卷积算法，然后选择最优算法。该参数保证性能最优。
 - `implicit_gemm`：该算法将卷积隐式转换成矩阵乘法，完成计算。不需要显式将输入张量数据转换成矩阵形式保存。

- **precomp_gemm**: 该算法将卷积隐式转换成矩阵乘法, 完成计算。但是需要一些额外的内存空间去保存预计算得到的索引值, 以便隐式地将输入张量数据转换成矩阵形式。
- **gemm**: 该算法将卷积显式转换成矩阵乘法, 完成计算。在显式完成矩阵乘法过程中, 需要额外申请内存空间, 将输入转换成矩阵形式。
- **direct**: 该算法直接完成卷积计算, 不会隐式或显式地将卷积转换成矩阵乘法。
- **fft**: 该算法利用快速傅里叶变换完成卷积计算。需要额外申请内存空间, 保存中间结果。
- **fft_tiling**: 该算法利用快速傅里叶变换完成卷积计算, 但是需要对输入进行分块。同样需要额外申请内存空间, 保存中间结果, 但是对大尺寸的输入, 所需内存空间小于 **fft** 算法。
- **winograd**: 该算法利用 Winograd 变换完成卷积计算。需要额外申请合理的内存空间, 保存中间结果。
- **winograd_nonfused**: 该算法利用 Winograd 变形算法完成卷积计算。需要额外申请较大的内存空间, 保存中间结果。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.gpu.op_tuning.conv_fprop_algo("performance")
```

10.2.7 mindspore.device_context.gpu.op_tuning.conv_wgrad_algo

`mindspore.device_context.gpu.op_tuning.conv_wgrad_algo(mode)`

指定 cuDNN 的卷积输入卷积核的反向算法。详细信息请查看 [NVIDIA cuDNN 关于 cudnnConvolutionBackwardFilter](#) 的说明。

参数:

- **mode (str)** - cuDNN 的卷积输入卷积核的反向算法。如未配置, 框架默认为 **normal**。其值范围如下:
 - **normal**: 使用 cuDNN 自带的启发式搜索算法, 会根据卷积形状和类型快速选择合适的卷积算法。该参数不保证性能最优。
 - **performance**: 使用 cuDNN 自带的试运行搜索算法, 会根据卷积形状和类型试运行所有卷积算法, 然后选择最优算法。该参数保证性能最优。
 - **algo_0**: 该算法将卷积表示为矩阵乘积的和, 而没有实际显式地形成保存输入张量数据的矩阵。求和使用原子加法操作完成, 因此结果是不确定的。
 - **algo_1**: 该算法将卷积表示为矩阵乘积, 而没有实际显式地形成保存输入张量数据的矩阵。结果是确定的。
 - **algo_3**: 该算法类似于 **algo_0**, 但使用一些小的工作空间来预计算一些索引。结果也是不确定的。
 - **fft**: 该算法利用快速傅里叶变换完成卷积计算。需要额外申请较大的内存空间, 保存中间结果。结果是确定的。
 - **fft_tiling**: 该算法利用快速傅里叶变换完成卷积计算, 但是需要对输入进行分块。同样需要额外申请内存空间, 保存中间结果, 但是对大尺寸的输入, 所需内存空间小于 **fft** 算法。结果是确定的。
 - **winograd_nonfused**: 该算法利用 Winograd 变形算法完成卷积计算。需要额外申请较大的内存空间, 保存中间结果。结果是确定的。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.gpu.op_tuning.conv_wgrad_algo("performance")
```

10.3 Ascend 设备后端管理

<code>mindspore.device_context.ascend.device_count</code>	查询可用的 Ascend 类型的设备数量。
<code>mindspore.device_context.ascend.is_available</code>	查询 Ascend 设备是否可用。
<code>mindspore.device_context.ascend.op_precision.conv_allow_hf32</code>	是否为 Conv 类算子使能 FP32 转换为 HF32。
<code>mindspore.device_context.ascend.op_precision.matmul_allow_hf32</code>	是否为 Matmul 类算子使能 FP32 转换为 HF32。
<code>mindspore.device_context.ascend.op_precision.precision_mode</code>	配置混合精度模式。
<code>mindspore.device_context.ascend.op_precision.op_precision_mode</code>	设置算子精度模式配置文件的路径。
<code>mindspore.device_context.ascend.op_debug.execute_timeout</code>	设置一个算子的最大执行时间，以秒为单位。
<code>mindspore.device_context.ascend.op_debug.debug_option</code>	使能 Ascend 算子调试选项，默认不开启。
<code>mindspore.device_context.ascend.op_debug.aclinit_config</code>	配置 aclInit 接口的配置项。
<code>mindspore.device_context.ascend.op_tuning.op_compile</code>	是否选择在线编译，框架默认设置为静态 shape 选择在线编译，动态 shape 选择算子二进制文件，该默认设置将来可能会发生变化。
<code>mindspore.device_context.ascend.op_tuning.aoe_tune_mode</code>	表示启动 AOE 调优，默认不设置。
<code>mindspore.device_context.ascend.op_tuning.aoe_job_type</code>	设置 AOE 工具专用的参数，需要和 <code>mindspore.device_context.op_tuning.aoe_tune_mode(tune_mode)</code> 配合使用，框架默认设置为 2。

10.3.1 mindspore.device_context.ascend.device_count

`mindspore.device_context.ascend.device_count()`
查询可用的 Ascend 类型的设备数量。

返回：

int，可用的 Ascend 类型的设备数量。

样例：

```
>>> import mindspore as ms
>>> print(ms.device_context.ascend.device_count())
8
```

10.3.2 mindspore.device_context.ascend.is_available

`mindspore.device_context.ascend.is_available()`

查询 Ascend 设备是否可用。

返回:

bool, Ascend 设备在当前 MindSpore 包中是否可用。

样例:

```
>>> import mindspore as ms
>>> print(ms.device_context.ascend.is_available())
True
```

10.3.3 mindspore.device_context.ascend.op_precision.conv_allow_hf32

`mindspore.device_context.ascend.op_precision.conv_allow_hf32(value)`

是否为 Conv 类算子使能 FP32 转换为 HF32。CANN 默认使能 Conv 类算子 FP32 转换为 HF32。如果您想了解更多详细信息, 请查询 [昇腾社区](#)。

说明:

- 这是一个实验特性, 可能会被更改或者删除。

参数:

- **value** (bool) - 是否为 Conv 类算子使能 FP32 转换为 HF32。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_precision.conv_allow_hf32(True)
```

10.3.4 mindspore.device_context.ascend.op_precision.matmul_allow_hf32

`mindspore.device_context.ascend.op_precision.matmul_allow_hf32(value)`

是否为 Matmul 类算子使能 FP32 转换为 HF32。CANN 默认不使能 Matmul 类算子 FP32 转换为 HF32。如果您想了解更多详细信息, 请查询 [昇腾社区](#)。

说明:

- 这是一个实验特性, 可能会被更改或者删除。

参数:

- **value** (bool) - 是否为 Matmul 类算子使能 FP32 转换为 HF32。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_precision.matmul_allow_hf32(True)
```

10.3.5 mindspore.device_context.ascend.op_precision.precision_mode

`mindspore.device_context.ascend.op_precision.precision_mode(mode)`

配置混合精度模式。框架默认设置 Atlas 训练系列产品的默认配置为 “allow_fp32_to_fp16”，Atlas A2 训练系列产品等其他产品的默认配置为 “must_keep_origin_dtype”。如果您想了解更多详细信息，请查询 [昇腾社区](#)。

说明：

- 该参数的默认值属于实验性质参数，将来可能会发生变化。

参数：

- **mode** (str) - 混合精度模式设置。

其值范围如下：

- **force_fp16**: 当算子既支持 float16，又支持 float32 时，直接选择 float16。
- **allow_fp32_to_fp16**: 对于矩阵类算子，使用 float16。对于矢量类算子，优先保持原图精度：如果网络模型中算子支持 float32，则保留原始精度 float32；如果网络模型中算子不支持 float32，则直接降低精度到 float16。
- **allow_mix_precision**: 自动混合精度，针对全网算子，按照内置的优化策略，自动将部分算子的精度降低到 float16 或 bfloat16。
- **must_keep_origin_dtype**: 保持原图精度。
- **force_fp32**: 当矩阵计算的算子输入为 float16，输出既支持 float16 又支持 float32 时，强制转换成 float32 输出。
- **allow_fp32_to_bf16**: 对于矩阵类算子，使用 bfloat16。对于矢量类算子，优先保持原图精度：如果网络模型中算子支持 float32，则保留原始精度 float32；如果网络模型中算子不支持 float32，则直接降低精度到 bfloat16。
- **allow_mix_precision_fp16**: 自动混合精度，针对全网算子，按照内置的优化策略，自动将部分算子的精度降低到 float16。
- **allow_mix_precision_bf16**: 自动混合精度，针对全网算子，按照内置的优化策略，自动将部分算子的精度降低到 bfloat16。

样例：

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_precision.precision_mode("force_fp16")
```

10.3.6 mindspore.device_context.ascend.op_precision.op_precision_mode

`mindspore.device_context.ascend.op_precision.op_precision_mode(path)`

设置算子精度模式配置文件的路径。如果您想了解更多详细信息，请查询 [昇腾社区](#)。

参数：

- **path** (str) - 算子精度模式配置文件（.ini 格式）的所在路径，用于设置算子精度模式。该目录可以包含字母、数字、下划线（_）、连字符（-）和点（.）。

样例：


```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_precision.op_precision_mode("./op_precision_config_file")
```

10.3.7 mindspore.device_context.ascend.op_debug.execute_timeout

`mindspore.device_context.ascend.op_debug.execute_timeout(op_timeout)`

设置一个算子的最大执行时间，以秒为单位。框架默认算子执行超时时间为 900 秒。详细信息请查看 [昇腾社区关于 aclrtSetOpExecuteTimeOut 文档说明](#)。

参数：

- **op_timeout** (int) - 算子的最大执行时间。如果执行时间超过这个值，系统将终止该任务。0 意味着使用默认值。AI Core 和 AI CPU 算子在不同硬件上的默认值有差异。

样例：

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_debug.execute_timeout(100)
```

10.3.8 mindspore.device_context.ascend.op_debug.debug_option

`mindspore.device_context.ascend.op_debug.debug_option(option_value)`

使能 Ascend 算子调试选项，默认不开启。

参数：

- **option_value** (str) - Ascend 算子调试配置，当前只支持内存访问越界检测，可配置为 oom。
 - oom: 涉及从全局内存中读写数据，例如读写算子数据等，该选项开启全局内存访问越界检测，实际执行算子时，若出现内存越界，AscendCL 会返回 EZ9999 错误码。

样例：

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_debug.debug_option("oom")
```

10.3.9 mindspore.device_context.ascend.op_debug.aclinit_config

`mindspore.device_context.ascend.op_debug.aclinit_config(config)`

配置 aclInit 接口的配置项。详细信息请查看 [昇腾社区关于 aclInit 的文档说明](#)。

参数：

- **config** (dict) - 初始化 AscendCL 时，可通过该配置接口开启或设置以下功能。
 - **max_opqueue_num** : 通过单算子模型方式执行时，为节约内存和平衡调用性能，可通过 **max_opqueue_num** 参数配置单算子模型映射队列的最大长度。如果长度达到最大，会先删除长期未使用的映射信息和缓存中的单算子模型，再加载最新的映射信息和对应的单算子模型。如果不配置映射队列的最大长度，则默认最大长度为 20000。
 - **err_msg_mode** : 用于控制按进程或线程级别获取错误信息，默认按进程级别。“0”表示按线程级别获取错误信息；“1”为默认值，表示按进程级别获取错误信息。

- dump: 控制开启 Ascend 算子异常 Dump。可设置为 {"dump_scene": "lite_exception"}、{"dump_scene": "lite_exception:disable"}。{"dump_scene": "lite_exception"} 表示开启异常 Dump。{"dump_scene": "lite_exception:disable"} 表示关闭异常 Dump。{"dump_scene": "lite_exception"} 为默认值, 表示开启异常 Dump。

样例:

```
>>> import mindspore as ms
>>> ms.set_device("Ascend", 0)
>>> ms.device_context.ascend.op_debug.aclinit_config({"max_opqueue_num": "20000", "err_msg_
↪mode": "1",
...
↪"dump": {"dump_scene": "lite_exception"
↪"}})
```

10.3.10 mindspore.device_context.ascend.op_tuning.op_compile

`mindspore.device_context.ascend.op_tuning.op_compile(value)`

是否选择在线编译, 框架默认设置为静态 shape 选择在线编译, 动态 shape 选择算子二进制文件, 该默认设置将来可能会发生变化。如果您想了解更多详细信息, 请查询 [昇腾社区](#) 了解。

参数:

- **value** (bool) - 表示是否选择在线编译。
 - True: 优先选择在线编译。
 - False: 优先选择系统中已经编译好的算子二进制文件, 提升编译性能。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_tuning.op_compile(True)
```

10.3.11 mindspore.device_context.ascend.op_tuning.aoe_tune_mode

`mindspore.device_context.ascend.op_tuning.aoe_tune_mode(tune_mode)`

表示启动 AOE 调优, 默认不设置。

参数:

- **tune_mode** (str) - AOE 调优模式配置。
 - "online": 启动在线调优。
 - "offline": 为离线调优保存 GE 图。

样例:

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_tuning.aoe_tune_mode("online")
```

10.3.12 mindspore.device_context.ascend.op_tuning.aoe_job_type

`mindspore.device_context.ascend.op_tuning.aoe_job_type` (*config*)

设置 AOE 工具专用的参数，需要和 `mindspore.device_context.op_tuning.aoe_tune_mode(tune_mode)` 配合使用，框架默认设置为 2。

参数：

- **config** (str) - 选择调优类型。
 - "1": 设置为子图调优。
 - "2": 设置为算子调优。

样例：

```
>>> import mindspore as ms
>>> ms.device_context.ascend.op_tuning.aoe_job_type("1")
```


mindspore.communication

集合通信接口。

注意，集合通信接口需要先配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

<code>mindspore.communication.GlobalComm</code>	<code>GlobalComm</code> 是一个储存通信信息的全局类。
<code>mindspore.communication.init</code>	初始化通信服务需要的分布式后端，例如 <i>HCCL</i> 、 <i>NCCL</i> 或 <i>MCCL</i> 服务。
<code>mindspore.communication.release</code>	释放分布式资源，例如 <i>HCCL</i> 或 <i>NCCL</i> 或 <i>MCCL</i> 服务。
<code>mindspore.communication.create_group</code>	创建用户自定义的通信组实例。
<code>mindspore.communication.destroy_group</code>	注销用户通信组。
<code>mindspore.communication.get_comm_name</code>	获取指定通讯组的通讯器名称。
<code>mindspore.communication.get_group_size</code>	获取指定通信组实例的 <code>rank_size</code> 。
<code>mindspore.communication.get_group_rank_from_world_rank</code>	由通信集群中的全局设备序号获取指定用户通信组中的 <code>rank ID</code> 。
<code>mindspore.communication.get_local_rank</code>	获取指定通信组中当前设备的本地设备序号。
<code>mindspore.communication.get_local_rank_size</code>	获取指定通信组的本地设备总数。
<code>mindspore.communication.get_process_group_ranks</code>	获取指定通信组中的进程，并将通信组中的进程编号以列表方式返回。
<code>mindspore.communication.get_rank</code>	在指定通信组中获取当前的设备序号。
<code>mindspore.communication.get_world_rank_from_group_rank</code>	由指定通信组中的设备序号获取通信集群中的全局设备序号。
<code>mindspore.communication.HCCL_WORLD_COMM_GROUP</code>	" <code>hccl_world_group</code> " 字符串，指的是由 <i>HCCL</i> 创建的默认通信组。
<code>mindspore.communication.NCCL_WORLD_COMM_GROUP</code>	" <code>nccl_world_group</code> " 字符串，指的是由 <i>NCCL</i> 创建的默认通信组。
<code>mindspore.communication.MCCL_WORLD_COMM_GROUP</code>	" <code>mccl_world_group</code> " 字符串，指的是由 <i>MCCL</i> 创建的默认通信组。

11.1 mindspore.communication.GlobalComm

class mindspore.communication.GlobalComm

GlobalComm 是一个储存通信信息的全局类。成员包含：BACKEND、WORLD_COMM_GROUP。

- BACKEND：使用的通信库，"hccl"、"nccl" 或者 "mccl"。"hccl" 代表华为集合通信库 HCCL，"nccl" 代表英伟达集合通信库 NCCL，"mccl" 代表 MindSpore 集合通信库 MCCL。
- WORLD_COMM_GROUP：全局通信域，"hccl_world_group"、"nccl_world_group" 或者 "mccl_world_group"。

11.2 mindspore.communication.init

mindspore.communication.init(*backend_name=None*)

初始化通信服务需要的分布式后端，例如 HCCL、NCCL 或 MCCL 服务。通常在分布式并行场景下使用，并在使用通信服务前设置。

说明：

- HCCL 的全称是华为集合通信库（Huawei Collective Communication Library）。
- NCCL 的全称是英伟达集合通信库（NVIDIA Collective Communication Library）。
- MCCL 的全称是 MindSpore 集合通信库（MindSpore Collective Communication Library）。
- 在 Ascend 硬件平台下，init() 接口的设置需要在创建 Tensor 和 Parameter 之前，以及所有算子和网络的实例化和运行之前。

参数：

- **backend_name** (str) - 分布式后端的名称，可选 "hccl"、"nccl" 或者 "mccl"。在 Ascend 硬件平台下，应使用 "hccl"，在 GPU 硬件平台下，应使用 "nccl"，在 CPU 硬件平台下，应使用 "mccl"。如果未设置则根据硬件平台类型（device_target）自动进行推断，默认值：None。

异常：

- **TypeError** - 参数 *backend_name* 不是字符串。
- **RuntimeError** - 1) 硬件设备类型无效；2) 后端服务无效；3) 分布式计算初始化失败；4) 后端是 HCCL 的情况下，未设置环境变量 *RANK_ID* 或 *MINDSPORE_HCCL_CONFIG_PATH* 的情况下初始化 HCCL 服务。

支持平台：

Ascend GPU CPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 msrun 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> from mindspore.communication import init
>>> init()
```

11.3 mindspore.communication.release

`mindspore.communication.release()`

释放分布式资源，例如 *HCCL* 或 *NCCL* 或 *MCCL* 服务。

说明：

- *release* 方法应该在 *init* 方法之后使用。如果不使用，则在程序结束时资源才会自动释放。

异常：

- **RuntimeError** - 在释放分布式资源失败时抛出。

支持平台：

Ascend GPU CPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> from mindspore.communication import init, release
>>> init()
>>> release()
```

11.4 mindspore.communication.create_group

`mindspore.communication.create_group(group, rank_ids)`

创建用户自定义的通信组实例。

说明：

- MindSpore 的 GPU 和 CPU 版本不支持此方法。
- 列表 `rank_ids` 的长度应大于 1。
- 列表 `rank_ids` 内不能有重复数据。
- *create_group* 方法应该在 *init* 方法之后使用。
- 如果没有使用 `mpirun` 启动，PyNative 模式下仅支持全局单通信组。

参数：

- **group** (str) - 输入用户自定义的通信组实例名称，支持数据类型为 `str`。
- **rank_ids** (list) - 设备编号列表。

异常：

- **TypeError** - 参数 `group` 不是字符串或参数 `rank_ids` 不是列表。

- **ValueError** - 列表 `rank_ids` 的长度小于 1，或列表 `rank_ids` 内有重复数据，以及后端无效。
- **RuntimeError** - *HCCL* 服务不可用时，或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台：

Ascend GPU CPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore import set_context
>>> from mindspore import ops
>>> from mindspore.communication import init, create_group, get_rank
>>> set_context(mode=ms.GRAPH_MODE)
>>> ms.set_device(device_target="Ascend")
>>> init()
>>> group = "0-7"
>>> rank_ids = [0, 7]
>>> if get_rank() in rank_ids:
...     create_group(group, rank_ids)
...     allreduce = ops.AllReduce(group)
```

11.5 mindspore.communication.destroy_group

`mindspore.communication.destroy_group(group)`

注销用户通信组。

说明：

- MindSpore 的 GPU 和 CPU 版本不支持此方法。
- 参数 `group` 不能是 `"hccl_world_group"`。
- `destroy_group` 方法应该在 `init` 方法之后使用。

参数：

- **group** (str) - 被注销通信组实例（通常由 `create_group` 方法创建）的名称。

异常：

- **TypeError** - 在参数 `group` 不是字符串时抛出。
- **ValueError** - 在参数 `group` 是 `"hccl_world_group"` 或后端不可用时抛出。
- **RuntimeError** - *HCCL* 服务不可用时，或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台：

Ascend GPU CPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore import set_context
>>> from mindspore import ops
>>> from mindspore.communication import init, create_group, destroy_group, get_rank
>>> set_context(mode=ms.GRAPH_MODE)
>>> ms.set_device(device_target="Ascend")
>>> init()
>>> group = "0-2"
>>> rank_ids = [0, 2]
>>> if get_rank() in rank_ids:
...     create_group(group, rank_ids)
...     destroy_group(group)
```

11.6 mindspore.communication.get_comm_name

`mindspore.communication.get_comm_name` (*group=GlobalComm.WORLD_COMM_GROUP*)

获取指定通讯组的通讯器名称。

说明:

- MindSpore 的 GPU 和 CPU 版本不支持此方法。
- `get_comm_name` 方法应该在 `init` 方法之后使用。

参数:

- **group** (str) - 传入的通信组名称, 通常由 `create_group` 方法创建, 或默认使用 `GlobalComm.WORLD_COMM_GROUP`。

返回:

str, 指定通讯组的通讯器名称。

异常:

- **TypeError** - 在参数 `group` 不是字符串时抛出。
- **ValueError** - 在后端不可用时抛出。
- **RuntimeError** - `HCCL` 服务不可用时, 或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.communication import init, create_group, get_rank, get_comm_name
>>> ms.set_device(device_target="Ascend")
>>> init()
>>> world_group_comm_name = get_comm_name()
>>> group = "0-7"
>>> rank_ids = [0,7]
>>> if get_rank() in rank_ids:
...     create_group(group, rank_ids)
...     customizd_group_comm_name = get_comm_name(group)
...     print("comm_name of customizd group is ", customizd_group_comm_name)
>>> print("comm_name of world group is: ", world_group_comm_name)
comm_name of customizd group is: 11.22.33.44%eth0_60000_0_0123456789101112
comm_name of world group is: 11.22.33.44%eth0_60000_0_1211109876543210
```

11.7 mindspore.communication.get_group_size

`mindspore.communication.get_group_size(group=GlobalComm.WORLD_COMM_GROUP)`

获取指定通信组实例的 `rank_size`。

说明:

- `get_group_size` 方法应该在 `init` 方法之后使用。
-

参数:

- **group** (str) - 指定工作组实例(由 `create_group` 方法创建)的名称, 支持数据类型为 `str`, 默认值为 `GlobalComm.WORLD_COMM_GROUP`。

返回:

指定通信组实例的 `rank_size`, 数据类型为 `int`。

异常:

- **TypeError** - 在参数 `group` 不是字符串时抛出。
- **ValueError** - 在后端不可用时抛出。
- **RuntimeError** - 在 `HCCL` 或 `NCCL` 或 `MCCL` 服务不可用时抛出。

支持平台:

Ascend GPU CPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.communication import init, get_group_size
>>> init()
>>> group_size = get_group_size()
>>> print("group_size is: ", group_size)
group_size is: 8
```

11.8 mindspore.communication.get_group_rank_from_world_rank

`mindspore.communication.get_group_rank_from_world_rank(world_rank_id, group)`

由通信集群中的全局设备序号获取指定用户通信组中的 rank ID。

说明:

- MindSpore 的 GPU 和 CPU 版本不支持此方法。
- 参数 `group` 不能是 "hccl_world_group"。
- `get_group_rank_from_world_rank` 方法应该在 `init` 方法之后使用。

参数:

- **world_rank_id** (*int*) - 通信集群内的全局 rank ID。
- **group** (*str*) - 指定通信组实例（由 `create_group` 方法创建）的名称。

返回:

当前通信组内的 rank_ID，数据类型为 int。

异常:

- **TypeError** - 在参数 `world_rank_id` 不是数字或参数 `group` 不是字符串时抛出。
- **ValueError** - 在参数 `group` 是 "hccl_world_group" 或后端不可用时抛出。
- **RuntimeError** - HCCL 服务不可用时，或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台:

Ascend GPU CPU

样例:

说明: 运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore import set_context
>>> from mindspore.communication import init, create_group, get_group_rank_from_world_rank, \
->get_rank
>>> set_context(mode=ms.GRAPH_MODE)
>>> ms.set_device(device_target="Ascend")
>>> init()
```

(下页继续)

(续上页)

```
>>> group = "0-4"
>>> rank_ids = [0,4]
>>> if get_rank() in rank_ids:
...     create_group(group, rank_ids)
...     group_rank_id = get_group_rank_from_world_rank(4, group)
...     print("group_rank_id is: ", group_rank_id)
group_rank_id is: 1
```

11.9 mindspore.communication.get_local_rank

`mindspore.communication.get_local_rank` (*group=GlobalComm.WORLD_COMM_GROUP*)

获取指定通信组中当前设备的本地设备序号。

说明:

- MindSpore 的 GPU 和 CPU 版本不支持此方法。
- `get_local_rank` 方法应该在 `init` 方法之后使用。

参数:

- **group** (*str*) - 通信组名称, 通常由 `create_group` 方法创建, 否则将使用默认组名称。默认值: `GlobalComm.WORLD_COMM_GROUP`。

返回:

`int`, 调用该方法的进程对应的通信组内本地设备序号。

异常:

- **TypeError** - 在参数 `group` 不是字符串时抛出。
- **ValueError** - 在后端不可用时抛出。
- **RuntimeError** - `HCCL` 服务不可用时, 或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台:

Ascend GPU CPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.communication import init, get_rank, get_local_rank
>>> ms.set_device(device_target="Ascend")
>>> init()
>>> world_rank = get_rank()
>>> local_rank = get_local_rank()
>>> print("local_rank is: {}, world_rank is {}".format(local_rank, world_rank))
local_rank is: 1, world_rank is 9
```

11.10 mindspore.communication.get_local_rank_size

`mindspore.communication.get_local_rank_size` (*group=GlobalComm.WORLD_COMM_GROUP*)
获取指定通信组的本地设备总数。

说明:

- MindSpore 的 GPU 和 CPU 版本不支持此方法。
- `get_local_rank_size` 方法应该在 `init` 方法之后使用。

参数:

- **group** (str) - 传入的通信组名称, 通常由 `create_group` 方法创建, 或默认使用 `GlobalComm.WORLD_COMM_GROUP`。

返回:

int, 调用该方法的进程对应的通信组设备总数。

异常:

- **TypeError** - 在参数 `group` 不是字符串时抛出。
- **ValueError** - 在后端不可用时抛出。
- **RuntimeError** - `HCCL` 服务不可用时, 或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台:

Ascend GPU CPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore.communication import init, get_local_rank_size
>>> ms.set_device(device_target="Ascend")
>>> init()
>>> local_rank_size = get_local_rank_size()
>>> print("local_rank_size is: ", local_rank_size)
local_rank_size is: 8
```

11.11 mindspore.communication.get_process_group_ranks

`mindspore.communication.get_process_group_ranks` (*group*=`GlobalComm.WORLD_COMM_GROUP`)

获取指定通信组中的进程，并将通信组中的进程编号以列表方式返回。

参数：

- **group** (str, 可选) - 工作的通信组。默认值：`GlobalComm.WORLD_COMM_GROUP`（即 Ascend 平台为 `"hccl_world_group"`，GPU 平台为 `"nccl_world_group"`）。

返回：

List (List[int]) - 指定通信组中的进程编号列表。

异常：

- **TypeError** - *group* 不是字符串。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 4 卡环境下运行。

```
>>> import numpy as np
>>> from mindspore.communication import init, get_process_group_ranks
>>>
>>> init()
>>> output = get_process_group_ranks()
>>> print(output)
[0, 1, 2, 3]
```

11.12 mindspore.communication.get_rank

`mindspore.communication.get_rank` (*group*=`GlobalComm.WORLD_COMM_GROUP`)

在指定通信组中获取当前的设备序号。

说明：

- `get_rank` 方法应该在 `init` 方法之后使用。
-

参数：

- **group** (str) - 通信组名称，通常由 `create_group` 方法创建，否则将使用默认组。默认值：`GlobalComm.WORLD_COMM_GROUP`。

返回：

int, 调用该方法的进程对应的组内序号。

异常:

- **TypeError** - 在参数 *group* 不是字符串时抛出。
- **ValueError** - 在后端不可用时抛出。
- **RuntimeError** - 在 *HCCL* 或 *NCCL* 或 *MCCL* 服务不可用时抛出。

支持平台:

Ascend GPU CPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> from mindspore.communication import init, get_rank
>>> init()
>>> rank_id = get_rank()
>>> print(rank_id)
>>> # the result is the rank_id in world_group
```

11.13 mindspore.communication.get_world_rank_from_group_rank

`mindspore.communication.get_world_rank_from_group_rank(group, group_rank_id)`

由指定通信组中的设备序号获取通信集群中的全局设备序号。

说明:

- MindSpore 的 GPU 和 CPU 版本不支持此方法。
- 参数 *group* 不能是 "hccl_world_group"。
- `get_world_rank_from_group_rank` 方法应该在 `init` 方法之后使用。

参数:

- **group** (str) - 传入的通信组名称, 通常由 `create_group` 方法创建。
- **group_rank_id** (int) - 通信组内的设备序号。

返回:

int, 通信集群中的全局设备序号。

异常:

- **TypeError** - 参数 *group* 不是字符串或参数 *group_rank_id* 不是数字。
- **ValueError** - 参数 *group* 是 "hccl_world_group" 或后端不可用。
- **RuntimeError** - *HCCL* 服务不可用时, 或者使用了 MindSpore 的 GPU 或 CPU 版本。

支持平台:

Ascend GPU CPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

```
>>> import mindspore as ms
>>> from mindspore import set_context
>>> from mindspore.communication import init, create_group, get_world_rank_from_group_rank, \
    get_rank
>>> set_context(mode=ms.GRAPH_MODE)
>>> ms.set_device(device_target="Ascend")
>>> init()
>>> group = "0-4"
>>> rank_ids = [0,4]
>>> if get_rank() in rank_ids:
...     create_group(group, rank_ids)
...     world_rank_id = get_world_rank_from_group_rank(group, 1)
...     print("world_rank_id is: ", world_rank_id)
world_rank_id is: 4
```

11.14 mindspore.communication.HCCL_WORLD_COMM_GROUP

`mindspore.communication.HCCL_WORLD_COMM_GROUP`

"hccl_world_group" 字符串, 指的是由 HCCL 创建的默认通信组。在 Ascend 硬件平台下, 初始化通信服务后该字符串与 `GlobalComm.WORLD_COMM_GROUP` 等价, 推荐使用 `GlobalComm.WORLD_COMM_GROUP` 获取当前全局通信组。

11.15 mindspore.communication.NCCL_WORLD_COMM_GROUP

`mindspore.communication.NCCL_WORLD_COMM_GROUP`

"nccl_world_group" 字符串, 指的是由 NCCL 创建的默认通信组。在 GPU 硬件平台下, 初始化通信服务后该字符串与 `GlobalComm.WORLD_COMM_GROUP` 等价, 推荐使用 `GlobalComm.WORLD_COMM_GROUP` 获取当前全局通信组。

11.16 mindspore.communication.MCCL_WORLD_COMM_GROUP

`mindspore.communication.MCCL_WORLD_COMM_GROUP`

"mccl_world_group" 字符串, 指的是由 MCCL 创建的默认通信组。在 CPU 硬件平台下, 初始化通信服务后该字符串与 `GlobalComm.WORLD_COMM_GROUP` 等价, 推荐使用 `GlobalComm.WORLD_COMM_GROUP` 获取当前全局通信组。

11.17 mindspore.communication.comm_func

集合通信函数式接口。

<code>mindspore.communication.comm_func.all_gather_into_tensor</code>	汇聚指定的通信组中的 Tensor ，并返回汇聚后的张量。
<code>mindspore.communication.comm_func.all_reduce</code>	使用指定方式对通信组内的所有设备的 Tensor 数据进行规约操作，所有设备都得到相同的结果，返回规约操作后的 Tensor 。
<code>mindspore.communication.comm_func.all_to_all_single_with_output_shape</code>	根据用户输入的切分大小，把输入 <i>tensor</i> 切分后，发送到其他的设备上，并从其他设备接收切分块，然后合并到一个输出 Tensor 中。
<code>mindspore.communication.comm_func.all_to_all_with_output_shape</code>	根据用户输入的张量列表，将对应的张量发送到远端设备，并从其他设备接收张量，返回一个接收的张量列表。
<code>mindspore.communication.comm_func.barrier</code>	同步通信域内的多个进程。
<code>mindspore.communication.comm_func.batch_isend_irecv</code>	异步地发送和接收张量。
<code>mindspore.communication.comm_func.broadcast</code>	对输入数据整组广播。
<code>mindspore.communication.comm_func.gather_into_tensor</code>	对通信组的输入 Tensor 进行聚合。
<code>mindspore.communication.comm_func.irecv</code>	异步接收 Tensor 到指定线程。
<code>mindspore.communication.comm_func.isend</code>	异步发送 Tensor 到指定线程。
<code>mindspore.communication.comm_func.recv</code>	同步接收 Tensor 到指定线程。
<code>mindspore.communication.comm_func.send</code>	同步发送 Tensor 到指定线程。
<code>mindspore.communication.comm_func.P2POp</code>	用于存放关于'isend'、'irecv'相关的信息，并用于 <i>batch_isend_irecv</i> 接口的入参。
<code>mindspore.communication.comm_func.reduce</code>	规约指定通信组中的 Tensor ，并将规约结果发送到目标为 <i>dst</i> 的进程 (全局的进程编号) 中，返回发送到目标进程的 Tensor 。
<code>mindspore.communication.comm_func.reduce_scatter_tensor</code>	规约并且分发指定通信组中的 Tensor ，返回分发后的 Tensor 。
<code>mindspore.communication.comm_func.scatter_tensor</code>	将输入的 Tensor 均匀散射到通信域的卡上。

11.17.1 mindspore.communication.comm_func.all_gather_into_tensor

`mindspore.communication.comm_func.all_gather_into_tensor` (*tensor*,
group=GlobalComm.WORLD_COMM_GROUP,
async_op=False)

汇聚指定的通信组中的 **Tensor**，并返回汇聚后的张量。

说明:

- 集合中所有进程的 **Tensor** 必须具有相同的 *shape* 和格式。

参数:

- tensor** (**Tensor**) - 输入待汇聚操作的 **Tensor**，**Tensor** 的 *shape* 为 $(x_1, x_2, \dots, x_R)$ 。

- **group** (str, 可选) - 工作的通信组, 默认值: `GlobalComm.WORLD_COMM_GROUP` (即 Ascend 平台为 "hccl_world_group", GPU 平台为 "nccl_world_group")。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值: `False`。

返回:

`Tuple(Tensor, CommHandle)`, 如果组中的 device 数量为 N , 则输出 Tensor 的 shape 为 $(N, x_1, x_2, \dots, x_R)$ 。若 `async_op` 是 `True`, `CommHandle` 是一个异步工作句柄。若 `async_op` 是 `False`, `CommHandle` 将返回 `None`。

异常:

- **TypeError** - 首个输入的数据类型不为 `Tensor`, `group` 不是 `str`。
- **ValueError** - 调用进程的 rank id 大于本通信组的 rank 大小。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> comm.init()
>>> input_tensor = ms.Tensor(np.ones([2, 8]).astype(np.float32))
>>> output, _ = comm.comm_func.all_gather_into_tensor(input_tensor)
>>> print(output)
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

11.17.2 mindspore.communication.comm_func.all_reduce

`mindspore.communication.comm_func.all_reduce` (*tensor*, *op=ReduceOp.SUM*,
group=GlobalComm.WORLD_COMM_GROUP,
async_op=False)

使用指定方式对通信组内的所有设备的 Tensor 数据进行规约操作, 所有设备都得到相同的结果, 返回规约操作后的 Tensor。

说明: 集合中的所有进程的 Tensor 必须具有相同的 shape 和格式。

参数:

- **tensor** (Tensor) - 输入待规约操作的 Tensor, Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。

- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max" 和 "min"。默认值: ReduceOp.SUM。
- **group** (str, 可选) - 工作的通信组。默认值: GlobalComm.WORLD_COMM_GROUP (即 Ascend 平台为 "hccl_world_group", GPU 平台为 "nccl_world_group")。
- **async_op** (bool, 可选) - 本算子是否为异步算子。默认值: False。

返回:

Tuple(Tensor, CommHandle), 输出 shape 与输入相同, 即 $(x_1, x_2, \dots, x_R)$ 。其内容取决于操作。若 *async_op* 是 True, CommHandle 是一个异步工作句柄。若 *async_op* 是 False, CommHandle 将返回 None。

异常:

- **TypeError** - 首个输入的数据类型不为 Tensor, *op* 或 *group* 不是 str。
- **RuntimeError** - 目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 msrun 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> comm.init()
>>> input_tensor = ms.Tensor(np.ones([2, 8]).astype(np.float32))
>>> output, _ = comm.comm_func.all_reduce(input_tensor)
>>> print(output)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]]
```

11.17.3 mindspore.communication.comm_func.all_to_all_single_with_output_shape

`mindspore.communication.comm_func.all_to_all_single_with_output_shape` (*output_shape*, *tensor*,
output_split_sizes=None,
input_split_sizes=None,
group=None,
async_op=False)

根据用户输入的切分大小, 把输入 *tensor* 切分后, 发送到其他的设备上, 并从其他设备接收切分块, 然后合并到一个输出 Tensor 中。

说明: 各个 rank 之间发送和接收的切分块大小需要互相匹配。仅支持 PyNative 模式, 目前不支持 Graph 模式。

参数:

- **output_shape** (Union(Tensor, Tuple(int))) - 表示接收 Tensor 的形状。

- **tensor** (Tensor) - 要发送到远端设备的 Tensor。
- **output_split_sizes** (Union(Tuple(int), List(int))) - 接收 Tensor 在 0 维的切分大小列表。默认值: None, 表示均匀切分。
- **input_split_sizes** (Union(Tuple(int), List(int))) - 发送 Tensor 在 0 维的切分大小列表。默认值: None, 表示均匀切分。
- **group** (str, 可选) - 通信执行所在的通信组。默认值: None。为 None 时, 在 Ascend 上将使用为 `hccl_world_group`, 在 GPU 上使用 `nccl_world_group`。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值: False。

返回:

Tuple(Tensor, CommHandle), 其中 Tensor 为从远端设备接收分块并合并的结果。如果从其他设备接收的 Tensor 为空, 则返回空张量, 且张量数值无意义。若 `async_op` 是 True, CommHandle 是一个异步工作句柄。若 `async_op` 是 False, CommHandle 将返回 None。

异常:

- **TypeError** - `tensor` 不是 Tensor 类型。
- **TypeError** - `output_shape` 不是元组或者 Tensor 类型。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> comm.init()
>>> rank = comm.get_rank()
>>> input = ms.Tensor([0, 1]) + rank * 2
>>> output_shape = (2,)
>>> result, _ = comm.comm_func.all_to_all_single_with_output_shape(output_shape, input)
>>> print(result)
rank 0:
[ 0.  2.]
rank 1:
[ 1.  3.]
```

11.17.4 mindspore.communication.comm_func.all_to_all_with_output_shape

`mindspore.communication.comm_func.all_to_all_with_output_shape` (*output_shape_list*, *input_tensor_list*, *group=None*, *async_op=False*)

根据用户输入的张量列表，将对应的张量发送到远端设备，并从其他设备接收张量，返回一个接收的张量列表。

说明：各个设备之间发送和接收的张量形状需要互相匹配。仅支持 PyNative 模式，目前不支持 Graph 模式。

参数：

- **output_shape_list** (Union[Tuple(Tensor), List(Tensor), Tuple(Tuple(int))]) - 包含接收张量形状 of 列表。
- **input_tensor_list** (Union[Tuple(Tensor), List(Tensor)]) - 包含发送到其他设备张量的列表。
- **group** (str, 可选) - 通信所使用的通信组。默认值:None。为 None 时，在 Ascend 上将使用 `hccl_world_group`，在 GPU 使用 `nccl_world_group`。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值: False。

返回：

`Tuple(Tuple(Tensor), CommHandle), Tuple(Tensor)` 为从远端设备接收的张量列表。若 `async_op` 是 True，`CommHandle` 是一个异步工作句柄。若 `async_op` 是 False，`CommHandle` 将返回 None。

异常：

- **TypeError** - *input_tensor_list* 中不全是张量类型。
- **TypeError** - *output_shape_list* 中不全是张量或者元组类型。
- **TypeError** - *input_tensor_list* 中张量的数据类型不全部一致。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> comm.init()
>>> this_rank = comm.get_rank()
>>> if this_rank == 0:
...     send_tensor_list = [ms.Tensor(1.), ms.Tensor([[2, 3], [4, 5.]])]
...     recv_tensor_list = [(), (2,)]
>>> if this_rank == 1:
...     send_tensor_list = [ms.Tensor([2, 2.]), ms.Tensor([4, 5, 6, 7.])]
...     recv_tensor_list = [(2, 2), (4,)]
>>> output, _ = comm.comm_func.all_to_all_with_output_shape(recv_tensor_list, send_tensor_
↪list)
```

(下页继续)

(续上页)

```
>>> print(output)
rank 0:
(Tensor(shape=[], dtype=Float32, value= 1),
Tensor(shape=[2], dtype=Float32, value= [2.00000000e+00, 2.00000000e+00]))
rank 1:
(Tensor(shape=[2, 2], dtype=Float32, value=
[[2.00000000e+00, 3.00000000e+00],
[4.00000000e+00, 5.00000000e+00]]),
Tensor(shape=[4], dtype=Float32, value=[4.00000000e+00, 5.00000000e+00, 6.00000000e+00, 7.
→00000000e+00]))
```

11.17.5 mindspore.communication.comm_func.barrier

`mindspore.communication.comm_func.barrier` (*group=GlobalComm.WORLD_COMM_GROUP*)

同步通信域内的多个进程。进程调用到该算子后进入阻塞状态，直到通信域内所有进程调用到该算子，进程被唤醒并继续执行。

参数：

- **group** (str, 可选) - 表示通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

异常：

- **RuntimeError** - 后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在多卡环境下运行。

```
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> # Launch 2 processes.
>>> comm.init()
>>> comm.comm_func.barrier()
>>> print("barrier finish!")
barrier finish!
```

11.17.6 mindspore.communication.comm_func.batch_isend_irecv

`mindspore.communication.comm_func.batch_isend_irecv(p2p_op_list)`
异步地发送和接收张量。

说明:

- 不同设备中, `p2p_op_list` 中的 `P2POp` 的 "isend" 和 "irecv" 应该互相匹配。
- `p2p_op_list` 中的 `P2POp` 应该使用同一个通信组。
- 暂不支持 `p2p_op_list` 中的 `P2POp` 含有 `tag` 入参。
- `p2p_op_list` 中的 `P2POp` 的 `tensor` 的值不会被最后的结果原地修改。
- 仅支持 PyNative 模式, 目前不支持 Graph 模式。

参数:

- **p2p_op_list** (`P2POp`) - 包含 `P2POp` 类型对象的列表。`P2POp` 指的是 `mindspore.communication.comm_func.P2POp`。

返回:

`Tuple(Tensor)`。根据 `p2p_op_list` 中的 `P2POp` 的发送/接收顺序, 得到的接收张量元组。当 `P2POp` 为发送时, 相应位置的结果是没有意义的张量。当 `P2POp` 为接收时, 相应位置的结果是从其他设备接收到的张量。

异常:

- **TypeError** - `p2p_op_list` 中不全是 `P2POp` 类型。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> comm.init()
>>> this_rank = comm.get_rank()
>>> world_size = comm.get_group_size()
>>> next_rank = (this_rank + 1) % world_size
>>> prev_rank = (this_rank + world_size - 1) % world_size
>>>
>>> send_tensor = ms.Tensor(this_rank + 1, dtype=ms.float32)
>>> recv_tensor = ms.Tensor(0., dtype=ms.float32)
>>>
>>> send_op = comm.comm_func.P2POp('isend', send_tensor, next_rank)
>>> recv_op = comm.comm_func.P2POp('irecv', recv_tensor, prev_rank)
```

(下页继续)

(续上页)

```
>>>
>>> p2p_op_list = [send_op, recv_op]
>>> output = comm.comm_func.batch_isend_irecv(p2p_op_list)
>>> print(output)
rank 0:
(Tensor(shape=[], dtype=Float32, value= 0), Tensor(shape=[], dtype=Float32, value= 2))
rank 1:
(Tensor(shape=[], dtype=Float32, value= 0), Tensor(shape=[], dtype=Float32, value= 1))
```

11.17.7 mindspore.communication.comm_func.broadcast

`mindspore.communication.comm_func.broadcast` (*tensor*, *src=0*, *group=GlobalComm.WORLD_COMM_GROUP*)
对输入数据整组广播。

说明：集合中的所有进程的 Tensor 的 shape 和数据格式必须相同。当前支持 PyNative 模式，不支持 Graph 模式。

参数：

- **tensor** (Tensor) - 输入待广播的 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **src** (int, 可选) - 表示发送源的进程编号。只有该进程会广播张量。默认值：0。
- **group** (str, 可选) - 表示通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

返回：

Tensor, Tensor 的 shape 与输入相同，即 $(x_1, x_2, \dots, x_R)$ 。

异常：

- **TypeError** - src 不是 int 或 group 不是 str。
- **RuntimeError** - 目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在多卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> # Launch 2 processes.
>>>
>>> comm.init()
>>> data = ms.Tensor(np.arange(8).reshape([2, 4]).astype(np.float32))
>>> out = comm.comm_func.broadcast(tensor=data, src=0)
```

(下页继续)

(续上页)

```
>>> print(out)
[[0. 1. 2. 3.]
 [4. 5. 6. 7.]]
```

11.17.8 mindspore.communication.comm_func.gather_into_tensor

`mindspore.communication.comm_func.gather_into_tensor` (*tensor*, *dst=0*,
group=GlobalComm.WORLD_COMM_GROUP)

对通信组的输入 Tensor 进行聚合。操作会将每张卡的输入 Tensor 的第 0 维度上进行聚合，发送到对应卡上。

说明：只有目标为 *dst* 的进程 (全局的进程编号) 才会收到聚合操作后的输出。其他进程只得到一个形状为 [1] 的 Tensor，且该 Tensor 没有数学意义。当前支持 PyNative 模式，不支持 Graph 模式。

参数：

- **tensor** (Tensor) - 输入待聚合的 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **dst** (int, 可选) - 表示发送源的进程编号。只有该进程会接收聚合后的 Tensor。默认值：0。
- **group** (str, 可选) - 表示通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

返回：

Tensor，即 $(\sum x_1, x_2, \dots, x_R)$ 。Tensor 第 0 维等于输入数据第 0 维求和，其他 shape 相同。

异常：

- **TypeError** - 首个输入的数据类型不为 Tensor，*op* 或 *group* 不是 str。
- **RuntimeError** - 目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在多卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> # Launch 2 processes.
>>>
>>> comm.init()
>>> input = ms.Tensor(np.arange(4).reshape([2, 2]).astype(np.float32))
>>> output = comm.comm_func.gather_into_tensor(tensor=input, dst=0)
>>> print(output)
Process with rank 0: [[0. 1.]
```

(下页继续)

(续上页)

```
[2. 3.],
[0. 1.],
[2. 3.]]
Process with rank 1: [0]
```

11.17.9 mindspore.communication.comm_func.irecv

`mindspore.communication.comm_func.irecv (tensor, src=0, group=GlobalComm.WORLD_COMM_GROUP, tag=0)`
异步接收 Tensor 到指定线程。

说明: Send 和 Receive 算子需组合使用, 且有同一个 *tag*。输入的 *tensor* 的 shape 和 dtype 将用于接收 Tensor, 但 *tensor* 的数据值不起作用。当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **tensor** (Tensor) - 输入 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。输入的 *tensor* 的 shape 和 dtype 将用于接收 Tensor, 但 *tensor* 的数据值不起作用。
- **src** (int, 可选) - 表示发送源的进程编号。只会接收来自源进程的 Tensor。默认值: 0。
- **group** (str, 可选) - 工作的通信组。默认值: Ascend 平台为 "hccl_world_group", GPU 平台为 "nccl_world_group"。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将被接收来自相同 *tag* 的 Send 发送的 Tensor。默认值: 0。

返回:

Tuple(Tensor, CommHandle), 输出 Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。CommHandle 是一个异步工作句柄。

异常:

- **TypeError** - src 不是 int 或 group 不是 str。
- **ValueError** - 该线程的 rank id 大于通信组的 rank size。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore.communication.comm_func import isend, irecv
>>> from mindspore.communication import get_rank, get_group_size
>>>
>>> np.random.seed(1)
```

(下页继续)

(续上页)

```

>>> init()
>>> rank = get_rank()
>>> size = get_group_size()
>>> x = np.ones([2, 2]).astype(np.float32) * 0.01 * (rank + 1)
>>> x2 = np.ones([2, 2]).astype(np.float32)
>>>
>>>
>>> if rank < size / 2:
...     _x = ms.Tensor(x)
...     isend(_x, rank + size // 2)
... else:
...     _x2 = ms.Tensor(x2)
...     output, handle = irecv(_x2, rank - size // 2)
...     handle.wait()
...     print(output)
rank1:
[[0.01  0.01]
 [0.01  0.01]]

```

11.17.10 mindspore.communication.comm_func.isend

`mindspore.communication.comm_func.isend(tensor, dst=0, group=GlobalComm.WORLD_COMM_GROUP, tag=0)`
异步发送 Tensor 到指定线程。

说明: Send 和 Receive 算子需组合使用, 且有同一个 *tag*。当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **tensor** (Tensor) - 输入 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **dst** (int, 可选) - 表示发送目标的进程编号。只有目标进程会收到 Tensor。默认值: 0。
- **group** (str, 可选) - 工作的通信组。默认值: Ascend 平台为 "hccl_world_group", GPU 平台为 "nccl_world_group"。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将被拥有相同 *tag* 的 Receive 接收。默认值: 0。

返回:

CommHandle 是一个异步工作句柄。

异常:

- **TypeError** - dst 不是 int 或 group 不是 str。
- **ValueError** - 该线程的 rank id 大于通信组的 rank size。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore.communication.comm_func import isend, irecv
>>> from mindspore.communication import get_rank, get_group_size
>>>
>>> np.random.seed(1)
>>> init()
>>> rank = get_rank()
>>> size = get_group_size()
>>> x = np.ones([2, 2]).astype(np.float32) * 0.01 * (rank + 1)
>>> x2 = np.ones([2, 2]).astype(np.float32)
>>>
>>>
>>> if rank < size / 2:
...     _x = ms.Tensor(x)
...     isend(_x, rank + size // 2)
... else:
...     _x2 = ms.Tensor(x2)
...     output, handle = irecv(_x2, rank - size // 2)
...     handle.wait()
...     print(output)
rank1:
[[0.01  0.01]
 [0.01  0.01]]
```

11.17.11 mindspore.communication.comm_func.recv

`mindspore.communication.comm_func.recv` (*tensor*, *src*=0, *group*=*GlobalComm.WORLD_COMM_GROUP*, *tag*=0)

同步接收 Tensor 到指定线程。

说明: Send 和 Receive 算子需组合使用, 且有同一个 *tag*。输入的 *tensor* 的 *shape* 和 *dtype* 将用于接收 Tensor, 但 *tensor* 的数据值不起作用。当前支持 PyNative 模式, 不支持 Graph 模式。

参数:

- **tensor** (Tensor) - 输入 Tensor。Tensor 的 *shape* 为 $(x_1, x_2, \dots, x_R)$ 。输入的 *tensor* 的 *shape* 和 *dtype* 将用于接收 Tensor, 但 *tensor* 的数据值不起作用。
- **src** (int, 可选) - 表示发送源的进程编号。只会接收来自源进程的 Tensor。默认值: 0。
- **group** (str, 可选) - 工作的通信组。默认值: Ascend 平台为 "hccl_world_group", GPU 平台为 "nccl_world_group"。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将被接收来自相同 *tag* 的 Send 发送的 Tensor。默认值: 0。

返回:

Tensor, 其 *shape* 为 $(x_1, x_2, \dots, x_R)$ 。

异常:

- **TypeError** - *src* 不是 int 或 *group* 不是 str。

- **ValueError** - 如果该线程的 rank id 大于通信组的 rank size。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore.communication.comm_func import send, recv
>>> from mindspore.communication import get_rank, get_group_size
>>>
>>> np.random.seed(1)
>>> init()
>>> rank = get_rank()
>>> size = get_group_size()
>>> x = np.ones([2, 2]).astype(np.float32) * 0.01 * (rank + 1)
>>> x2 = np.ones([2, 2]).astype(np.float32)
>>>
>>>
>>> if rank < size // 2:
...     _x = ms.Tensor(x)
...     send(_x, rank + size // 2)
... else:
...     _x2 = ms.Tensor(x2)
...     output = recv(_x2, rank - size // 2)
...     print(output)
rank1:
[[0.01  0.01]
 [0.01  0.01]]
```

11.17.12 mindspore.communication.comm_func.send

`mindspore.communication.comm_func.send(tensor, dst=0, group=GlobalComm.WORLD_COMM_GROUP, tag=0)`
同步发送 Tensor 到指定线程。

说明：Send 和 Receive 算子需组合使用，且有同一个 *tag*。当前支持 PyNative 模式，不支持 Graph 模式。

参数：

- **tensor** (Tensor) - 输入 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **dst** (int, 可选) - 表示发送目标的进程编号。只有目标进程会收到 Tensor。默认值：0。
- **group** (str, 可选) - 工作的通信组。默认值：Ascend 平台为 "hccl_world_group"，GPU 平台为 "nccl_world_group"。
- **tag** (int, 可选) - 用于区分发送、接收消息的标签。该消息将被拥有相同 *tag* 的 Receive 接收。默认值：0。

异常：

- **TypeError** - dst 不是 int，或 group 不是 str。
- **ValueError** - 该线程的 rank id 大于通信组的 rank size。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore.communication.comm_func import send, recv
>>> from mindspore.communication import get_rank, get_group_size
>>>
>>> np.random.seed(1)
>>> init()
>>> rank = get_rank()
>>> size = get_group_size()
>>> x = np.ones([2, 2]).astype(np.float32) * 0.01 * (rank + 1)
>>> x2 = np.ones([2, 2]).astype(np.float32)
>>>
>>>
>>> if rank < size / 2:
...     _x = ms.Tensor(x)
...     send(_x, rank + size // 2)
... else:
...     _x2 = ms.Tensor(x2)
...     output = recv(_x2, rank - size // 2)
...     print(output)
rank1:
[[0.01  0.01]
 [0.01  0.01]]
```

11.17.13 mindspore.communication.comm_func.P2POp

class mindspore.communication.comm_func.**P2POp**(*op, tensor, peer, group=None, tag=0, *, recv_dtype=None*)
用于存放关于‘isend’、‘irecv’相关的信息，并用于 *batch_isend_irecv* 接口的入参。

说明：

- 当 *op* 入参为‘irecv’时，*tensor* 入参允许不传入张量类型，可以只传入接收张量的形状。
- *tensor* 入参不会被最后的结果原地修改。

参数：

- **op** (Union[str, function]) - 对于字符串类型, 只允许'isend'和'irecv'。对于函数类型, 只允许 `comm_func.isend` 和 `comm_func.irecv` 函数。
- **tensor** (Union[Tensor, Tuple(int)]) - 用于发送或接收的张量。如果是 *op* 是'irecv', 可以传入接收张量的形状。
- **peer** (int) - 发送或接收的远程设备的全局编号。
- **group** (str, 可选) - 工作的通信组, 默认值: `GlobalComm.WORLD_COMM_GROUP` (即 Ascend 平台为 "hccl_world_group", GPU 平台为 "nccl_world_group")。
- **tag** (int, 可选) - 当前暂不支持。默认值: 0。

关键字参数:

- **recv_dtype** (mindspore.dtype, 可选) - 表示接收张量的数据类型。当 *tensor* 传入的是张量的形状时, 该入参必须要配置。默认值: None。

返回:

P2POp 对象。

异常:

- **ValueError** - 当 *op* 不是与'isend'和'irecv'相关的字符串或函数。
- **TypeError** - 当 *tensor* 不是张量或者元组类型。
- **NotImplementedError** - 当 *tag* 入参不为 0。

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> send_tensor = ms.Tensor(1.)
>>> send_op = comm.comm_func.P2POp('isend', send_tensor, 1)
>>> send_op = comm.comm_func.P2POp(comm.comm_func.isend, send_tensor, 1)
>>> recv_tensor = ms.Tensor(0.)
>>> recv_op = comm.comm_func.P2POp('irecv', recv_tensor, 0)
>>> recv_op = comm.comm_func.P2POp(comm.comm_func.irecv, recv_tensor, 0)
>>> recv_op = comm.comm_func.P2POp('irecv', (), 0, recv_dtype=ms.float32)
```

11.17.14 mindspore.communication.comm_func.reduce

`mindspore.communication.comm_func.reduce` (*tensor*, *dst*, *op=ReduceOp.SUM*,
group=GlobalComm.WORLD_COMM_GROUP)

规约指定通信组中的 Tensor，并将规约结果发送到目标为 *dst* 的进程 (全局的进程编号) 中，返回发送到目标进程的 Tensor。

说明：只有目标为 *dst* 的进程 (全局的进程编号) 才会收到规约操作后的输出。当前支持 PyNative 模式，不支持 Graph 模式。其他进程只得到一个形状为 [1] 的 Tensor，且该 Tensor 没有数学意义。

参数：

- **tensor** (Tensor) - 输入待规约的 Tensor，Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **dst** (int) - 指定接收输出的目标进程编号，只有该进程会接收规约操作后的输出结果。
- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max" 和 "min"。默认值：ReduceOp.SUM。
- **group** (str, 可选) - 工作的通信组，默认值：GlobalComm.WORLD_COMM_GROUP（即 Ascend 平台为 "hccl_world_group"，GPU 平台为 "nccl_world_group"）。

返回：

Tensor，数据类型与输入的 *tensor* 一致，shape 为 $(x_1, x_2, \dots, x_R)$ 。

异常：

- **TypeError** - 首个输入的数据类型不为 Tensor，*op* 和 *group* 不是字符串。
- **RuntimeError** - 目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 4 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> # Launch 4 processes.
>>> comm.init()
>>> dest_rank=1
>>> input_tensor = ms.Tensor(np.ones([2, 8]).astype(np.float32))
>>> output = comm.comm_func.reduce(input_tensor, dst=dest_rank)
>>> print(output)
Process with rank 1: [[4. 4. 4. 4. 4. 4. 4. 4.]
[4. 4. 4. 4. 4. 4. 4. 4.]],
Other proesses: [0.]
```


11.17.15 mindspore.communication.comm_func.reduce_scatter_tensor

`mindspore.communication.comm_func.reduce_scatter_tensor` (*tensor*, *op*=`ReduceOp.SUM`,
group=`GlobalComm.WORLD_COMM_GROUP`,
async_op=`False`)

规约并且分发指定通信组中的 Tensor，返回分发后的 Tensor。

说明：在集合的所有过程中，Tensor 必须具有相同的 shape 和格式。

参数：

- **tensor** (Tensor) - 输入待规约且分发的 Tensor，假设其形状为 $(N, *)$ ，其中 $*$ 为任意数量的额外维度。N 必须能够被 `rank_size` 整除，`rank_size` 为当前通讯组里面的计算卡数量。
- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max" 和 "min"。默认值：`ReduceOp.SUM`。
- **group** (str, 可选) - 工作的通信组，默认值：`GlobalComm.WORLD_COMM_GROUP`（即 Ascend 平台为 "hccl_world_group"，GPU 平台为 "nccl_world_group"）。
- **async_op** (bool, 可选) - 本算子是否是异步算子。默认值：`False`。

返回：

Tuple(Tensor, CommHandle)，输出 Tensor 数据类型与 *input_x* 一致，shape 为 $(N/rank\_size, *)$ 。若 *async_op* 是 True，CommHandle 是一个异步工作句柄。若 *async_op* 是 False，CommHandle 将返回 None。

异常：

- **TypeError** - 首个输入的数据类型不为 Tensor，*op* 和 *group* 不是字符串。
- **ValueError** - 输入的第一个维度不能被 rank size 整除。
- **RuntimeError** - 目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> comm.init()
>>> input_tensor = ms.Tensor(np.ones([8, 8]).astype(np.float32))
>>> output, _ = comm.comm_func.reduce_scatter_tensor(input_tensor)
>>> print(output)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]]
```

11.17.16 mindspore.communication.comm_func.scatter_tensor

`mindspore.communication.comm_func.scatter_tensor` (*tensor*, *src*=0,
group=`GlobalComm.WORLD_COMM_GROUP`)

将输入的 **Tensor** 均匀散射到通信域的卡上。

说明：该接口只支持 **Tensor** 输入，且只支持均匀切分。只有源为 *src* 的进程 (全局的进程编号) 才会将输入 **Tensor** 作为散射源。当前支持 **PyNative** 模式，不支持 **Graph** 模式。

参数：

- **tensor** (**Tensor**) - 输入待散射的 **Tensor**。**Tensor** 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **src** (int, 可选) - 表示发送源的进程编号。只有该进程会发送散射源 **Tensor**。默认值：0。
- **group** (str, 可选) - 表示通信域。默认值：`GlobalComm.WORLD_COMM_GROUP`。

返回：

Tensor，即 $(x_1/src\_rank, x_2, \dots, x_R)$ 。**Tensor** 第 0 维等于输入数据第 0 维除以 *src*，其他维度相同。

异常：

- **TypeError** - 首个输入的数据类型不为 **Tensor**，*op* 或 *group* 不是 str。
- **RuntimeError** - 目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在多卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.communication as comm
>>>
>>> # Launch 2 processes.
>>>
>>> comm.init()
>>> input = ms.Tensor(np.arange(8).reshape([4, 2]).astype(np.float32))
>>> out = comm.comm_func.scatter_tensor(tensor=input, src=0)
>>> print(out)
# rank_0
[[0. 1.]
 [2. 3.]]
# rank_1
[[4. 5.]
 [6. 7.]]
```

MindSpore 的核心数据加载模块是 Dataset，是一种基于 Pipeline 设计的 [数据引擎](#)。

该模块提供了以下几种数据加载方式，帮助用户加载数据集到 MindSpore 中。

- 自定义数据集加载：允许用户通过 Python 定义 [可随机访问 \(Map-style\) 数据集](#) 和 [可迭代 \(Iterable-style\) 数据集](#) 自定义数据读取、处理逻辑。
- 标准格式数据集加载：支持加载业界标准数据格式的数据集文件，包括 [MindRecord](#)、[TFRecord](#) 等。
- 开源数据集加载：支持部分 [开源数据集](#) 的解析读取，如 MNIST、CIFAR-10、CLUE、LJSpeech 等。

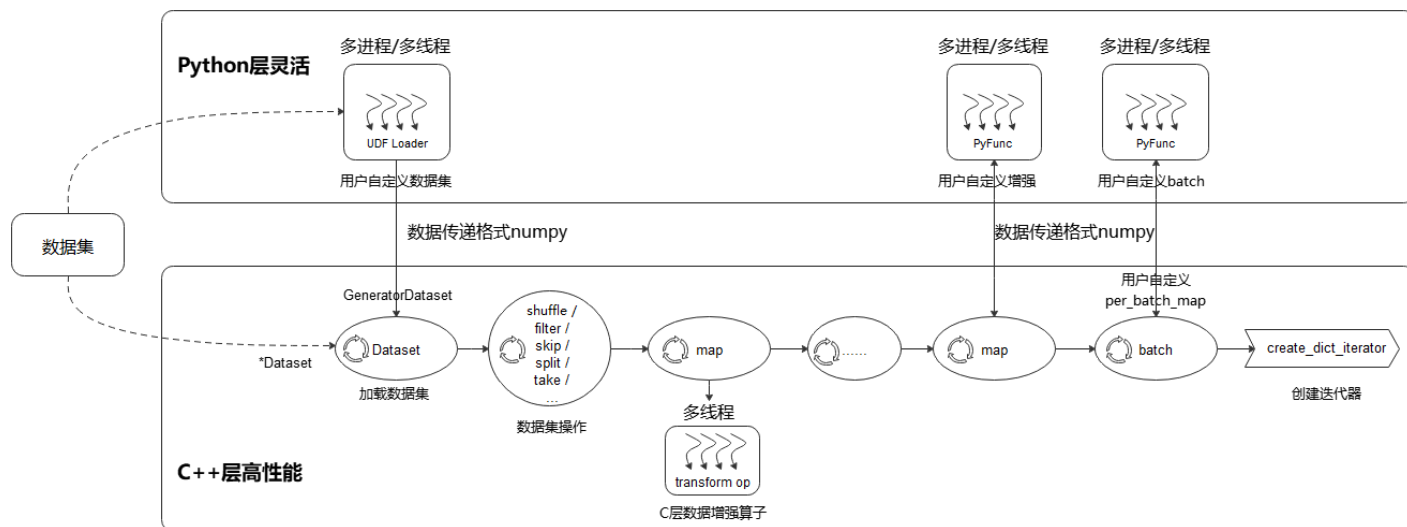
此外，该模块也提供了对数据进行采样、增强变换、批处理等功能，以及随机种子、并行数等基础配置，与数据集加载 API 配合使用。

- 数据采样器：提供了多种常见 [采样器](#)，如 [RandomSampler](#)、[DistributedSampler](#) 等。
- 数据增强变换：提供了多种 [数据集操作](#)，以此对数据进行增强，批处理等。
- 基础配置：提供了 [Pipeline 配置](#) 用于随机种子设置、并行数设置、数据恢复模式等功能。

常用数据集术语说明如下：

- Dataset，所有数据集的基类，提供了数据处理方法来帮助预处理数据。
- SourceDataset，一个抽象类，表示数据集管道的来源，从文件和数据库等数据源生成数据。
- MappableDataset，一个抽象类，表示支持随机访问的源数据集。
- Iterator，用于枚举元素的数据集迭代器的基类。

12.1 数据处理 Pipeline 介绍



MindSpore Dataset Pipeline 示例

如上图所示，MindSpore Dataset 模块使得用户很简便地定义数据预处理 Pipeline，并以最高效（多进程/多线程）的方式处理数据集中样本，具体的步骤参考如下：

- 加载数据集（Dataset）：用户可以方便地使用 Dataset 类（标准格式数据集、vision 数据集、nlp 数据集、audio 数据集）来加载已支持的数据集，或者使用自定义数据集加载，通过 Python 逻辑自定义数据集行为；
- 数据集操作（filter/skip）：用户通过数据集对象方法 .shuffle / .filter / .skip / .split / .take / ... 来实现数据集的进一步混洗、过滤、跳过、最多获取条数等操作；
- 数据集样本变换操作（map）：用户可以将数据变换操作（vision 数据变换，nlp 数据变换，audio 数据变换）添加到 .map 操作中执行，数据预处理过程中可以定义多个 map 操作，用于执行不同变换操作，数据变换操作也可以支持传入用户自定义 Python 函数；
- 批（batch）：用户在样本完成变换后，使用 .batch 操作将多个样本组织成 batch，也可以通过 batch 的参数 per_batch_map 来自定义 batch 逻辑；
- 迭代器（iterator）：最后用户通过数据集对象方法 .create_dict_iterator / .create_tuple_iterator 来创建迭代器将预处理完成的数据循环输出。

12.2 数据处理 Pipeline 快速上手

如何快速使用 Dataset Pipeline，可以将使用数据 Pipeline 加载 & 处理数据集下载到本地，按照顺序执行并观察输出结果。

12.3 自定义数据集加载

`mindspore.dataset.GeneratorDataset`

自定义 Python 数据源，通过迭代该数据源构造数据集。

12.3.1 mindspore.dataset.GeneratorDataset

```
class mindspore.dataset.GeneratorDataset (source, column_names=None, column_types=None, schema=None,
                                         num_samples=None, num_parallel_workers=1, shuffle=None,
                                         sampler=None, num_shards=None, shard_id=None,
                                         python_multiprocessing=True, max_rowsize=None,
                                         batch_sampler=None, collate_fn=None)
```

自定义 Python 数据源，通过迭代该数据源构造数据集。生成的数据集的列名和列类型取决于用户定义的 Python 数据源。

参数：

- **source** (Union[Random Accessible, Iterable]) - 自定义数据集，表示从此数据对象加载数据。MindSpore 支持两种类型的数据集。
 - 可随机访问 (map-style) 数据集：它是一种实现了 `__getitem__()` 和 `__len__()` 方法的数据集对象，记录从索引/键到数据样本的映射。例如，使用 `source[idx]` 访问数据集时，可以从磁盘上的文件夹中读取第 `idx` 个样本，详情请参阅 [可随机访问数据集样例](#)。
 - 可迭代 (iterable-style) 数据集：它是一种实现了 `__iter__()` 和 `__next__()` 方法的数据集对象，表示数据样本的可迭代性。这种类型的数据集适用于随机读取成本较高甚至不可能的情况，以及适用于批量大小取决于获取数据的情况。例如，使用 `iter(source)` 访问数据集时，可以返回从数据库、远程服务器读取的数据流，详情请参阅 [可迭代数据集样例](#)。
- **column_names** (Union[str, list[str]], 可选) - 指定数据集生成的列名。默认值：None，不指定。用户可以通过此参数或 `schema` 参数指定列名。
- **column_types** (list[mindspore.dtype], 可选) - 指定生成数据集各个数据列的数据类型。默认值：None，不指定。如果未指定该参数，则自动推断类型；如果指定了该参数，将在数据输出时做类型匹配检查（后续版本将废弃此参数）。
- **schema** (Union[str, Schema], 可选) - 数据格式策略，用于指定读取数据列的数据类型、数据维度等信息。支持传入 JSON 文件路径或 `mindspore.dataset.Schema` 构造的对象（后续版本将废弃此参数）。默认值：None。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作进程数/线程数（由参数 `python_multiprocessing` 决定当前为多进程模式或多线程模式）。默认值：1。
- **shuffle** (bool, 可选) - 是否混洗数据集。只有输入的 `source` 参数带有可随机访问属性（`__getitem__`）时，才可以指定该参数。默认值：None。下表中会展示不同配置的预期行为。
- **sampler** (Union[Sampler, Iterable], 可选) - 指定从数据集中选取样本的采样器。只有输入的 `source` 参数带有可随机访问属性（`__getitem__`）时，才可以指定该参数。默认值：None。下表中会展示不同配置的预期行为。

- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **python_multiprocessing** (bool, 可选) - 启用 Python 多进程模式加速运算。默认值: True。当传入 `source` 的 Python 对象的计算量很大时, 开启此选项可能会有较好效果。
- **max_rowsize** (int, 可选) - 指定在多线程之间复制数据时, 共享内存分配的基本单位, 单位为 MB, 总占用的共享内存会随着 `num_parallel_workers` 和 `mindspore.dataset.config.set_prefetch_size()` 增加而变大。如果设置为 -1, 共享内存将随数据大小动态分配。仅当参数 `python_multiprocessing` 设为 True 时, 此参数才会生效。默认值: None, 动态分配共享内存 (后续版本将废弃此参数)。
- **batch_sampler** (Iterable, 可选) - 与 `sampler` 类似, 但每次返回 1 批索引, 对应的数据将被合并为 1 个 Batch。不可与 `num_samples`, `shuffle`, `num_shards`, `shard_id` 和 `sampler` 等参数同时使用。默认值: None, 不使用批采样器。
- **collate_fn** (Callable[List[numpy.ndarray]], 可选) - 定义如何将数据列表合并为 1 个 Batch。仅在使用了 `batch_sampler` 时有效。默认值: None, 不指定合并函数。

异常:

- **RuntimeError** - Python 对象 `source` 在执行期间引发异常。
- **RuntimeError** - `column_names` 参数指定的列名数量与 `source` 参数输出的数据数量不匹配。
- **ValueError** - `num_parallel_workers` 参数超过最大线程数。
- **ValueError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **ValueError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **ValueError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **ValueError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **TypeError** - 如果 `batch_sampler` 不为可迭代类型。
- **ValueError** - 如果 `batch_sampler` 与 `num_samples`, `shuffle`, `num_shards`, `shard_id` 和 `sampler` 同时指定。
- **TypeError** - 如果 `collate_fn` 不为可调用函数。
- **ValueError** - 如果在指定 `collate_fn` 时没有指定 `batch_sampler`。

样例:

```
>>> import mindspore.dataset as ds
>>> import numpy as np
>>>
>>> # 1) Multidimensional generator function as callable input.
>>> def generator_multidimensional():
...     for i in range(64):
...         yield (np.array([[i, i + 1], [i + 2, i + 3]]),)
>>>
>>> dataset = ds.GeneratorDataset(source=generator_multidimensional, column_names=["multi_
->dimensional_data"])
>>>
>>> # 2) Multi-column generator function as callable input.
>>> def generator_multi_column():
...     for i in range(64):
...         yield np.array([i]), np.array([[i, i + 1], [i + 2, i + 3]])
```

(下页继续)

(续上页)

```

>>>
>>> dataset = ds.GeneratorDataset(source=generator_multi_column, column_names=["col1", "col2",
↳"])
>>>
>>> # 3) Iterable dataset as iterable input.
>>> class MyIterable:
...     def __init__(self):
...         self._index = 0
...         self._data = np.random.sample((5, 2))
...         self._label = np.random.sample((5, 1))
...
...     def __next__(self):
...         if self._index >= len(self._data):
...             raise StopIteration
...         else:
...             item = (self._data[self._index], self._label[self._index])
...             self._index += 1
...             return item
...
...     def __iter__(self):
...         self._index = 0
...         return self
...
...     def __len__(self):
...         return len(self._data)
>>>
>>> dataset = ds.GeneratorDataset(source=MyIterable(), column_names=["data", "label"])
>>>
>>> # 4) Random accessible dataset as random accessible input.
>>> class MyAccessible:
...     def __init__(self):
...         self._data = np.random.sample((5, 2))
...         self._label = np.random.sample((5, 1))
...
...     def __getitem__(self, index):
...         return self._data[index], self._label[index]
...
...     def __len__(self):
...         return len(self._data)
>>>
>>> dataset = ds.GeneratorDataset(source=MyAccessible(), column_names=["data", "label"])
>>>
>>> # list, dict, tuple of Python is also random accessible
>>> dataset = ds.GeneratorDataset(source=[(np.array(0),), (np.array(1),), (np.array(2),)],
↳column_names=["col"])

```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：

- 如果配置 `python_multiprocessing=True`（默认值：True）和 `num_parallel_workers>1`（默认值：1）表示启动了多进程方式进行数据 load 加速，此时随着数据集迭代，子进程的内存占用会逐渐增加，主要是因为自定义数据集的子进程以 Copy-On-Write 的方式获取主进程中的成员变量。举例：如果自定义数据集 `__init__` 函数中包含大量成员变量数据（例如：在数据集构建时加载了一个非常大的文件名列表）并且使用了多进程方式，那这可能会

导致产生 OOM 的问题 (总内存的预估使用量是: (子进程数量 + 1) * 父进程的内存大小)。最简单的解决方法是成员变量用非引用数据类型 (如: Pandas、Numpy 或 PyArrow 对象) 替换 Python 对象 (如: list / dict / int / float / string 等), 或者加载更少的元数据以减小成员变量, 或者配置 `python_multiprocessing=False` 使用多线程方式。

你可以使用以下类/函数来减少成员变量的大小:

- `mindspore.dataset.utils.LineReader`: 在 `__init__` 函数中, 使用该类初始化你的文本文件对象, 然后在 `__getitem__` 函数中通过该对象按行号读取文件内容。
- `source` 参数接收用户自定义的 Python 函数 (PyFuncs), 通过 `ds.config.set_multiprocessing_start_method("spawn")` 方式设置多进程的启动方式为 `spawn` 模式, 且 `python_multiprocessing=True` 和 `num_parallel_workers>1` 时, 支持将 `mindspore.nn` 和 `mindspore.ops` 目录下或其他的网络计算算子添加到 `source` 中, 否则不支持添加到 `source` 中。
- 当 `source` 自定义数据集对象在数据集加载及处理时, 调用了 DVPP 算子, 那么支持的场景如下:

	多线程	多进程	
		spawn	fork
独立进程	数据处理: 支持 数据处理 + 网络训练: 不支持	数据处理: 支持 数据处理 + 网络训练: 支持	数据处理: 支持 数据处理 + 网络训练: 不支持
非独立进程	数据处理: 支持 数据处理 + 网络训练: 支持	数据处理: 支持 数据处理 + 网络训练: 支持	数据处理: 支持 数据处理 + 网络训练: 不支持

- 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 2: 参数 `sampler` 和 `num_samples`, `shuffle`, `num_shards`, `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mindspore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler, num_samples = num_samples)</code>
iterable 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler, num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards, shard_id = shard_id, shuffle = True, num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards, shard_id = shard_id, shuffle = False, num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True, num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

mindspore.dataset.Dataset.apply

`mindspore.dataset.Dataset.apply (apply_func)`

对数据集对象执行给定操作函数。

参数：

- **apply_func** (function) - 数据集处理函数，要求该函数的输入是一个 *Dataset* 对象，返回的是处理后的 *Dataset* 对象。

返回：

Dataset，应用了上述操作的新数据集对象。

异常：

- **TypeError** - *apply_func* 的类型不是函数。
- **TypeError** - *apply_func* 未返回 *Dataset* 对象。

样例：

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>>
>>> # Declare an apply_func function which returns a Dataset object
>>> def apply_func(data):
...     data = data.batch(2)
...     return data
>>>
>>> # Use apply to call apply_func
>>> dataset = dataset.apply(apply_func)
```

mindspore.dataset.Dataset.concat

`mindspore.dataset.Dataset.concat(datasets)`

对传入的多个数据集对象进行拼接操作。可以使用“+”运算符对数据集进行拼接。

说明：用于拼接的多个数据集对象，每个数据集对象的列名、每列数据的维度（rank）和数据类型必须相同。

参数：

- **datasets** (Union[list, Dataset]) - 与当前数据集对象拼接的数据集对象列表或单个数据集对象。

返回：

Dataset，应用了上述操作的新数据集对象。

样例：

```
>>> import mindspore.dataset as ds
>>> dataset_1 = ds.GeneratorDataset([1, 2, 3], "column1", shuffle=False)
>>> dataset_2 = ds.GeneratorDataset([4, 5, 6], "column1", shuffle=False)
>>>
>>> # Create a dataset by concatenating dataset_1 and dataset_2 with "+" operator
>>> dataset = dataset_1 + dataset_2
>>> # Create a dataset by concatenating dataset_1 and dataset_2 with concat operation
>>> dataset = dataset_1.concat(dataset_2)
>>>
>>> # Check the data order of dataset
>>> dataset_1 = ds.GeneratorDataset([1, 2, 3], "column1", shuffle=False)
>>> dataset_2 = ds.GeneratorDataset([4, 5, 6], "column1", shuffle=False)
>>> dataset = dataset_1 + dataset_2
>>> result = list(dataset)
>>> # [[Tensor(shape=[], dtype=Int64, value= 1)], [Tensor(shape=[], dtype=Int64, value= 2)]],
>>> # [Tensor(shape=[], dtype=Int64, value= 3)], [Tensor(shape=[], dtype=Int64, value= 4)]],
>>> # [Tensor(shape=[], dtype=Int64, value= 5)], [Tensor(shape=[], dtype=Int64, value= 6)]]
>>>
>>> # Change the data order of concatenated dataset with sharding selection
>>> dataset_1 = ds.GeneratorDataset([1, 2, 3], "column1", shuffle=False)
>>> dataset_2 = ds.GeneratorDataset([4, 5, 6], "column1", shuffle=False)
>>> dataset = dataset_1.concat(dataset_2)
>>> dataset.use_sampler(ds.DistributedSampler(num_shards=2, shard_id=1, shuffle=False))
>>> result = list(dataset)
>>> # [[Tensor(shape=[], dtype=Int64, value= 2)], [Tensor(shape=[], dtype=Int64, value= 4)]],
>>> # [Tensor(shape=[], dtype=Int64, value= 6)]]
>>>
>>> # Change the data order of concatenated dataset with random selection
>>> dataset_1 = ds.GeneratorDataset([1, 2, 3], "column1", shuffle=False)
>>> dataset_2 = ds.GeneratorDataset([4, 5, 6], "column1", shuffle=False)
>>> dataset = dataset_1.concat(dataset_2)
>>> dataset.use_sampler(ds.RandomSampler())
>>> result = list(dataset)
>>> # [[Tensor(shape=[], dtype=Int64, value= 1)], [Tensor(shape=[], dtype=Int64, value= 4)]],
>>> # [Tensor(shape=[], dtype=Int64, value= 2)], [Tensor(shape=[], dtype=Int64, value= 5)]],
>>> # [Tensor(shape=[], dtype=Int64, value= 6)], [Tensor(shape=[], dtype=Int64, value= 3)]]
```

`mindspore.dataset.Dataset.filter`

`mindspore.dataset.Dataset.filter` (*predicate*, *input_columns=None*, *num_parallel_workers=None*)

通过自定义判断条件对数据集对象中的数据进行过滤。

参数：

- **predicate** (callable) - Python 可调用对象。要求该对象接收 *n* 个入参，用于指代每个数据列的数据，最后返回值一个 bool 值。如果返回值为 `False`，则表示过滤掉该条数据。注意 *n* 的值与参数 *input_columns* 表示的输入列数量一致。
- **input_columns** (Union[str, list[str]], 可选) - *filter* 操作的输入数据列。默认值：`None`，*predicate* 将应用于数据集中的所有列。
- **num_parallel_workers** (int, 可选) - 指定 *filter* 操作的并发线程数。默认值：`None`，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。

返回：

`Dataset`，应用了上述操作的新数据集对象。

样例：

```
>>> # generator data (0 ~ 19)
>>> # filter the data that greater than or equal to 11
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(20)], "data")
>>> dataset = dataset.filter(predicate=lambda data: data < 11, input_columns = ["data"])
```

`mindspore.dataset.Dataset.flat_map`

`mindspore.dataset.Dataset.flat_map` (*func*)

对数据集对象中每一条数据执行给定的数据处理，并将结果展平。

参数：

- **func** (function) - 数据处理函数，要求输入必须为一个 `numpy.ndarray`，返回值是一个 `Dataset` 对象。

返回：

`Dataset`，应用了上述操作的新数据集对象。

异常：

- **TypeError** - *func* 不是函数。
- **TypeError** - *func* 的返回值不是 `Dataset` 对象。

样例：

```
>>> import mindspore.dataset as ds
>>> # 1) flat_map on one column dataset
>>> dataset = ds.NumpySlicesDataset([[0, 1], [2, 3]], shuffle=False)
>>>
>>> def repeat(array):
...     # create a NumpySlicesDataset with the array
...     data = ds.NumpySlicesDataset(array, shuffle=False)
...     # repeat the dataset twice
...     data = data.repeat(2)
...     return data
```

(下页继续)

(续上页)

```

>>>
>>> dataset = dataset.flat_map(repeat)
>>> # [0, 1, 0, 1, 2, 3, 2, 3]
>>>
>>> # 2) flat_map on multi column dataset
>>> dataset = ds.NumpySlicesDataset([[0, 1], [2, 3]], [[0, -1], [-2, -3]], shuffle=False)
>>>
>>> def plus_and_minus(col1, col2):
...     # apply different methods on columns
...     data = ds.NumpySlicesDataset((col1 + 1, col2 - 1), shuffle=False)
...     return data
>>>
>>> dataset = dataset.flat_map(plus_and_minus)
>>> # ([1, 2, 3, 4], [-1, -2, -3, -4])

```

mindspore.dataset.Dataset.map

`mindspore.dataset.Dataset.map` (*operations*, *input_columns=None*, *output_columns=None*, *column_order=None*, *num_parallel_workers=None*, ***kwargs*)

给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。

每个数据增强操作将数据集对象中的一个或多个数据列作为输入，将数据增强的结果输出为一个或多个数据列。第一个数据增强操作将 *input_columns* 中指定的列作为输入。如果数据增强列表中存在多个数据增强操作，则上一个数据增强的输出列将作为下一个数据增强的输入列。

最后一个数据增强的输出列的列名由 *output_columns* 指定，如果没有指定 *output_columns*，输出列名与 *input_columns* 一致。

- 如果使用的是 *mindspore dataset* 提供的数据增强（*vision* 类，*nlp* 类，*audio* 类），请使用如下参数：

operations + input_columns + output_columns + num_parallel_workers

必选	可选	可选	可选

- 如果使用的是自定义 *PyFunc* 数据增强，请使用如下参数：

operations + input_columns + output_columns + num_parallel_workers + python_multiprocessing + max_rowsize

必选	可选	可选	可选	可选	可选

参数：

- operations** (*Union*[*list*[*TensorOperation*], *list*[*functions*]]) - 一组数据增强操作，支持数据集增强操作或者用户自定义的 *Python Callable* 对象。map 操作将按顺序将一组数据增强作用在数据集对象上。
- input_columns** (*Union*[*str*, *list*[*str*]], 可选) - 第一个数据增强操作的输入数据列。此列表的长度必须与 *operations* 列表中第一个数据增强的预期输入列数相匹配。默认值：*None*。表示所有数据列都将传递给第一个数据增强操作。

- **output_columns** (Union[str, list[str]], 可选) - 最后一个数据增强操作的输出数据列。如果 *input_columns* 长度不等于 *output_columns* 长度, 则必须指定此参数。列表的长度必须与最后一个数据增强的输出列数相匹配。默认值: None, 输出列将与输入列具有相同的名称。
- **num_parallel_workers** (int, 可选) - 指定 map 操作的多进程/多线程并发数, 加快处理速度。默认值: None, 将使用 *set_num_parallel_workers* 设置的并发数。
- ****kwargs** - 其他参数。
 - **python_multiprocessing** (bool, 可选) - 启用 Python 多进程模式加速 map 操作。当传入的 *operations* 计算量很大时, 开启此选项可能会有较好效果。默认值: False。
 - **max_rowsize** (Union[int, list[int]], 可选) - 指定在多线程之间复制数据时, 共享内存分配的基本单位, 单位为 MB, 总占用的共享内存会随着 num_parallel_workers 和 *mindspore.dataset.config.set_prefetch_size()* 增加而变大。仅当 *python_multiprocessing* 为 True 时, 该选项有效。默认值: None, 动态分配共享内存 (后续版本将废弃此参数)。
 - * 如果设置为 -1 / None, 共享内存将随数据大小动态分配;
 - * 如果是 int 值, 代表 input_columns 和 output_columns 均使用该值为单位创建共享内存;
 - * 如果是列表, 第一个元素代表 input_columns 使用该值为单位创建共享内存, 第二个元素代表 output_columns 使用该值为单位创建共享内存。
 - **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。
 - **callbacks** (*DSCallback*, list[*DSCallback*], 可选) - 要调用的 Dataset 回调函数列表。默认值: None。
 - **offload** (bool, 可选) - 是否进行异构硬件加速, 详情请阅读 [数据准备异构加速](#)。默认值: None。

说明:

- *operations* 参数接收 *TensorOperation* 类型的数据处理操作, 以及用户定义的 Python 函数 (PyFuncs)。
- 通过 *ds.config.set_multiprocessing_start_method("spawn")* 方式设置多进程的启动方式为 *spawn* 模式, 且 *python_multiprocessing=True* 和 *num_parallel_workers>1* 时, 支持将 *mindspore.nn* 和 *mindspore.ops* 或其他的网络计算算子添加到 *operations* 中, 否则不支持添加到 *operations* 中。
- 当前仅部分场景支持在 *operations* 参数传入的 Python 函数中调用 DVPP 算子:

	多线程	多进程	
		spawn	fork
独立进程	数据处理: 支持 数据处理 + 网络训练: 不支持	数据处理: 支持 数据处理 + 网络训练: 支持	数据处理: 支持 数据处理 + 网络训练: 不支持
非独立进程	数据处理: 支持 数据处理 + 网络训练: 支持	数据处理: 支持 数据处理 + 网络训练: 支持	数据处理: 支持 数据处理 + 网络训练: 不支持

返回:

Dataset, 应用了上述操作的新数据集对象。

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> # dataset is an instance of Dataset which has 2 columns, "image" and "label".
```

(下页继续)

(续上页)

```

>>> # image is of type bytes type which can be decoded to RGB
>>> # label is of type int32
>>> cifar10_dataset_dir = "/path/to/cifar10_dataset_directory"
>>> dataset = ds.Cifar10Dataset(dataset_dir=cifar10_dataset_dir)
>>>
>>> # Define two operations, where each operation accepts 1 input column and outputs 1
↳column.
>>> decode_op = vision.Decode(to_pil=False)
>>> random_jitter_op = vision.RandomColorAdjust(brightness=(0.8, 0.8), contrast=(1, 1),
...                                             saturation=(1, 1), hue=(0, 0))
>>>
>>> # 1) Simple map example.
>>>
>>> # Apply decode_op on column "image".
>>> dataset = dataset.map(operations=[decode_op], input_columns=["image"])
>>>
>>> # Decode and rename column "image" to "decoded_image".
>>> dataset = dataset.map(operations=[decode_op], input_columns=["image"], output_columns=[
↳"decoded_image"])
>>>
>>> # A simple example for user defined python function transform.
>>> dataset = ds.NumpySlicesDataset(data=[[0, 1, 2]], column_names=["data"])
>>> dataset = dataset.map(operations=[(lambda x: x - 1)], input_columns=["data"])
>>>
>>> # 2) Map example with more than one operation.
>>>
>>> # Create a dataset where the images are decoded, then randomly color jittered.
>>> # decode_op takes column "image" as input and outputs one column. The column
>>> # outputted by decode_op is passed as input to random_jitter_op.
>>> # random_jitter_op will output one column. Column "image" will be replaced by
>>> # the column outputted by random_jitter_op (the very last operation). All other
>>> # columns are unchanged.
>>> dataset = dataset.map(operations=[decode_op, random_jitter_op], input_columns=["image"])
>>>
>>> # Rename the column outputted by random_jitter_op to "image_mapped".
>>> dataset = dataset.map(operations=[decode_op, random_jitter_op], input_columns=["image"],
...                       output_columns=["image_mapped"])
>>>
>>> # Map with multiple operations using pyfunc and rename column's name
>>> dataset = ds.NumpySlicesDataset(data=[[0, 1, 2]], column_names=["data"])
>>> dataset = dataset.map(operations=[(lambda x: x * x), (lambda x: x - 1)], input_columns=[
↳"data"],
...                               output_columns=["data_mapped"])
>>>
>>> # 3) Example where number of input columns is not equal to number of output columns.
>>>
>>> # operations[0] is a lambda that takes 2 columns as input and outputs 3 columns.
>>> # operations[1] is a lambda that takes 3 columns as input and outputs 1 column.
>>> # operations[2] is a lambda that takes 1 column as input and outputs 4 columns.
>>> #
>>> # Note: The number of output columns of operation[i] must equal the number of
>>> # input columns of operation[i+1]. Otherwise, this map call will also result
>>> # in an error.
>>> operations = [(lambda x, y: (x, x + y, x + y + 1)),
...              (lambda x, y, z: x * y * z),
...              (lambda x: (x % 2, x % 3, x % 5, x % 7))]

```

(下页继续)

(续上页)

```
>>> dataset = ds.NumpySlicesDataset(data=([[0, 1, 2]], [[3, 4, 5]]), column_names=["x", "y"])
>>> dataset = dataset.map(operations, input_columns=["x", "y"],
...                       output_columns=["mod2", "mod3", "mod5", "mod7"])
```

mindspore.dataset.Dataset.project

`mindspore.dataset.Dataset.project(columns)`

从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。未指定的数据列将被丢弃。

参数：

- **columns** (Union[str, list[str]]) - 要选择的数据列的列名列表。

返回：

Dataset，应用了上述操作的新数据集对象。

样例：

```
>>> import mindspore.dataset as ds
>>> # Create a dataset with 3 columns
>>> input_columns = ["column1", "column2", "column3"]
>>> dataset = ds.GeneratorDataset([(1, 2, 3), (3, 4, 5), (5, 6, 7)], column_names=input_
↳ columns)
>>>
>>> columns_to_project = ["column3", "column1", "column2"]
>>> # in that order, regardless of the original order of columns.
>>> dataset = dataset.project(columns=columns_to_project)
```

mindspore.dataset.Dataset.rename

`mindspore.dataset.Dataset.rename(input_columns, output_columns)`

对数据集对象按指定的列名进行重命名。

参数：

- **input_columns** (Union[str, list[str]]) - 待重命名的列名列表。
- **output_columns** (Union[str, list[str]]) - 重命名后的列名列表。

返回：

Dataset，应用了上述操作的新数据集对象。

样例：

```
>>> import mindspore.dataset as ds
>>> input_columns = ["input_col1", "input_col2", "input_col3"]
>>> output_columns = ["output_col1", "output_col2", "output_col3"]
>>>
>>> # Create a dataset with 3 columns
>>> dataset = ds.GeneratorDataset([(1, 2, 3), (3, 4, 5), (5, 6, 7)], column_names=input_
↳ columns)
>>>
>>> # Rename "input_col1" to "output_col1", "input_col2" to "output_col2", "input_col3" to
↳ "output_col3"
>>> dataset = dataset.rename(input_columns=input_columns, output_columns=output_columns)
```

mindspore.dataset.Dataset.repeat

`mindspore.dataset.Dataset.repeat(count=None)`

重复此数据集 *count* 次。如果 *count* 为 `None` 或 `-1`，则无限重复迭代。

说明: `repeat` 和 `batch` 的顺序反映了 `batch` 的数量。建议: `repeat` 操作在 `batch` 操作之后使用。

参数:

- **count** (int) - 数据集重复的次数。默认值: `None`。

返回:

`Dataset`，应用了上述操作的新数据集对象。

样例:

```
>>> import mindspore.dataset as ds
>>>
>>> # Create a dataset with 10 elements
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> ori_size = dataset.get_dataset_size()
>>>
>>> # Repeat the dataset 50 times.
>>> dataset = dataset.repeat(50)
>>> repeated_size = dataset.get_dataset_size()
>>> print("ori_size", ori_size, ", repeated_size", repeated_size)
ori_size 10 , repeated_size 500
>>>
>>> # Since the original dataset size is less than batch_size, thus no data is returned
>>> dataset1 = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> dataset1 = dataset1.batch(batch_size=20, drop_remainder=True)
>>> dataset1 = dataset1.repeat(6)
>>>
>>> # Repeat the original dataset to 60 elements, thus 3 batches are returned
>>> dataset2 = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> dataset2 = dataset2.repeat(6)
>>> dataset2 = dataset2.batch(batch_size=20, drop_remainder=True)
>>> print("dataset1 size", dataset1.get_dataset_size(), ", dataset2 size", dataset2.get_
↪dataset_size())
dataset1 size 0 , dataset2 size 3
```

mindspore.dataset.Dataset.reset

`mindspore.dataset.Dataset.reset()`

重置下一个 epoch 的数据集对象。

样例:

```
>>> import mindspore.dataset as ds
>>> mind_dataset_dir = ["/path/to/mind_dataset_file"]
>>> dataset = ds.MindDataset(dataset_files=mind_dataset_dir)
>>> for _ in range(5):
...     num_iter = 0
```

(下页继续)

(续上页)

```
...     for data in dataset.create_tuple_iterator(num_epochs=1, output_numpy=True):
...         num_iter += 1
...     dataset.reset()
```

mindspore.dataset.Dataset.save

`mindspore.dataset.Dataset.save (file_name, num_files=1, file_type='mindrecord')`
将数据处理管道中正处理的数据保存为通用的数据集格式。数据集格式仅支持: 'mindrecord'。可以使用 `mindspore.dataset.MindDataset` 类来读取保存的 'mindrecord' 文件。
将数据保存为 'mindrecord' 格式时存在隐式类型转换。转换表展示如何执行类型转换。

表 4: 保存为'mindrecord' 格式时的隐式类型转换

'dataset' 类型	'mindrecord' 类型	说明
bool	int32	变更为 int32
int8	int32	
uint8	int32	
int16	int32	
uint16	int32	
int32	int32	
uint32	int64	
int64	int64	
uint64	int64	有可能反转
float16	float32	
float32	float32	
float64	float64	
string	string	不支持多维字符串
bytes	bytes	不支持多维 bytes

- 说明:
- 1. 如需按顺序保存数据, 将数据集的 `shuffle` 设置为 `False`, 将 `num_files` 设置为 1。
 - 2. 在执行保存操作之前, 不要使用 `batch` 操作、`repeat` 操作或具有随机属性的数据增强的 `map` 操作。
 - 3. 当数据的维度可变时, 只支持一维数组或者在第零维变化的多维数组。
 - 4. 不支持多维 `string` 类型、多维 `bytes` 类型。

- 参数:
- `file_name` (str) - 数据集文件的路径。
 - `num_files` (int, 可选) - 数据集文件的数量。默认值: 1。
 - `file_type` (str, 可选) - 数据集格式。默认值: 'mindrecord'。

样例:

```
>>> import mindspore.dataset as ds
>>> import numpy as np
>>>
```

(下页继续)

(续上页)

```
>>> def generator_1d():
...     for i in range(10):
...         yield (np.array([i]),)
>>>
>>> # apply dataset operations
>>> d1 = ds.GeneratorDataset(generator_1d, ["data"], shuffle=False)
>>> d1.save('/path/to/save_file')
```

mindspore.dataset.Dataset.shuffle

mindspore.dataset.Dataset.**shuffle**(*buffer_size*)

通过创建 *buffer_size* 大小的缓存来混洗该数据集。

1. 生成一个混洗缓冲区包含 *buffer_size* 条数据行。
2. 从混洗缓冲区中随机选择一个数据行，传递给下一个操作。
3. 从上一个操作获取下一个数据行（如果有的话），并将其放入混洗缓冲区中。
4. 重复步骤 2 和 3，直到混洗缓冲区中没有数据行为止。

在第一个 epoch 中可以通过 *dataset.config.set_seed* 来设置随机种子。在随后的每个 epoch，种子都会被设置成一个新产生的随机值。

参数：

- **buffer_size** (int) - 用于混洗的缓冲区大小（必须大于 1）。将 *buffer_size* 设置为数据集大小将进行全局混洗。

返回：

Dataset，应用了上述操作的新数据集对象。

异常：

- **RuntimeError** - 混洗前存在通过 *dataset.sync_wait* 进行同步操作。

样例：

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>>
>>> # Optionally set the seed for fixed randomness
>>> ds.config.set_seed(58)
>>>
>>> # Create a shuffled dataset using a shuffle buffer of size 4
>>> dataset = dataset.shuffle(4)
```

mindspore.dataset.Dataset.skip

mindspore.dataset.Dataset.**skip**(*count*)

跳过此数据集对象的前 *count* 条数据。

参数：

- **count** (int) - 要跳过数据的条数。

返回：

Dataset，应用了上述操作的新数据集对象。

样例:

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> # Skip first 3 elements of dataset and retain 7 elements.
>>> dataset = dataset.skip(3)
```

`mindspore.dataset.Dataset.split`

`mindspore.dataset.Dataset.split(sizes, randomize=True)`

将数据集拆分为多个不重叠的子数据集。

参数:

- **sizes** (Union[list[int], list[float]]) - 如果指定了一列整数 [s1, s2, ..., sn], 数据集将被拆分为 n 个大小为 s1、s2、...、sn 的数据集。如果所有输入大小的总和不等于原始数据集大小, 则报错。如果指定了一列浮点数 [f1, f2, ..., fn], 则所有浮点数必须介于 0 和 1 之间, 并且总和必须为 1, 否则报错。数据集将被拆分为 n 个大小为 $\text{round}(f1 * K)$ 、 $\text{round}(f2 * K)$ 、...、 $\text{round}(fn * K)$ 的数据集, 其中 K 是原始数据集的大小。如果 round 四舍五入计算后:
 - 任何子数据集的大小等于 0, 都将发生错误。
 - 如果子数据集大小的总和小于 K, $K - \text{sigma}(\text{round}(fi * k))$ 的值将添加到第一个子数据集, sigma 为求和操作。
 - 如果子数据集大小的总和大于 K, $\text{sigma}(\text{round}(fi * K)) - K$ 的值将从第一个足够大的子数据集中删除, 且删除后的子数据集大小至少大于 1。
- **randomize** (bool, 可选) - 确定是否随机拆分数据。默认值: True, 数据集将被随机拆分。否则将按顺序拆分为多个不重叠的子数据集。

说明:

1. 如果进行拆分操作的数据集对象为 `MappableDataset` 类型, 则将自动调用一个优化后的 `split` 操作。
2. 如果进行 `split` 操作, 则不对数据集对象进行分片操作 (如指定 `num_shards` 或使用 `mindspore.dataset.DistributedSampler`)。相反, 如果创建一个 `mindspore.dataset.DistributedSampler`, 并在 `split` 操作拆分后的子数据集对象上进行分片操作, 强烈建议在每个子数据集上设置相同的种子, 否则每个分片可能不是同一个子数据集的一部分 (请参见示例)。
3. 强烈建议不要对数据集进行混洗, 而是使用随机化 (`randomize` 为 True)。对数据集进行混洗的结果具有不确定性, 每个拆分后的子数据集中的数据在每个 `epoch` 可能都不同。

异常:

- **RuntimeError** - 数据集对象不支持 `get_dataset_size` 或者 `get_dataset_size` 返回 None。
- **RuntimeError** - `sizes` 是 list[int], 并且 `sizes` 中所有元素的总和不等于数据集大小。
- **RuntimeError** - `sizes` 是 list[float], 并且计算后存在大小为 0 的拆分子数据集。
- **RuntimeError** - 数据集对象在调用拆分之前已进行分片。
- **ValueError** - `sizes` 是 list[float], 且并非所有 float 数值都在 0 和 1 之间, 或者 float 数值的总和不等于 1。

返回:

Tuple[Dataset], 从原数据集拆分出的新数据集构成的元组。

样例:

```
>>> # Split the data into train part and test part.
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> train_dataset, test_dataset = dataset.split([0.9, 0.1])
```

mindspore.dataset.Dataset.take

`mindspore.dataset.Dataset.take(count=-1)`

截取数据集的前指定条数据。

参数:

- **count** (int, 可选) - 想要截取的数据条数。若该值超过数据集样本总数，则返回全部数据。默认值: -1，将返回全部数据。

说明: 当数据处理流水线中存在会改变数据集样本数量的操作时, `take` 操作所处的位置会影响其效果。例如, `batch` 操作会将连续指定 `batch_size` 条样本合并成 1 条样本, 则 `.batch(batch_size).take(1)` 与 `.take(batch_size).batch(batch_size)` 效果相当。

返回:

`Dataset`, 应用了上述操作的新数据集对象。

样例:

```
>>> import mindspore.dataset as ds
>>> mnist_dataset_dir = "/path/to/mnist_dataset_directory"
>>> dataset = ds.MnistDataset(dataset_dir=mnist_dataset_dir)
>>> # Take 50 samples from MNIST dataset.
>>> dataset = dataset.take(50)
```

mindspore.dataset.Dataset.zip

`mindspore.dataset.Dataset.zip(datasets)`

将多个 `dataset` 对象按列进行合并压缩, 多个 `dataset` 对象不能有相同的列名。

参数:

- **datasets** (Union[Dataset, tuple[Dataset]]) - 要合并的 (多个) `dataset` 对象。

返回:

`Dataset`, 应用了上述操作的新数据集对象。

异常:

- **TypeError** - `datasets` 参数不是 `dataset` 对象/tuple[dataset]。

样例:

```
>>> # Create a dataset which is the combination of dataset_1 and dataset_2
>>> import mindspore.dataset as ds
>>> dataset_1 = ds.GeneratorDataset([1, 2, 3], "column1")
>>> dataset_2 = ds.GeneratorDataset([1, 2, 3], "column2")
>>> dataset = dataset_1.zip(dataset_2)
```

Batch (批操作)

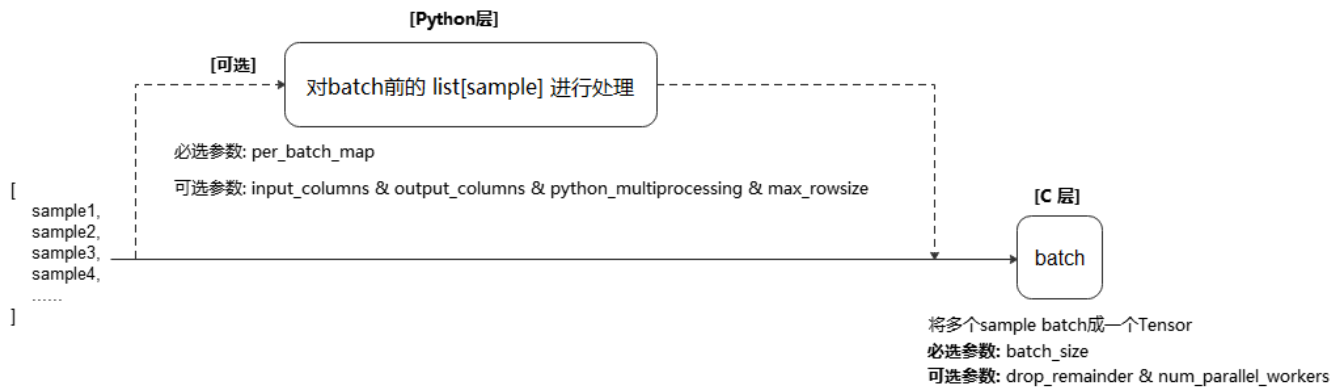
<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

`mindspore.dataset.Dataset.batch`

`mindspore.dataset.Dataset.batch` (*batch_size*, *drop_remainder=False*, *num_parallel_workers=None*, ***kwargs*)
将数据集中连续 *batch_size* 条数据组合为一个批数据，并可通过可选参数 *per_batch_map* 指定组合前要进行的预处理操作。

batch 操作要求每列中的数据具有相同的 *shape*。

执行流程参考下图：



说明：

- 执行 *repeat* 和 *batch* 操作的先后顺序，会影响批处理数据的数量及 *per_batch_map* 的结果。建议在 *batch* 操作完成后执行 *repeat* 操作。
- 在静态图模式使用数据下沉时，网络输入的 *shape* 应保持一致，此时应设置 *drop_remainder* 为 *True* 以丢弃最后一个不完整的批数据，或通过补充/删除样本以确保数据集大小能被 *batch_size* 整除。

参数：

- **batch_size** (Union[int, Callable]) - 指定每个批处理数据包含的数据条目。如果 *batch_size* 为整型, 则直接表示每个批处理数据大小; 如果为可调用对象, 则可以通过自定义行为动态指定每个批处理数据大小, 要求该可调用对象接收一个参数 **BatchInfo**, 返回一个整形代表批处理大小, 用法请参考样例 (3)。
- **drop_remainder** (bool, 可选) - 当最后一个批处理数据包含的数据条目小于 *batch_size* 时, 是否将该批处理丢弃, 不传递给下一个操作。默认值: `False`, 不丢弃。
- **num_parallel_workers** (int, 可选) - 指定 *batch* 操作的并发进程数/线程数 (由参数 *python_multiprocessing* 决定当前为多进程模式或多线程模式)。默认值: `None`, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- ****kwargs** - 其他参数。
 - **per_batch_map** (Callable[[List[numpy.ndarray], ..., List[numpy.ndarray], BatchInfo], (List[numpy.ndarray], ..., List[numpy.ndarray])], 可选) - 可调用对象, 以 (list[numpy.ndarray], ..., list[numpy.ndarray], BatchInfo) 作为输入参数, 处理后返回 (list[numpy.ndarray], list[numpy.ndarray], ...) 作为新的数据列。输入参数中每个 list[numpy.ndarray] 代表给定数据列中的一批 numpy.ndarray, list[numpy.ndarray] 的个数应与 *input_columns* 中传入列名的数量相匹配, 在返回的 (list[numpy.ndarray], list[numpy.ndarray], ...) 中, list[numpy.ndarray] 的个数应与输入相同, 如果输出列数与输入列数不一致, 则需要指定 *output_columns*。该可调用对象的最后一个输入参数始终是 **BatchInfo**, 用于获取数据集的信息, 用法参考样例 (2)。
 - **input_columns** (Union[str, list[str]], 可选) - 指定 *batch* 操作的输入数据列。如果 *per_batch_map* 不为 `None`, 列表中列名的个数应与 *per_batch_map* 中包含的列数匹配。默认值: `None`, 不指定。
 - **output_columns** (Union[str, list[str]], 可选) - 指定 *batch* 操作的输出数据列。如果输入数据列与输入数据列的长度不相等, 则必须指定此参数。此列表中列名的数量必须与 *per_batch_map* 方法的返回值数量相匹配。默认值: `None`, 输出列将与输入列具有相同的名称。
 - **python_multiprocessing** (bool, 可选) - 是否启动 Python 多进程模式并行执行 *per_batch_map*, `True` 意为 Python 多进程模式, `False` 意为 Python 多线程模式。如果 *per_batch_map* 是 I/O 密集型任务可以用多线程, CPU 密集型任务建议使用多进程, 以避免 GIL 锁影响。默认值: `False`, 启用多线程模式。
 - **max_rowsize** (Union[int, list[int]], 可选) - 指定在多进程之间复制数据时, 共享内存分配的基本单位, 单位为 MB, 总占用的共享内存会随着 *num_parallel_workers* 和 `mindspore.dataset.config.set_prefetch_size()` 增加而变大。仅当 *python_multiprocessing* 为 `True` 时, 该选项有效。默认值: `None`, 动态分配共享内存 (后续版本将废弃此参数)。
 - * 如果设置为 `-1 / None`, 共享内存将随数据大小动态分配;
 - * 如果是 `int` 值, 代表 *input_columns* 和 *output_columns* 均使用该值为单位创建共享内存;
 - * 如果是列表, 代表 *input_columns* 使用第一个元素为单位创建共享内存, *output_columns* 使用第二个元素为单位创建共享内存。

返回:

`Dataset`, 应用了上述操作的新数据集对象。

样例:

```
>>> # 1) Create a dataset where every 5 rows are combined into a batch
>>> # and drops the last incomplete batch if there is one.
>>> import mindspore.dataset as ds
>>> from PIL import Image
>>>
>>> cifar10_dataset_dir = "/path/to/cifar10_dataset_directory"
>>> dataset = ds.Cifar10Dataset(dataset_dir=cifar10_dataset_dir, num_samples=10)
>>> dataset = dataset.batch(5, True)
>>>
>>> # 2) resize image according to its batch number, if it's 5-th batch, resize to (5^2, 5^
↪2) = (25, 25)
```

(下页继续)

(续上页)

```

>>> def np_resize(col, BatchInfo):
...     output = col.copy()
...     s = (BatchInfo.get_batch_num() + 1) ** 2
...     index = 0
...     for c in col:
...         img = Image.fromarray(c.astype('uint8')).convert('RGB')
...         img = img.resize((s, s))
...         output[index] = np.array(img)
...         index += 1
...     return (output,)
>>> dataset = dataset.batch(batch_size=8, input_columns=["image"], per_batch_map=np_resize)
>>>
>>> # 3) Create a dataset where its batch size is dynamic
>>> # Define a callable batch size function and let batch size increase 1 each time.
>>> def add_one(BatchInfo):
...     return BatchInfo.get_batch_num() + 1
>>> dataset = dataset.batch(batch_size=add_one, drop_remainder=True)

```

mindspore.dataset.Dataset.bucket_batch_by_length

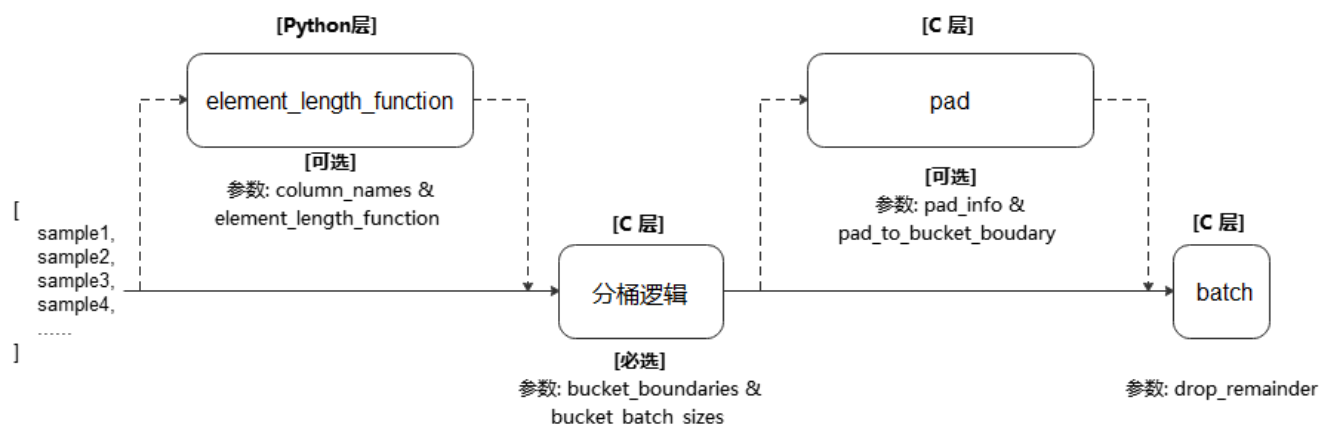
`mindspore.dataset.Dataset.bucket_batch_by_length` (*column_names, bucket_boundaries, bucket_batch_sizes, element_length_function=None, pad_info=None, pad_to_bucket_boundary=False, drop_remainder=False*)

根据数据的长度进行分桶。每个桶将在数据填满的时候进行填充和批处理操作。

对数据集中的每一条数据进行长度计算。根据该条数据的长度计算结果和每个分桶的范围将该数据归类到特定的桶里面。

当某个分桶中数据条数达到指定的大小 *bucket_batch_sizes* 时，将根据 *pad_info* 的信息对分桶进行填充，再进行批处理。

执行流程参考下图：



说明：

- 在静态图模式使用数据下沉时，网络输入的 *shape* 应保持一致，此时应设置 *drop_remainder* 为“True”以丢弃最后一个不完整的批数据，或通过补充/删除样本以确保数据集大小能被 *batch_size* 整除。

参数:

- **column_names** (list[str]) - 传递给参数 *element_length_function* 的数据列，用于计算数据的长度。
- **bucket_boundaries** (list[int]) - 指定各个分桶的上边界值，列表的数值必须严格递增。如果有 *n* 个边界，则会创建 *n*+1 个桶，分配后桶的边界如下：[0, bucket_boundaries[0]), [bucket_boundaries[i], bucket_boundaries[i+1]), [bucket_boundaries[n-1], inf)，其中， $0 < i < n-1$ 。
- **bucket_batch_sizes** (list[int]) - 指定每个分桶的批数据大小，必须包含 *len(bucket_boundaries)+1* 个元素。
- **element_length_function** (Callable, 可选) - 长度计算函数。要求接收 *len(column_names)* 个输入参数，并返回一个整数代表该条数据的长度。如果未指定该参数，则参数 *column_names* 的长度必须为 1，此时该列数据的 *shape[0]* 值将被当做数据长度。默认值：None，不指定。
- **pad_info** (dict, 可选) - 对指定数据列进行填充。通过传入 dict 来指定列信息与填充信息，要求 dict 的键是要填充的数据列名，dict 的值是包含 2 个元素的元组。元组中第 1 个元素表示要扩展至的目标 *shape*，第 2 个元素表示要填充的值。如果某一个数据列未指定将要填充后的 *shape* 和填充值，则该列中的每条数据都将填充至该批次中最长数据的长度，且填充值为 0。注意，*pad_info* 中任何填充 *shape* 为 None 的列，其每条数据长度都将被填充为当前批处理中最长数据的长度，除非指定 *pad_to_bucket_boundary* 为 True。默认值：None，不填充。
- **pad_to_bucket_boundary** (bool, 可选) - 如果为 True，则 *pad_info* 中填充 *shape* 为 None 的列，会被填充至由参数 *bucket_batch_sizes* 指定的对应分桶长度-1 的长度。如果有任何数据落入最后一个分桶中，则将报错。默认值：False。
- **drop_remainder** (bool, 可选) - 当每个分桶中的最后一个批处理数据条目小于 *bucket_batch_sizes* 时，是否丢弃该批处理数据。默认值：False，不丢弃。

返回:

Dataset，应用了上述操作的新数据集对象。

样例:

```
>>> # Create a dataset where certain counts rows are combined into a batch
>>> # and drops the last incomplete batch if there is one.
>>> import mindspore.dataset as ds
>>> import numpy as np
>>> def generate_2_columns(n):
...     for i in range(n):
...         yield (np.array([i]), np.array([j for j in range(i + 1)]))
>>>
>>> column_names = ["col1", "col2"]
>>> dataset = ds.GeneratorDataset(generate_2_columns(8), column_names)
>>> bucket_boundaries = [5, 10]
>>> bucket_batch_sizes = [2, 1, 1]
>>> element_length_function = (lambda col1, col2: max(len(col1), len(col2)))
>>> # Will pad col2 to shape [bucket_boundaries[i]] where i is the
>>> # index of the bucket that is currently being batched.
>>> pad_info = {"col2": ([None], -1)}
>>> pad_to_bucket_boundary = True
>>> dataset = dataset.bucket_batch_by_length(column_names, bucket_boundaries,
...                                         bucket_batch_sizes,
...                                         element_length_function, pad_info,
...                                         pad_to_bucket_boundary)
```

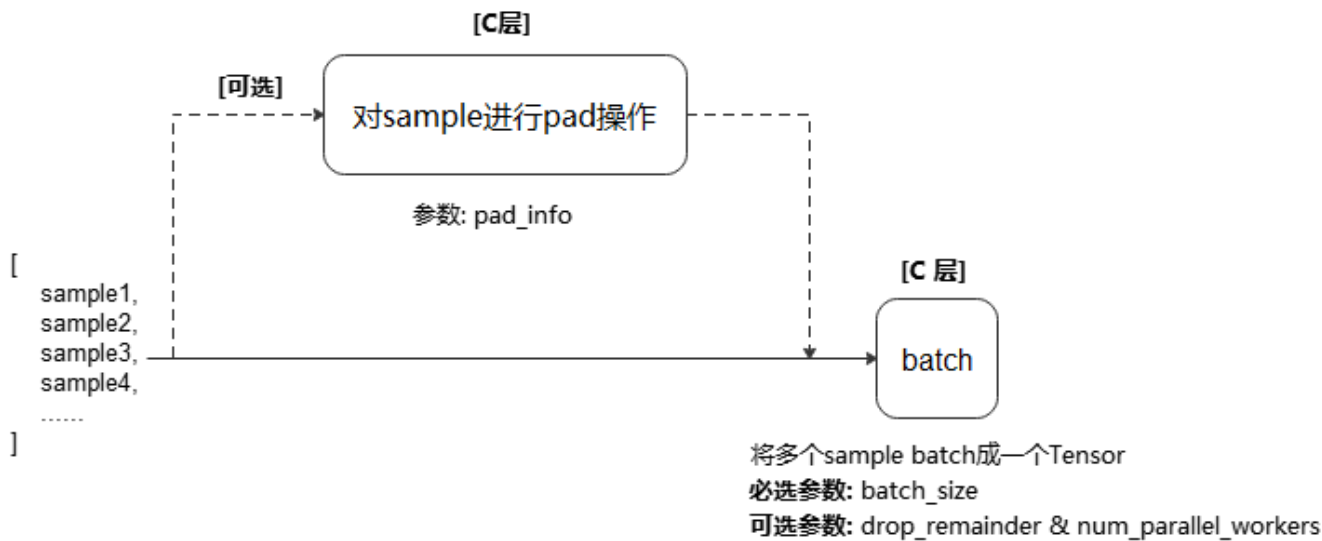

mindspore.dataset.Dataset.padded_batch

`mindspore.dataset.Dataset.padded_batch (batch_size, drop_remainder=False, num_parallel_workers=None, pad_info=None)`

将数据集中连续 *batch_size* 条数据组合为一个批数据，并可通过可选参数 *pad_info* 预先将样本补齐。

batch 操作要求每列中的数据具有相同的 *shape*。

执行流程参考下图：



说明：

- 执行 *repeat* 和 *padded_batch* 操作的先后顺序，会影响批处理数据的数量。建议在 *padded_batch* 操作完成后执行 *repeat* 操作。
- 在静态图模式使用数据下沉时，网络输入的 *shape* 应保持一致，此时应设置 *drop_remainder* 为 *True* 以丢弃最后一个不完整的批数据，或通过补充/删除样本以确保数据集大小能被 *batch_size* 整除。

参数：

- **batch_size** (Union[int, Callable]) - 指定每个批处理数据包含的数据条目。如果 *batch_size* 为整型，则直接表示每个批处理数据大小；如果为可调对象，则可以通过自定义行为动态指定每个批处理数据大小，要求该可调对象接收一个参数 *BatchInfo*，返回一个整形代表批处理大小，用法请参考样例 (3)。
- **drop_remainder** (bool, 可选) - 当最后一个批处理数据包含的数据条目小于 *batch_size* 时，是否将该批处理丢弃，不传递给下一个操作。默认值：False，不丢弃。
- **num_parallel_workers** (int, 可选) - 指定 *padded_batch* 操作的并发线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **pad_info** (dict, 可选) - 对给定数据列进行填充。通过传入 dict 来指定列信息与填充信息，例如 *pad_info*={"col1":(224,224),0}，则将列名为"col1"的数据列扩充到 *shape* 为 (224, 224) 的 Tensor，缺失的值使用 0 填充。如果 *pad_info*={}，则每个 *batch* 中的所有样本会补齐至当前 *batch* 中样本最大的 *shape*。如果 *pad_info*={"col1": (None, 100)}，则每个 *batch* 中的所有样本会补齐至当前 *batch* 中样本最大的 *shape*，缺失的值使用 100 填充。默认值：None，不填充。

返回：

Dataset, 应用了上述操作的新数据集对象。

样例:

```
>>> # 1) Pad every sample to the largest sample's shape and batch the samples
>>> import mindspore.dataset as ds
>>> dataset = ds.NumpySlicesDataset([[1], [1, 2], [1, 2, 3], [1, 2, 3, 4]], "column1")
>>> dataset = dataset.padded_batch(2, True, pad_info={})
>>>
>>> # 2) Create a dataset where every 3 rows are combined into a batch
>>> # and drops the last incomplete batch if there is one.
>>> dataset = ds.NumpySlicesDataset([i for i in range(10)], "column1")
>>> dataset = dataset.padded_batch(3, True)
>>>
>>> # 3) Create a dataset where its batch size is dynamic
>>> # Define a callable batch size function and let batch size increase 1 each time.
>>> def add_one(BatchInfo):
...     return BatchInfo.get_batch_num() + 1
>>> dataset = dataset.padded_batch(batch_size=add_one, drop_remainder=True)
```

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

`mindspore.dataset.Dataset.create_dict_iterator`

`mindspore.dataset.Dataset.create_dict_iterator(num_epochs=-1, output_numpy=False, do_copy=True)`

创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。

参数:

- **num_epochs** (int, 可选) - 数据集迭代次数。默认值: -1, 数据集可以无限迭代。
- **output_numpy** (bool, 可选) - 是否保持输出数据类型为 NumPy 数组, 否则转换为 `mindspore.Tensor`。默认值: False。
- **do_copy** (bool, 可选) - 指定转换输出类型为 `mindspore.Tensor` 时是否拷贝数据, 否则直接复用数据缓冲区以获得更好的性能, 仅当 `output_numpy` 为 False 时有效。默认值: True。

返回:

Iterator, 返回字典形式样本的迭代器。

样例:

```
>>> import mindspore.dataset as ds
>>>
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "data")
>>> num_epochs = 3
>>> iterator = dataset.create_dict_iterator(num_epochs=num_epochs)
>>> for epoch in range(num_epochs):
```

(下页继续)

(续上页)

```
...     for item in iterator:
...         # output is of type dict
...         print(type(item))
...         break
...     break
<class 'dict'>
```

mindspore.dataset.Dataset.create_tuple_iterator

`mindspore.dataset.Dataset.create_tuple_iterator` (*columns=None, num_epochs=- 1, output_numpy=False, do_copy=True*)

创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

参数：

- **columns** (list[str], 可选) - 指定输出数据列及其顺序。默认值：None，保留所有输出列及其原始顺序。
- **num_epochs** (int, 可选) - 数据集迭代次数。默认值：-1，数据集可以无限迭代。
- **output_numpy** (bool, 可选) - 是否保持输出数据类型为 NumPy 数组，否则转换为 *mindspore.Tensor*。默认值：False。
- **do_copy** (bool, 可选) - 指定转换输出类型为 *mindspore.Tensor* 时是否拷贝数据，否则直接复用数据缓冲区以获得更好的性能，仅当 *output_numpy* 为 False 时有效。默认值：True。

返回：

Iterator，返回列表形式样本的迭代器。

样例：

```
>>> import mindspore.dataset as ds
>>>
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "data")
>>> num_epochs = 3
>>> iterator = dataset.create_tuple_iterator(num_epochs=num_epochs)
>>> for epoch in range(num_epochs):
...     for item in iterator:
...         # output is of type tuple
...         print(type(item))
...         break
...     break
<class 'list'>
```

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。

下页继续

表 7 - 续上页

<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <code>repeat</code> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <code>shape</code> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

mindspore.dataset.Dataset.get_batch_size

`mindspore.dataset.Dataset.get_batch_size()`
获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。

返回：

`int`，一个批处理数据中包含的数据条数。

样例：

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> dataset = dataset.batch(2)
>>> batch_size = dataset.get_batch_size()
>>> print(batch_size)
2
```

mindspore.dataset.Dataset.get_class_indexing

`mindspore.dataset.Dataset.get_class_indexing()`
获取类别名称到类别索引的映射字典。

该字典可用于查询某一类别索引对应的是哪一个类别名称。

返回：

`Dict[str, int]`，类别名称到类别索引的映射关系。

样例：

```
>>> import mindspore.dataset as ds
>>> # Read image files
>>> image_folder_dataset_dir = "/path/to/image_folder_dataset_directory"
>>> dataset = ds.ImageFolderDataset(dataset_dir=image_folder_dataset_dir)
>>> # Check how many classes exist in image folder
>>> class_indexing = dataset.get_class_indexing()
```

mindspore.dataset.Dataset.get_col_names

mindspore.dataset.Dataset.get_col_names()

返回数据集对象中包含的列名。

返回:

list, 数据集中所有列名组成列表。

样例:

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> col_names = dataset.get_col_names()
>>> print(col_names)
['column1']
```

mindspore.dataset.Dataset.get_dataset_size

mindspore.dataset.Dataset.get_dataset_size()

返回一个 epoch 中的 batch 数。

返回:

int, batch 的数目。

样例:

```
>>> import mindspore.dataset as ds
>>> import numpy as np
>>>
>>> # A generator return 66 samples
>>> def generator1():
...     for i in range(66):
...         yield np.ones((16, 83, 83)), np.array([i])
>>>
>>> dataset = ds.GeneratorDataset(generator1, ["data1", "data2"])
>>> dataset_size = dataset.get_dataset_size()
>>> print(dataset_size)
66
```

mindspore.dataset.Dataset.get_repeat_count

mindspore.dataset.Dataset.get_repeat_count()

获取 RepeatDataset 中定义的 repeat 操作的次数。默认值: 1。

返回:

int, repeat 操作的次数。

样例:

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> dataset = dataset.repeat(5)
>>> repeat_count = dataset.get_repeat_count()
```

(下页继续)

(续上页)

```
>>> print(repeat_count)
5
```

`mindspore.dataset.Dataset.input_indexs`

property `mindspore.dataset.Dataset.input_indexs`

获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。

返回：

int，数据集的 input index 信息。

样例：

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>> # set input_indexs
>>> dataset.input_indexs = 10
>>> print(dataset.input_indexs)
10
```

`mindspore.dataset.Dataset.num_classes`

`mindspore.dataset.Dataset.num_classes()`

获取数据集对象中所有样本的类别数目。

返回：

int，类别的数目。

样例：

```
>>> import mindspore.dataset as ds
>>> # Read image files
>>> image_folder_dataset_dir = "/path/to/image_folder_dataset_directory"
>>> dataset = ds.ImageFolderDataset(dataset_dir=image_folder_dataset_dir)
>>> # Check how many classes exist in image folder
>>> num_classes = dataset.num_classes()
```

`mindspore.dataset.Dataset.output_shapes`

`mindspore.dataset.Dataset.output_shapes (estimate=False)`

获取数据集对象中每列数据的 shape。

参数：

- **estimate** (bool, 可选) - 如果 *estimate* 为 False，将返回数据集第一条数据的 shape。否则将遍历整个数据集以获取数据集的真实 shape 信息，其中动态变化的维度将被标记为 None（可用于动态 shape 数据集场景）。默认值：False。

返回：

list，每列数据的 shape 列表。

样例：

```
>>> import mindspore.dataset as ds
>>> import numpy as np
>>>
>>> def generator1():
...     for i in range(1, 100):
...         yield np.ones((16, 83, 83)), np.array([i])
>>>
>>> dataset = ds.GeneratorDataset(generator1, ["data1", "data2"])
>>> output_shapes = dataset.output_shapes()
>>> print(output_shapes)
[[16, 83, 83], [1]]
```

`mindspore.dataset.Dataset.output_types`

`mindspore.dataset.Dataset.output_types()`

获取数据集对象中每列数据的数据类型。

返回:

list, 每列数据的数据类型列表。

样例:

```
>>> import mindspore.dataset as ds
>>> import numpy as np
>>>
>>> def generator1():
...     for i in range(1, 100):
...         yield np.ones((16, 83, 83)).astype(np.float32), np.array([i]).astype(np.int32)
>>>
>>> dataset = ds.GeneratorDataset(generator1, ["data1", "data2"])
>>> output_types = dataset.output_types()
>>> print(output_types)
[dtype('float32'), dtype('int32')]
```

应用采样方法

`mindspore.dataset.MappableDataset.add_sampler`

为当前数据集添加子采样器。

`mindspore.dataset.MappableDataset.use_sampler`

替换当前数据集的最末子采样器, 保持父采样器不变。

mindspore.dataset.MappableDataset.add_sampler

`mindspore.dataset.MappableDataset.add_sampler(new_sampler)`

为当前数据集添加子采样器。

说明：被添加的 `sampler` 如果有 `shuffle` 属性，其值必须是 `Shuffle.GLOBAL`，且原 `sampler` 的 `shuffle` 属性值不能是 `Shuffle.PARTIAL`。

参数：

- **new_sampler** (Sampler) - 待添加的子采样器。

样例：

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>>
>>> new_sampler = ds.DistributedSampler(10, 2)
>>> dataset.add_sampler(new_sampler)
```

mindspore.dataset.MappableDataset.use_sampler

`mindspore.dataset.MappableDataset.use_sampler(new_sampler)`

替换当前数据集的最末子采样器，保持父采样器不变。

参数：

- **new_sampler** (Sampler) - 用于替换的新采样器。

样例：

```
>>> import mindspore.dataset as ds
>>> dataset = ds.GeneratorDataset([i for i in range(10)], "column1")
>>>
>>> # use a DistributedSampler instead
>>> new_sampler = ds.DistributedSampler(10, 2)
>>> dataset.use_sampler(new_sampler)
```

其他方法

`mindspore.dataset.Dataset.sync_update`

释放阻塞条件并使用给定数据触发回调函数。

`mindspore.dataset.Dataset.sync_wait`

为同步操作在数据集对象上添加阻塞条件。

`mindspore.dataset.Dataset.to_json`

将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

`mindspore.dataset.Dataset.sync_update`

`mindspore.dataset.Dataset.sync_update(condition_name, num_batch=None, data=None)`

释放阻塞条件并使用给定数据触发回调函数。

参数：

- **condition_name** (str) - 用于触发发送下一个数据行的条件名称。
- **num_batch** (Union[int, None]) - 释放的 batch (row) 数。当 `num_batch` 为 `None` 时，将默认为 `sync_wait` 操作指定的值。默认值：None。
- **data** (Any) - 用户自定义传递给回调函数的数据。默认值：None。

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>>
>>> def gen():
...     for i in range(100):
...         yield (np.array(i),)
>>>
>>> class Augment:
...     def __init__(self, loss):
...         self.loss = loss
...
...     def preprocess(self, input_):
...         return input_
...
...     def update(self, data):
...         self.loss = data["loss"]
>>>
>>> batch_size = 10
>>> dataset = ds.GeneratorDataset(gen, column_names=["input"])
>>> aug = Augment(0)
>>> dataset = dataset.sync_wait(condition_name='', num_batch=1)
>>> dataset = dataset.map(input_columns=["input"], operations=[aug.preprocess])
>>> dataset = dataset.batch(batch_size)
>>>
>>> count = 0
>>> for data in dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     count += 1
...     data = {"loss": count}
...     dataset.sync_update(condition_name="", data=data)
```

`mindspore.dataset.Dataset.sync_wait`

`mindspore.dataset.Dataset.sync_wait(condition_name, num_batch=1, callback=None)`

为同步操作在数据集对象上添加阻塞条件。

参数：

- **condition_name** (str) - 用于触发发送下一行数据的条件名称。
- **num_batch** (int, 可选) - 每个 epoch 开始时无阻塞的 batch 数。默认值：1。
- **callback** (function, 可选) - `sync_update` 操作中将调用的回调函数。默认值：None。

返回：

Dataset，应用了上述操作的新数据集对象。

异常：

- **RuntimeError** - 条件名称已存在。

样例：

```
>>> import mindspore.dataset as ds
>>> import numpy as np
>>> def gen():
...     for i in range(100):
...         yield (np.array(i),)
>>>
>>> class Augment:
...     def __init__(self, loss):
...         self.loss = loss
...
...     def preprocess(self, input_):
...         return input_
...
...     def update(self, data):
...         self.loss = data["loss"]
>>>
>>> batch_size = 4
>>> dataset = ds.GeneratorDataset(gen, column_names=["input"])
>>>
>>> aug = Augment(0)
>>> dataset = dataset.sync_wait(condition_name="policy", callback=aug.update)
>>> dataset = dataset.map(operations=[aug.preprocess], input_columns=["input"])
>>> dataset = dataset.batch(batch_size)
>>> count = 0
>>> for data in dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     assert data["input"][0] == count
...     count += batch_size
...     data = {"loss": count}
...     dataset.sync_update(condition_name="policy", data=data)
```

`mindspore.dataset.Dataset.to_json`

`mindspore.dataset.Dataset.to_json(filename=)`

将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

参数：

- **filename** (str) - 保存 JSON 文件的路径（包含文件名）。默认值：""。

返回：

str，数据处理管道序列化后的 JSON 字符串。

样例：

```
>>> import mindspore.dataset as ds
>>> mnist_dataset_dir = "/path/to/mnist_dataset_directory"
>>> dataset = ds.MnistDataset(dataset_dir=mnist_dataset_dir)
>>> dataset_json = dataset.to_json("/path/to/mnist_dataset_pipeline.json")
```

12.4 标准格式数据集加载

<code>mindspore.dataset.MindDataset</code>	读取和解析 MindRecord 数据文件构建数据集。
<code>mindspore.dataset.OBSMindDataset</code>	读取和解析存放在华为云 OBS、Minio 以及 AWS S3 等云存储上的 MindRecord 格式数据集。
<code>mindspore.dataset.TFRecordDataset</code>	读取和解析 TFData 格式的数据文件构建数据集。

12.4.1 mindspore.dataset.MindDataset

class mindspore.dataset.**MindDataset** (*dataset_files, columns_list=None, num_parallel_workers=None, shuffle=None, num_shards=None, shard_id=None, sampler=None, padded_sample=None, num_padded=None, num_samples=None, cache=None*)

读取和解析 **MindRecord** 数据文件构建数据集。生成的数据集的列名和列类型取决于 **MindRecord** 文件中保存的列名与类型。

参数：

- **dataset_files** (Union[str, list[str]]) - **MindRecord** 文件路径，支持单文件路径字符串、多文件路径字符串列表。如果 *dataset_files* 的类型是字符串，则它代表一组具有相同前缀名的 **MindRecord** 文件，同一路径下具有相同前缀名的其他 **MindRecord** 文件将会被自动寻找并加载。如果 *dataset_files* 的类型是列表，则它表示所需读取的 **MindRecord** 数据文件。
- **columns_list** (list[str], 可选) - 指定从 **MindRecord** 文件中读取的数据列。默认值：None，读取所有列。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：None，采用 `mindspore.dataset.Shuffle.ADAPTIVE`。如果 *shuffle* 为 False，则不混洗，如果 *shuffle* 为 True，等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.ADAPTIVE`。通过传入枚举变量设置数据混洗的模式，枚举变量参考链接 [mindspore.dataset.Shuffle](#)：
 - `Shuffle.ADAPTIVE`：当数据集样本小于等于 1 亿时，采用 `Shuffle.GLOBAL`，当大于 1 亿时，采用局部 `Shuffle.PARTIAL`，每 100 万样本混洗一次。
 - `Shuffle.GLOBAL`：执行全局混洗，一次性混洗数据集中所有样本。占用内存大。
 - `Shuffle.PARTIAL`：执行局部混洗，每 100 万个样本混洗一次。占用内存小于 `Shuffle.GLOBAL`。
 - `Shuffle.FILES`：仅混洗文件序列，不混洗文件中的数据。
 - `Shuffle.INFILE`：保持读入文件的序列，仅混洗每个文件中的数据。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 数据并行模式训练 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。当前此数据集仅支持以下采样器：`mindspore.dataset.SubsetRandomSampler`、`mindspore.dataset.PKSampler`、`mindspore.dataset.RandomSampler`、`mindspore.dataset.SequentialSampler` 和 `mindspore.dataset.DistributedSampler`。
- **padded_sample** (dict, 可选) - 指定额外添加到数据集的样本，可用于在分布式训练时补齐分片数据，注意字典的键名需要与 *columns_list* 指定的列名相同。默认值：None，不添加样本。需要与 *num_padded* 参数同时使用。

- **num_padded** (int, 可选) - 指定额外添加的数据集样本的数量。在分布式训练时可用于为数据集补齐样本，使得总样本数量可被 *num_shards* 整除。默认值：None，不添加样本。需要与 *padded_sample* 参数同时使用。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **ValueError** - *dataset_files* 参数所指向的文件无效或不存在。
- **ValueError** - *num_parallel_workers* 参数超过最大线程数。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, num\_shards)$ 范围。
- **TypeError** - *shuffle* 的类型不是 None 或者 bool 或者 Shuffle。

样例：

```
>>> import mindspore.dataset as ds
>>> mindrecord_files = ["/path/to/mind_dataset_file"] # contains 1 or multiple MindRecord_
    ↳ files
>>> dataset = ds.MindDataset(dataset_files=mindrecord_files)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：对 MindRecord 进行分片（配置 *num_shards* 和 *shard_id*）时，数据的切分逻辑有 2 种实现策略，此 API 默认采用策略 1，可通过设置环境变量 *MS_DEV_MINDRECORD_SHARD_BY_BLOCK=True* 切换回策略 2。该环境变量只对 *DistributedSampler* 采样器生效。

表 11: 数据分片的实现策略 1

rank 0	rank 1	rank 2	rank 3
0	1	2	3
4	5	6	7
8	9	10	11

表 12: 数据分片的实现策略 2

rank 0	rank 1	rank 2	rank 3
0	3	6	9
1	4	7	10
2	5	8	11

说明：入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器，其与入参 *sampler* 搭配使用的效果如下。

表 13: 参数 `sampler` 和 `num_samples`, `shuffle`, `num_shards`, `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mindspore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>DatasetSampler(indices = sampler , num_samples = num_samples)</code>
iterable 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>DatasetSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理, 并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表, 按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列, 并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <code>count</code> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <code>epoch</code> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <code>buffer_size</code> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <code>count</code> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <code>dataset</code> 对象按列进行合并压缩, 多个 <code>dataset</code> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器, 保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

12.4.2 mindspore.dataset.OBSMindDataset

class mindspore.dataset.OBSMindDataset (dataset_files, server, ak, sk, sync_obs_path, columns_list=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, shard_equal_rows=True)

读取和解析存放在华为云 OBS、Minio 以及 AWS S3 等云存储上的 MindRecord 格式数据集。生成的数据集的列名和列类型取决于 MindRecord 文件中的保存的列名与类型。

说明:

- 该接口会访问 /cache 目录做节点同步，需要用户确保有 /cache 目录的访问权限。

参数:

- **dataset_files** (list[str]) - 云存储上 MindRecord 格式数据集文件的路径列表，每个文件的路径格式为 s3://bucketName/objectKey。
- **server** (str) - 连接云存储服务地址。可包含协议类型、域名、端口号。假如为华为云 OBS，服务地址为: <obs.cn-north-4.myhuaweicloud.com>。假如为 Minio，服务地址为: <https://your-endpoint:9000>。
- **ak** (str) - 用于访问 OBS 数据的访问密钥 ID。
- **sk** (str) - 用于访问 OBS 数据的私有访问密钥。
- **sync_obs_path** (str) - 用于同步操作云存储上的路径，用户需要提前创建，目录路径的格式为 s3://bucketName/objectKey。
- **columns_list** (list[str], 可选) - 指定从 MindRecord 文件中读取的数据列。默认值: None，读取所有列。
- **shuffle** (Union[bool, Shuffle], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值: Shuffle.GLOBAL。如果 shuffle 为 False，则不混洗，如果 shuffle 为 True，等同于将 shuffle 设置为 mindspore.dataset.Shuffle.GLOBAL。通过传入枚举变量设置数据混洗的模式：
 - Shuffle.GLOBAL：混洗文件和文件中的数据。
 - Shuffle.FILES：仅混洗文件。
 - Shuffle.INFILE：保持读入文件的序列，仅混洗每个文件中的数据。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。一般在 数据并行模式训练 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 num_shards 时才能指定此参数。
- **shard_equal_rows** (bool, 可选) - 分布式训练时，为所有分片获取等量的数据行数。默认值: True。如果 shard_equal_rows 为 False，则可能会使得每个分片的数据条目不相等，从而导致分布式训练失败。因此当每

个 MindRecord 文件的数据数量不相等时, 建议将此参数设置为 True。注意, 只有当指定了 `num_shards` 时才能指定此参数。

异常:

- **RuntimeError** - `sync_obs_path` 参数指定的目录不存在。
- **ValueError** - `columns_list` 参数无效。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> # OBS
>>> bucket = "iris" # your obs bucket name
>>> # the bucket directory structure is similar to the following:
>>> # - imagenet21k
>>> # | - mr_imagenet21k_01
>>> # | - mr_imagenet21k_02
>>> # - sync_node
>>> dataset_obs_dir = ["s3://" + bucket + "/imagenet21k/mr_imagenet21k_01",
...                    "s3://" + bucket + "/imagenet21k/mr_imagenet21k_02"]
>>> sync_obs_dir = "s3://" + bucket + "/sync_node"
>>> num_shards = 8
>>> shard_id = 0
>>> dataset = ds.OBSMindDataset(dataset_obs_dir, "obs.cn-north-4.myhuaweicloud.com",
...                               "AK of OBS", "SK of OBS",
...                               sync_obs_dir, shuffle=True, num_shards=num_shards, shard_
↪ id=shard_id)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明:

- 需要用户提前在云存储上创建同步用的目录, 然后通过 `sync_obs_path` 指定。
 - 如果线下训练, 建议为每次训练设置 `BATCH_JOB_ID` 环境变量。
 - 分布式训练中, 假如使用多个节点 (服务器), 则必须使用每个节点全部的 8 张卡。如果只有一个节点 (服务器), 则没有这样的限制。
-

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

12.4.3 mindspore.dataset.TFRecordDataset

class mindspore.dataset.**TFRecordDataset** (*dataset_files, schema=None, columns_list=None, num_samples=None, num_parallel_workers=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, shard_equal_rows=False, cache=None, compression_type=None*)

读取和解析 TFData 格式的数据文件构建数据集。生成的数据集的列名和列类型取决于 TFRecord 文件中的保存的列名与类型。

说明：Windows 平台尚不支持 *TFRecordDataset* 。

参数：

- **dataset_files** (Union[str, list[str]]) - 数据集文件路径, 支持单文件路径字符串、多文件路径字符串列表或可被 glob 库模式匹配的字符串, 文件列表将在内部进行字典排序。
- **schema** (Union[str, *Schema*], 可选) - 数据格式策略, 用于指定读取数据列的数据类型、数据维度等信息。支持传入 JSON 文件路径或 `mindspore.dataset.Schema` 构造的对象。默认值: None。
- **columns_list** (list[str], 可选) - 指定从 TFRecord 文件中读取的数据列。默认值: None, 读取所有列。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。当指定了 `num_shards` 和 `shard_id` 参数时, `num_samples` 或 `numRows` 字段 (由参数 `schema` 定义) 将表示每个分片读取的数据量。`num_samples` 的处理优先级如下:
 - 指定了 `num_samples` 参数, 且值大于 0, 则读取 `num_samples` 条数据。此时 `schema` 参数的 `numRows` 字段会失效。
 - 不指定 `num_samples` 参数, 指定了 `schema` 参数并定义了 `numRows` 字段, 且值大于 0, 则读取 `numRows` 条数据。
 - 不指定 `num_samples` 参数与 `schema` 参数, 读取所有样本数据。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式, 支持传入 bool 类型与枚举类型进行指定。默认值: `Shuffle.GLOBAL`。如果 `shuffle` 为 False, 则不混洗, 如果 `shuffle` 为 True, 等同于将 `shuffle` 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式:
 - `Shuffle.GLOBAL`: 混洗文件和样本。
 - `Shuffle.FILES`: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **shard_equal_rows** (bool, 可选) - 分布式训练时, 为所有分片获取等量的数据行数。默认值: False。如果 `shard_equal_rows` 为 False, 则可能会使得每个分片的数据条目不相等, 从而导致分布式训练失败。因此当每个 TFRecord 文件的数据数量不相等时, 建议将此参数设置为 True。注意, 只有当指定了 `num_shards` 时才能指定此参数。当 `compression_type` 非 None, 且指定了 `num_samples` 或 `numRows` 字段 (由参数 `schema` 定义) 时, `shard_equal_rows` 会被视为 True。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。
- **compression_type** (str, 可选) - 用于所有文件的压缩类型, 必须是 "", "GZIP" 或 "ZLIB"。默认值: None, 即空字符串。建议在 `compression_type` 为 "GZIP" 或 "ZLIB" 时, 指定 `num_samples` 或 `numRows` 字段 (由参数 `schema` 定义) 以避免出现为了获取文件大小对同一个文件进行多次解压而导致性能下降的问题。

异常:

- **ValueError** - `dataset_files` 参数所指向的文件无效或不存在。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **ValueError** - `compression_type` 不是 '', 'GZIP', 'ZLIB' 三者之一。
- **ValueError** - `compression_type` 有效但是数据集文件数量小于 `num_shards`。

- **ValueError** - `num_samples` 小于 0。

样例:

```
>>> import mindspore.dataset as ds
>>> from mindspore import dtype as mstype
>>>
>>> tfrecord_dataset_dir = ["/path/to/tfrecord_dataset_file"] # contains 1 or multiple
↳TFRecord files
>>> tfrecord_schema_file = "/path/to/tfrecord_schema_file"
>>>
>>> # 1) Get all rows from tfrecord_dataset_dir with no explicit schema.
>>> # The meta-data in the first row will be used as a schema.
>>> dataset = ds.TFRecordDataset(dataset_files=tfrecord_dataset_dir)
>>>
>>> # 2) Get all rows from tfrecord_dataset_dir with user-defined schema.
>>> schema = ds.Schema()
>>> schema.add_column(name='col_1d', de_type=mstype.int64, shape=[2])
>>> dataset = ds.TFRecordDataset(dataset_files=tfrecord_dataset_dir, schema=schema)
>>>
>>> # 3) Get all rows from tfrecord_dataset_dir with the schema file.
>>> dataset = ds.TFRecordDataset(dataset_files=tfrecord_dataset_dir, schema=tfrecord_schema_
↳file)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <code>count</code> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <code>epoch</code> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <code>buffer_size</code> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <code>count</code> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <code>dataset</code> 对象按列进行合并压缩，多个 <code>dataset</code> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

12.5 开源数据集加载

12.5.1 视觉数据集

<code>mindspore.dataset.Caltech101Dataset</code>	Caltech 101 数据集。
<code>mindspore.dataset.Caltech256Dataset</code>	Caltech 256 数据集。
<code>mindspore.dataset.CelebADataset</code>	CelebA (CelebFaces Attributes) 数据集。
<code>mindspore.dataset.Cifar10Dataset</code>	CIFAR-10 数据集。
<code>mindspore.dataset.Cifar100Dataset</code>	CIFAR-100 数据集。
<code>mindspore.dataset.CityscapesDataset</code>	Cityscapes 数据集。
<code>mindspore.dataset.CocoDataset</code>	COCO (Common Objects in Context) 数据集。
<code>mindspore.dataset.DIV2KDataset</code>	DIV2K (DIVERse 2K resolution image) 数据集。
<code>mindspore.dataset.EMnistDataset</code>	EMNIST (Extended MNIST) 数据集。
<code>mindspore.dataset.FakeImageDataset</code>	生成虚假图像构建数据集。
<code>mindspore.dataset.FashionMnistDataset</code>	Fashion-MNIST 数据集。
<code>mindspore.dataset.FlickrDataset</code>	Flickr8k 和 Flickr30k 数据集。
<code>mindspore.dataset.Flowers102Dataset</code>	Oxford 102 Flower 数据集。
<code>mindspore.dataset.Food101Dataset</code>	Food101 数据集。
<code>mindspore.dataset.ImageFolderDataset</code>	从树状结构的文件目录中读取图片构建源数据集。
<code>mindspore.dataset.KITTI Dataset</code>	KITTI 数据集。
<code>mindspore.dataset.KMNISTDataset</code>	KMNIST (Kuzushiji-MNIST) 数据集。
<code>mindspore.dataset.LFWDataset</code>	LFW (Labeled Faces in the Wild) 数据集。
<code>mindspore.dataset.LSUNDataset</code>	LSUN (Large-scale Scene UNDERstanding) 数据集。
<code>mindspore.dataset.ManifestDataset</code>	读取和解析 Manifest 数据文件构建数据集。
<code>mindspore.dataset.MnistDataset</code>	MNIST 数据集。
<code>mindspore.dataset.OmniglotDataset</code>	Omniglot 数据集。
<code>mindspore.dataset.PhotoTourDataset</code>	PhotoTour 数据集。
<code>mindspore.dataset.Places365Dataset</code>	Places365 数据集。
<code>mindspore.dataset.QMNISTDataset</code>	QMNIST 数据集。
<code>mindspore.dataset.RenderedSST2Dataset</code>	RenderedSST2 (Rendered Stanford Sentiment Treebank v2) 数据集。
<code>mindspore.dataset.SBDataset</code>	SB (Semantic Boundaries) 数据集。
<code>mindspore.dataset.SBUDataset</code>	SBU (SBU Captioned Photo) 数据集。
<code>mindspore.dataset.SemeionDataset</code>	Semeion 数据集。
<code>mindspore.dataset.STL10Dataset</code>	STL-10 数据集。
<code>mindspore.dataset.SUN397Dataset</code>	SUN397 (Scene UNDERstanding) 数据集。
<code>mindspore.dataset.SVHNDataset</code>	SVHN (Street View House Numbers) 数据集。
<code>mindspore.dataset.USPSDataset</code>	USPS (U.S. Postal Service) 数据集。

下页继续

表 32 – 续上页

<code>mindspore.dataset.VOCDataset</code>	VOC (Visual Object Classes) 数据集。
<code>mindspore.dataset.WIDERFaceDataset</code>	WIDERFace 数据集。

mindspore.dataset.Caltech101Dataset

```
class mindspore.dataset.Caltech101Dataset (dataset_dir, target_type=None, num_samples=None,
                                           num_parallel_workers=1, shuffle=None, decode=False,
                                           sampler=None, num_shards=None, shard_id=None)
```

Caltech 101 数据集。

根据不同的 `target_type` 配置，数据集会生成不同的输出列。

- `target_type` 为 'category'，输出列为 `[image, category]`。
- `target_type` 为 'annotation'，输出列为 `[image, annotation]`。
- `target_type` 为 'all'，输出列为 `[image, category, annotation]`。

列 `image` 为 `uint8` 类型。列 `category` 为 `uint32` 类型。列 `annotation` 是一个二维的 `ndarray`，存储了图像的轮廓，由一系列的点组成。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径，该路径下将包含 2 个子目录，目录 `101_ObjectCategories` 用于存储图像，目录 `Annotations` 用于存储图像的标注。
- **target_type** (str, 可选) - 指定数据集的子集，可取值为 'category'、'annotation' 或 'all'。取值为 'category' 时将读取图像的类别标注作为 `label`，取值为 'annotation' 时将读取图像的轮廓标注作为 `label`，取值为 'all' 时将同时输出图像的类别标注和轮廓标注。默认值：None，表示 'category'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作进程数。默认值：1。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `shard_id` 参数错误，小于 0 或者大于等于 `num_shards`。
- **ValueError** - `target_type` 参数取值不为 'category'、'annotation' 或 'all'。

- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> caltech101_dataset_directory = "/path/to/caltech101_dataset_directory"
>>>
>>> # 1) Read all samples (image files) in caltech101_dataset_directory with 8 threads
>>> dataset = ds.Caltech101Dataset(dataset_dir=caltech101_dataset_directory, num_parallel_
↳workers=8)
>>>
>>> # 2) Read all samples (image files) with the target_type "annotation"
>>> dataset = ds.Caltech101Dataset(dataset_dir=caltech101_dataset_directory, target_type=
↳"annotation")
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

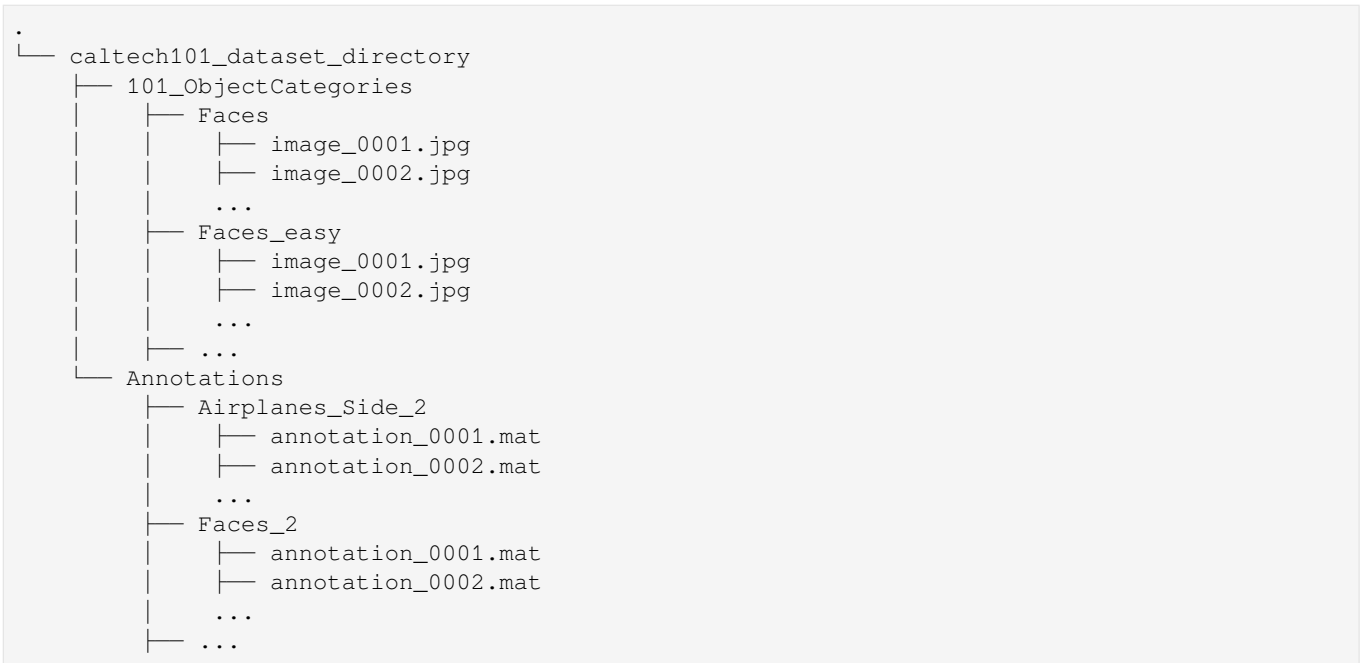
表 33: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuf- fle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray</i> , <i>list</i> , <i>tuple</i> , <i>int</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	DistributedSampler (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IterativeSampler (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **Caltech101** 数据集:

Caltech101 数据集包含 101 种类别的图片。每种类别大约 40 到 800 张图像, 大多数类别有大约 50 张图像。每张图像的大小约为 300 x 200 像素。数据集中也提供了每张图片中每个物体的轮廓数据, 用于检测和定位。

您可以解压缩原始 Caltech101 数据集文件到如下目录结构, 并通过 MindSpore 的 API 进行读取。



引用:

```
@article{FeiFei2004LearningGV,
author   = {Li Fei-Fei and Rob Fergus and Pietro Perona},
title    = {Learning Generative Visual Models from Few Training Examples:
            An Incremental Bayesian Approach Tested on 101 Object Categories},
journal  = {Computer Vision and Pattern Recognition Workshop},
year     = {2004},
url      = {http://data.caltech.edu/records/20086},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。

下页继续

表 34 – 续上页

<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.Caltech256Dataset

```
class mindspore.dataset.Caltech256Dataset (dataset_dir, num_samples=None, num_parallel_workers=None,
                                           shuffle=None, decode=False, sampler=None, num_shards=None,
                                           shard_id=None, cache=None)
```

Caltech 256 数据集。

生成的数据集有两列 `[image, label]`。`image` 列的数据类型为 `uint8`。`label` 列的数据类型为 `uint32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过`mindspore.dataset.config.set_num_parallel_workers()`配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples`表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。

- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, num\_shards)$ 范围。
- **ValueError** - *target_type* 参数取值不为 'category'、'annotation' 或 'all'。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例：

```
>>> import mindspore.dataset as ds
>>> caltech256_dataset_dir = "/path/to/caltech256_dataset_directory"
>>>
>>> # 1) Read all samples (image files) in caltech256_dataset_dir with 8 threads
>>> dataset = ds.Caltech256Dataset(dataset_dir=caltech256_dataset_dir, num_parallel_
↪workers=8)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器，其与入参 *sampler* 搭配使用的效果如下。

表 40: 参数 *sampler* 和 *num_samples*，*shuffle*，*num_shards*，*shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mindspore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray</i> , <i>list</i> , <i>tuple</i> , <i>int</i> 类型	<i>None</i>	<i>/</i>	<i>num_samples</i>	<i>IndexSampler</i> (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	<i>IndexSampler</i> (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	<i>DistributedSampler</i> (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	<i>DistributedSampler</i> (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	<i>RandomSampler</i> (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	<i>RandomSampler</i> (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	<i>SequentialSampler</i> (<i>num_samples</i> = num_samples)

关于 **Caltech256** 数据集：

Caltech-256 是一个对象识别数据集, 包含 30,607 张不同大小的真实世界图像, 共有 257 个类别 (256 类物体和 1 个其他类), 每个类别由至少 80 张图像。该数据集是 Caltech101 数据集的超集。

您可以解压缩原始 Caltech256 数据集文件到如下目录结构, 并通过 MindSpore 的 API 进行读取。

```
.
├── caltech256_dataset_directory
│   ├── 001.ak47
│   │   ├── 001_0001.jpg
│   │   ├── 001_0002.jpg
│   │   └── ...
│   ├── 002.american-flag
│   │   ├── 002_0001.jpg
│   │   ├── 002_0002.jpg
│   │   └── ...
│   ├── 003.backpack
│   │   ├── 003_0001.jpg
│   │   ├── 003_0002.jpg
│   │   └── ...
│   └── ...
```

引用:

```
@article{griffin2007caltech,
title      = {Caltech-256 object category dataset},
added-at   = {2021-01-21T02:54:42.000+0100},
author     = {Griffin, Gregory and Holub, Alex and Perona, Pietro},
biburl     = {https://www.bibsonomy.org/bibtex/21f746f23ff0307826cca3e3be45f8de7/s364315},
interhash  = {bfe1e648c1778c04baa60f23d1223375},
intrahash  = {1f746f23ff0307826cca3e3be45f8de7},
publisher  = {California Institute of Technology},
timestamp  = {2021-01-21T02:54:42.000+0100},
year       = {2007}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理, 并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表, 按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列, 并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 epoch 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。

下页继续

表 41 – 续上页

<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.CelebADataset

`class mindspore.dataset.CelebADataset (dataset_dir, num_parallel_workers=None, shuffle=None, usage='all', sampler=None, decode=False, extensions=None, num_samples=None, num_shards=None, shard_id=None, cache=None, decrypt=None)`

CelebA (CelebFaces Attributes) 数据集。

目前仅支持解析 CelebA 数据集的 `list_attr_celeba.txt` 文件作为数据集的 label。生成的数据集有两列 `[image, attr]`。`image` 列的数据类型为 `uint8`。`attr` 列的数据类型为 `uint32`，并以 `one-hot` 编码的形式生成。

参数：

- dataset_dir** (str) - 包含数据集文件的根目录路径。
- num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'valid'、'test' 或 'all'。默认值：'all'，全部样本图片。
- sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- extensions** (list[str], 可选) - 指定文件的扩展名，仅读取与指定扩展名匹配的文件到数据集中。默认值：None。
- num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。

- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: `None`, 不使用缓存。
- **decrypt** (callable, 可选) - 图像解密函数, 接收加密的图片路径并返回 `bytes` 类型的解密数据。默认值: `None`, 不进行解密。

异常:

- **RuntimeError** - *dataset_dir* 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 `[0, num_shards)` 范围。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - *usage* 参数取值不为 `'train'`、`'valid'`、`'test'` 或 `'all'`。

样例:

```
>>> import mindspore.dataset as ds
>>> celeba_dataset_dir = "/path/to/celeba_dataset_directory"
>>>
>>> # Read 5 samples from CelebA dataset
>>> dataset = ds.CelebADataset(dataset_dir=celeba_dataset_dir, usage='train', num_samples=5)
>>>
>>> # Note: In celeba dataset, each data dictionary owns keys "image" and "attr"
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

说明: 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器, 其与入参 *sampler* 搭配使用的效果如下。

表 47: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
iterable 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **CelebA** 数据集:

CelebFaces Attributes Dataset (CelebA) 数据集是一个大规模数据集, 拥有超过 20 万张名人图像, 每个图像都有 40 个属性标注。此数据集包含了大量不同姿态、各种背景的图像, 种类丰富、数量庞大、标注充分。数据集总体包含:

- 10177 个不同的身份
- 202599 张图像
- 每张图像拥有 5 个五官位置标注, 40 个属性标签

此数据集可用于各种计算机视觉任务的训练和测试, 包括属性识别、检测和五官定位等。

原始 CelebA 数据集结构:

```
.
├── CelebA
│   ├── README.md
│   ├── Img
│   │   ├── img_celeba.7z
│   │   ├── img_align_celeba_png.7z
│   │   └── img_align_celeba.zip
│   ├── Eval
│   │   └── list_eval_partition.txt
│   └── Anno
│       ├── list_landmarks_celeba.txt
│       ├── list_landmarks_align_celeba.txt
│       ├── list_bbox_celeba.txt
│       ├── list_attr_celeba.txt
│       └── identity_CelebA.txt
```

您可以将上述 Anno 目录下的 txt 文件与 Img 目录下的文件解压放至同一目录，并通过 MindSpore 的 API 进行读取。

```
.
├── celeba_dataset_directory
│   ├── list_attr_celeba.txt
│   ├── 000001.jpg
│   ├── 000002.jpg
│   ├── 000003.jpg
│   └── ...
```

引用:

```
@article{DBLP:journals/corr/LiuLWT14,
author    = {Ziwei Liu and Ping Luo and Xiaogang Wang and Xiaoou Tang},
title     = {Deep Learning Attributes in the Wild},
journal   = {CoRR},
volume    = {abs/1411.7766},
year      = {2014},
url       = {http://arxiv.org/abs/1411.7766},
archivePrefix = {arXiv},
eprint    = {1411.7766},
timestamp = {Tue, 10 Dec 2019 15:37:26 +0100},
biburl    = {https://dblp.org/rec/journals/corr/LiuLWT14.bib},
bibsource = {dblp computer science bibliography, https://dblp.org},
howpublished = {http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.Cifar10Dataset

class mindspore.dataset.Cifar10Dataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None)

CIFAR-10 数据集。

该 API 目前仅支持解析二进制版本的 CIFAR-10 文件（CIFAR-10 binary version）。生成的数据集有两列: [image, label]。image 列的数据类型是 uint8。label 列的数据类型是 uint32。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 50,000 个训练样本，取值为 'test' 时将会读取 10,000 个测试样本，取值为 'all' 时将会读取全部 60,000 个样本。默认值：None，读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

样例：

```
>>> import mindspore.dataset as ds
>>> cifar10_dataset_dir = "/path/to/cifar10_dataset_directory"
>>>
>>> # 1) Get all samples from CIFAR10 dataset in sequence
>>> dataset = ds.Cifar10Dataset(dataset_dir=cifar10_dataset_dir, shuffle=False)
>>>
>>> # 2) Randomly select 350 samples from CIFAR10 dataset
>>> dataset = ds.Cifar10Dataset(dataset_dir=cifar10_dataset_dir, num_samples=350,
    ↪ shuffle=True)
>>>
```

(下页继续)

(续上页)

```
>>> # 3) Get samples from CIFAR10 dataset for shard 0 in a 2-way distributed training
>>> dataset = ds.Cifar10Dataset(dataset_dir=cifar10_dataset_dir, num_shards=2, shard_id=0)
>>>
>>> # In CIFAR10 dataset, each dictionary has keys "image" and "label"
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - `usage` 参数取值不为 'train'、'test' 或 'all'。

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 54: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **CIFAR-10** 数据集：

CIFAR-10 数据集由 60000 张 32x32 彩色图片组成，总共有 10 个类别，每类 6000 张图片。有 50000 个训练样本和 10000 个测试样本。10 个类别包含飞机、汽车、鸟类、猫、鹿、狗、青蛙、马、船和卡车。

以下为原始 CIFAR-10 数据集的结构。您可以将数据集文件解压得到如下的文件结构，并通过 MindSpore 的 API 进行读取。

```
.
├── cifar-10-batches-bin
│   ├── data_batch_1.bin
│   ├── data_batch_2.bin
│   ├── data_batch_3.bin
│   ├── data_batch_4.bin
│   ├── data_batch_5.bin
│   ├── test_batch.bin
│   ├── readme.html
│   └── batches.meta.text
```

引用：

```
@techreport{Krizhevsky09,
author      = {Alex Krizhevsky},
title       = {Learning multiple layers of features from tiny images},
institution = {},
year        = {2009},
howpublished = {http://www.cs.toronto.edu/~kriz/cifar.html}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.Cifar100Dataset

`class mindspore.dataset.Cifar100Dataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None)`

CIFAR-100 数据集。

生成的数据集有三列: `[image, coarse_label, fine_label]`。 `image` 列的数据类型为 `uint8`。 `coarse_label` 和 `fine_labels` 列的数据类型为 `uint32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 50,000 个训练样本，取值为 'test' 时将会读取 10,000 个测试样本，取值为 'all' 时将会读取全部 60,000 个样本。默认值：None，读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

样例：

```
>>> import mindspore.dataset as ds
>>> cifar100_dataset_dir = "/path/to/cifar100_dataset_directory"
>>>
>>> # 1) Get all samples from CIFAR100 dataset in sequence
>>> dataset = ds.Cifar100Dataset(dataset_dir=cifar100_dataset_dir, shuffle=False)
>>>
>>> # 2) Randomly select 350 samples from CIFAR100 dataset
>>> dataset = ds.Cifar100Dataset(dataset_dir=cifar100_dataset_dir, num_samples=350,
↪ shuffle=True)
```

(下页继续)

(续上页)

```
>>>
>>> # In CIFAR100 dataset, each dictionary has 3 keys: "image", "fine_label" and "coarse_
↳label"
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - `usage` 参数取值不为 'train'、'test' 或 'all' 。

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 61: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **CIFAR-100** 数据集：

CIFAR-100 数据集和 CIFAR-10 数据集非常相似，CIFAR-100 有 100 个类别，每类包含 600 张图片。其中 500 张训练图片和 100 张测试图片。这 100 个类别又被分成 20 个超类。每个图片都有一个“fine” 标签（所属子类）和一个“coarse” 标签（所属超类）。

以下为原始 CIFAR-100 数据集的结构。您可以将数据集文件解压得到如下的文件结构，并通过 MindSpore 的 API 进行读取。

```
.
└─ cifar-100-binary
    ├── train.bin
    ├── test.bin
    ├── fine_label_names.txt
    └─ coarse_label_names.txt
```

引用：

```
@techreport{Krizhevsky09,
author      = {Alex Krizhevsky},
title       = {Learning multiple layers of features from tiny images},
institution = {},
year        = {2009},
howpublished = {http://www.cs.toronto.edu/~kriz/cifar.html}
}
```

预处理操作

mindspore.dataset.Dataset.apply	对数据集对象执行给定操作函数。
mindspore.dataset.Dataset.concat	对传入的多个数据集对象进行拼接操作。
mindspore.dataset.Dataset.filter	通过自定义判断条件对数据集对象中的数据进行过滤。
mindspore.dataset.Dataset.flat_map	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
mindspore.dataset.Dataset.map	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
mindspore.dataset.Dataset.project	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
mindspore.dataset.Dataset.rename	对数据集对象按指定的列名进行重命名。
mindspore.dataset.Dataset.repeat	重复此数据集 <i>count</i> 次。
mindspore.dataset.Dataset.reset	重置下一个 <i>epoch</i> 的数据集对象。
mindspore.dataset.Dataset.save	将数据处理管道中正处理的数据保存为通用的数据集格式。
mindspore.dataset.Dataset.shuffle	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
mindspore.dataset.Dataset.skip	跳过此数据集对象的前 <i>count</i> 条数据。
mindspore.dataset.Dataset.split	将数据集拆分为多个不重叠的子数据集。
mindspore.dataset.Dataset.take	截取数据集的前指定条数据。
mindspore.dataset.Dataset.zip	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.CityscapesDataset

```
class mindspore.dataset.CityscapesDataset (dataset_dir, usage='train', quality_mode='fine', task='instance',
                                             num_samples=None, num_parallel_workers=None, shuffle=None,
                                             decode=None, sampler=None, num_shards=None, shard_id=None,
                                             cache=None)
```

Cityscapes 数据集。

生成的数据集有两列 `[image, task]`。`image` 列的数据类型为 `uint8`。`task` 列的数据类型根据参数 `task` 的值而定，当参数 `task` 取值为 `'polygon'`，列的数据类型为 `string`；其他取值下，列的数据类型为 `uint8`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集。当参数 `quality_mode` 取值为 `'fine'` 时，此参数可取值为 `'train'`、`'test'`、`'val'` 或 `'all'`。当参数 `quality_mode` 取值为 `'coarse'` 时，此参数可取值为 `'train'`、`'train_extra'`、`'val'` 或 `'all'`。默认值：`'train'`，全部样本图片。
- **quality_mode** (str, 可选) - 指定数据集的质量模式，可取值为 `'fine'` 或 `'coarse'`。默认值：`'fine'`。
- **task** (str, 可选) - 指定数据集的任务类型，可取值为 `'instance'`、`'semantic'`、`'polygon'` 或 `'color'`。默认值：`'instance'`。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：`None`，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：`None`，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：`None`。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：`None`，表示 `False`，不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：`None`。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时，数据集划分的片数。默认值：`None`。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时，使用的分片 ID 号。默认值：`None`。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：`None`，不使用缓存。

样例：

```
>>> import mindspore.dataset as ds
>>> cityscapes_dataset_dir = "/path/to/cityscapes_dataset_directory"
>>>
```

(下页继续)

(续上页)

```

>>> # 1) Get all samples from Cityscapes dataset in sequence
>>> dataset = ds.CityscapesDataset(dataset_dir=cityscapes_dataset_dir, task="instance",
    ↪quality_mode="fine",
    ...                               usage="train", shuffle=False, num_parallel_workers=1)
>>>
>>> # 2) Randomly select 350 samples from Cityscapes dataset
>>> dataset = ds.CityscapesDataset(dataset_dir=cityscapes_dataset_dir, num_samples=350,
    ↪shuffle=True,
    ...                               num_parallel_workers=1)
>>>
>>> # 3) Get samples from Cityscapes dataset for shard 0 in a 2-way distributed training
>>> dataset = ds.CityscapesDataset(dataset_dir=cityscapes_dataset_dir, num_shards=2, shard_
    ↪id=0,
    ...                               num_parallel_workers=1)
>>>
>>> # In Cityscapes dataset, each dictionary has keys "image" and "task"

```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - `dataset_dir` 路径非法或不存在。
- **ValueError** - `task` 参数取值不为 'instance'、'semantic'、'polygon' 或 'color'。
- **ValueError** - `quality_mode` 参数取值不为 'fine' 或 'coarse'。
- **ValueError** - `usage` 参数取值不在给定的字段中。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 68: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuf- fle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray,list,tuple,int</i> 类型	<i>int</i>	<i>/</i>	<i>num_samples</i>	<code>DatasetSampler(indices = sampler , num_samples = num_samples)</code>
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	<code>RandomSampler(num_samples = num_samples)</code>
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 Cityscapes 数据集：

Cityscapes 数据集由来自 50 个城市的 24998 张彩色图像组成。其中 5000 张图像具有高质量的密集像素标注，19998 张图像具有粗糙的多边形标注。该数据集共有 30 个类，多边形标注包括密集语义分割，以及车辆和人的实例分割。您可以解压缩原始数据集文件到如下目录结构，并通过 MindSpore 的 API 进行读取。

```
.
├── Cityscapes
│   ├── leftImg8bit
│   │   ├── train
│   │   │   ├── aachen
│   │   │   │   ├── aachen_000000_000019_leftImg8bit.png
│   │   │   │   ├── aachen_000001_000019_leftImg8bit.png
│   │   │   │   └── ...
│   │   │   └── bochum
│   │   │       └── ...
│   │   └── test
│   │       └── ...
│   └── val
│       └── ...
├── gtFine
│   ├── train
│   │   ├── aachen
│   │   │   ├── aachen_000000_000019_gtFine_color.png
│   │   │   ├── aachen_000000_000019_gtFine_instanceIds.png
│   │   │   ├── aachen_000000_000019_gtFine_labelIds.png
│   │   │   └── aachen_000000_000019_gtFine_polygons.json
```

(下页继续)

(续上页)

```
|      |      |— aachen_000001_000019_gtFine_color.png
|      |      |— aachen_000001_000019_gtFine_instanceIds.png
|      |      |— aachen_000001_000019_gtFine_labelIds.png
|      |      |— aachen_000001_000019_gtFine_polygons.json
|      |      |— ...
|      |      |— bochum
|      |      |— ...
|      |      |— ...
|— test
|      |— ...
|— val
|      |— ...
```

引用:

```
@inproceedings{Cordts2016Cityscapes,
title      = {The Cityscapes Dataset for Semantic Urban Scene Understanding},
author     = {Cordts, Marius and Omran, Mohamed and Ramos, Sebastian and Rehfeld, Timo and
↪Enzweiler, Markus and Benenson, Rodrigo and Franke, Uwe and Roth, Stefan and Schiele,
↪Bernt},
booktitle  = {Proc. of the IEEE Conference on Computer Vision and Pattern Recognition
↪(CVPR)},
year       = {2016}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.CocoDataset

class mindspore.dataset.CocoDataset (*dataset_dir, annotation_file, task='Detection', num_samples=None, num_parallel_workers=None, shuffle=None, decode=False, sampler=None, num_shards=None, shard_id=None, cache=None, extra_metadata=False, decrypt=None*)

COCO (Common Objects in Context) 数据集。

该 API 支持解析 COCO2017 数据集, 支持四种类型的机器学习任务, 分别是目标检测、关键点检测、物体分割、全景分割和图片注解。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **annotation_file** (str) - 数据集标注 JSON 文件的路径。
- **task** (str, 可选) - 指定 COCO 数据的任务类型。支持的任务类型包括: 'Detection' (目标检测)、'Stuff' (物体分割)、'Panoptic' (全景分割)、'Keypoint' (关键点检测) 和 'Captioning' (图片注解)。默认值: 'Detection'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None, 表 2 中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: False, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None, 表 2 中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。
- **extra_metadata** (bool, 可选) - 用于指定是否额外输出一个数据列用于表示图片元信息。如果为 True, 则将额外输出一个名为 `[_meta-filename, dtype=string]` 的数据列。默认值: False。
- **decrypt** (callable, 可选) - 图像解密函数, 接受加密的图片路径并返回 bytes 类型的解密数据。默认值: None, 不进行解密。

根据不同 `task` 参数设置, 生成数据集具有不同的输出列:

<i>task</i>	输出列
Detection	[image, dtype=uint8] [bbox, dtype=float32] [category_id, dtype=uint32] [iscrowd, dtype=uint32]
Stuff	[image, dtype=uint8] [segmentation, dtype=float32] [iscrowd, dtype=uint32]
Keypoint	[image, dtype=uint8] [keypoints, dtype=float32] [num_keypoints, dtype=uint32]
Panoptic	[image, dtype=uint8] [bbox, dtype=float32] [category_id, dtype=uint32] [iscrowd, dtype=uint32] [area, dtype=uint32]
Captioning	[image, dtype=uint8] [captions, dtype=string]

异常：

- **RuntimeError** - *dataset_dir* 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **RuntimeError** - 解析 *annotation_file* 指定的 JSON 文件失败。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - *task* 参数取值不为 'Detection'、'Stuff'、'Panoptic'、'Keypoint' 或 'Captioning'。
- **ValueError** - *annotation_file* 参数对应的文件不存在。
- **ValueError** - *dataset_dir* 参数路径不存在。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, \text{num\_shards})$ 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> coco_dataset_dir = "/path/to/coco_dataset_directory/images"
>>> coco_annotation_file = "/path/to/coco_dataset_directory/annotation_file"
>>>
>>> # 1) Read COCO data for Detection task
>>> dataset = ds.CocoDataset(dataset_dir=coco_dataset_dir,
...                           annotation_file=coco_annotation_file,
...                           task='Detection')
>>>
>>> # 2) Read COCO data for Stuff task
>>> dataset = ds.CocoDataset(dataset_dir=coco_dataset_dir,
...                           annotation_file=coco_annotation_file,
...                           task='Stuff')
```

(下页继续)

(续上页)

```

>>>
>>> # 3) Read COCO data for Panoptic task
>>> dataset = ds.CocoDataset(dataset_dir=coco_dataset_dir,
...                           annotation_file=coco_annotation_file,
...                           task='Panoptic')
>>>
>>> # 4) Read COCO data for Keypoint task
>>> dataset = ds.CocoDataset(dataset_dir=coco_dataset_dir,
...                           annotation_file=coco_annotation_file,
...                           task='Keypoint')
>>>
>>> # 5) Read COCO data for Captioning task
>>> dataset = ds.CocoDataset(dataset_dir=coco_dataset_dir,
...                           annotation_file=coco_annotation_file,
...                           task='Captioning')
>>>
>>> # In COCO dataset, each dictionary has keys "image" and "annotation"

```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：

- 当参数 *extra_metadata* 为 True 时，还需使用 *rename* 操作删除额外数据列 '_meta-filename' 的前缀 '_meta-'，否则迭代得到的数据行中不会出现此额外数据列。
- 暂不支持指定 *sampler* 参数为 *mindspore.dataset.PKSampler*。
- 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器，其与入参 *sampler* 搭配使用的效果如下。

表 75: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **COCO** 数据集:

Microsoft Common Objects in Context (COCO) 是一个大型数据集，该数据集专门为目标检测，语义分割和字幕生成任务而设计。它拥有 330K 张图像（标记数量大于 200K 个）、1500000 个目标实例、80 个目标类别、91 个对象类别、每张图片均有 5 个字幕、带关键点标注的人有 250000 个。与流行的 ImageNet 数据集相比，COCO 的类别较少，但每个类别中的图片样本非常多。

您可以解压缩原始 COCO-2017 数据集文件得到如下目录结构，并通过 MindSpore 的 API 读取。

```
.
├── coco_dataset_directory
│   ├── train2017
│   │   ├── 000000000009.jpg
│   │   ├── 000000000025.jpg
│   │   └── ...
│   ├── test2017
│   │   ├── 000000000001.jpg
│   │   ├── 000000058136.jpg
│   │   └── ...
│   ├── val2017
│   │   ├── 000000000139.jpg
│   │   ├── 000000057027.jpg
│   │   └── ...
│   └── annotation
│       ├── captions_train2017.json
│       ├── captions_val2017.json
│       ├── instances_train2017.json
│       └── instances_val2017.json
```

(下页继续)

(续上页)

```
|— person_keypoints_train2017.json
|— person_keypoints_val2017.json
```

引用:

```
@article{DBLP:journals/corr/LinMBHPRDZ14,
author    = {Tsung{-}Yi Lin and Michael Maire and Serge J. Belongie and
             Lubomir D. Bourdev and Ross B. Girshick and James Hays and
             Pietro Perona and Deva Ramanan and Piotr Doll{\'{a}}r and C. Lawrence_
             Zitnick},
title     = {Microsoft {COCO:} Common Objects in Context},
journal   = {CoRR},
volume    = {abs/1405.0312},
year      = {2014},
url       = {http://arxiv.org/abs/1405.0312},
archivePrefix = {arXiv},
eprint    = {1405.0312},
timestamp = {Mon, 13 Aug 2018 16:48:13 +0200},
biburl    = {https://dblp.org/rec/journals/corr/LinMBHPRDZ14.bib},
bibsource = {dblp computer science bibliography, https://dblp.org}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.DIV2KDataset

class mindspore.dataset.DIV2KDataset (dataset_dir, usage='train', downgrade='bicubic', scale=2, num_samples=None, num_parallel_workers=None, shuffle=None, decode=None, sampler=None, num_shards=None, shard_id=None, cache=None)

DIV2K (DIVERse 2K resolution image) 数据集。

生成的数据集有两列 `[hr_image, lr_image]`。 `hr_image` 列和 `lr_image` 列的数据类型都为 `uint8`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集。可取值为 'train'、'valid' 或 'all'。默认值: 'train'。
- **downgrade** (str, 可选) - 指定数据集的下采样的模式, 可取值为 'bicubic'、'unknown'、'mild'、'difficult' 或 'wild'。默认值: 'bicubic'。
- **scale** (int, 可选) - 指定数据集的缩放尺度。当参数 `downgrade` 取值为 'bicubic' 时, 此参数可以取值为 2、3、4、8。当参数 `downgrade` 取值为 'unknown' 时, 此参数可以取值为 2、3、4。当参数 `downgrade` 取值为 'mild'、'difficult'、'wild' 时, 此参数仅可以取值为 4。默认值: 2。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: None, 默认为 False, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。

- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `dataset_dir` 路径非法或不存在。
- **ValueError** - `usage` 参数取值不为 'train'、'valid' 或 'all'。
- **ValueError** - `downgrade` 参数取值不为 'bicubic'、'unknown'、'mild'、'difficult' 或 'wild'。
- **ValueError** - `scale` 参数取值不在给定的字段中, 或与 `downgrade` 参数的值不匹配。
- **ValueError** - `scale` 参数取值为 8, 但 `downgrade` 参数的值不为 'bicubic'。
- **ValueError** - `downgrade` 参数取值为 'mild'、'difficult' 或 'wild', 但 `scale` 参数的值不为 4。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> div2k_dataset_dir = "/path/to/div2k_dataset_directory"
>>>
>>> # 1) Get all samples from DIV2K dataset in sequence
>>> dataset = ds.DIV2KDataset(dataset_dir=div2k_dataset_dir, usage="train", scale=2,
↳downgrade="bicubic",
...                               shuffle=False)
>>>
>>> # 2) Randomly select 350 samples from DIV2K dataset
>>> dataset = ds.DIV2KDataset(dataset_dir=div2k_dataset_dir, usage="train", scale=2,
↳downgrade="bicubic",
...                               num_samples=350, shuffle=True)
>>>
>>> # 3) Get samples from DIV2K dataset for shard 0 in a 2-way distributed training
>>> dataset = ds.DIV2KDataset(dataset_dir=div2k_dataset_dir, usage="train", scale=2,
↳downgrade="bicubic",
...                               num_shards=2, shard_id=0)
>>>
>>> # In DIV2K dataset, each dictionary has keys "hr_image" and "lr_image"
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 82: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>DatasetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **DIV2K** 数据集:

DIV2K 数据集由 1000 张 2K 分辨率图像组成, 其中 800 张用于训练, 100 张用于验证, 100 张用于测试。作为 NTIRE 比赛的数据集, NTIRE 2017 和 NTIRE 2018 仅包括 DIV2K 的训练数据集和验证数据集。

您可以解压缩原始 DIV2K 数据集文件到如下目录结构, 并通过 MindSpore 的 API 进行读取。

以训练数据集作为例子。

```
.
├── DIV2K
│   ├── DIV2K_train_HR
│   │   ├── 0001.png
│   │   ├── 0002.png
│   │   └── ...
│   ├── DIV2K_train_LR_bicubic
│   │   ├── x2
│   │   │   ├── 0001x2.png
│   │   │   ├── 0002x2.png
│   │   │   └── ...
│   │   ├── x3
│   │   │   ├── 0001x3.png
│   │   │   ├── 0002x3.png
│   │   │   └── ...
│   │   └── x4
│   │       ├── 0001x4.png
│   │       ├── 0002x4.png
│   │       └── ...
│   └── DIV2K_train_LR_unknown
```

(下页继续)

(续上页)

```
|      |--- X2
|      |   |--- 0001x2.png
|      |   |--- 0002x2.png
|      |   |--- ...
|      |--- X3
|      |   |--- 0001x3.png
|      |   |--- 0002x3.png
|      |   |--- ...
|      |--- X4
|      |   |--- 0001x4.png
|      |   |--- 0002x4.png
|      |   |--- ...
|--- DIV2K_train_LR_mild
|   |--- 0001x4m.png
|   |--- 0002x4m.png
|   |--- ...
|--- DIV2K_train_LR_difficult
|   |--- 0001x4d.png
|   |--- 0002x4d.png
|   |--- ...
|--- DIV2K_train_LR_wild
|   |--- 0001x4w.png
|   |--- 0002x4w.png
|   |--- ...
|--- DIV2K_train_LR_x8
|   |--- 0001x8.png
|   |--- 0002x8.png
|   |--- ...
```

引用:

```
@InProceedings{Agustsson_2017_CVPR_Workshops,
author      = {Agustsson, Eirikur and Timofte, Radu},
title       = {NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study},
booktitle   = {The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops},
url         = "http://www.vision.ee.ethz.ch/~timofte/publications/Agustsson-CVPRW-2017.pdf",
month       = {July},
year        = {2017}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。

下页继续

表 83 – 续上页

<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。

下页继续

表 86 – 续上页

<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <code>shape</code> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.EMnistDataset

```
class mindspore.dataset.EMnistDataset (dataset_dir, name, usage=None, num_samples=None,
                                         num_parallel_workers=None, shuffle=None, sampler=None,
                                         num_shards=None, shard_id=None, cache=None)
```

EMNIST（Extended MNIST）数据集。

生成的数据集有两列: `[image, label]`。`image` 列的数据类型为 `uint8`。`label` 列的数据类型为 `uint32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **name** (str) - 按给定规则对数据集进行拆分，可以是 'byclass'、'bymerge'、'balanced'、'letters'、'digits' 或 'mnist'。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 60,000 个训练样本，取值为 'test' 时将会读取 10,000 个测试样本，取值为 'all' 时将会读取全部 70,000 个样本。默认值：None，读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。

- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, num\_shards)$ 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> emnist_dataset_dir = "/path/to/emnist_dataset_directory"
>>>
>>> # Read 3 samples from EMNIST dataset
>>> dataset = ds.EMnistDataset(dataset_dir=emnist_dataset_dir, name="mnist", num_samples=3)
>>>
>>> # Note: In emnist_dataset dataset, each dictionary has keys "image" and "label"
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

说明: 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器, 其与入参 *sampler* 搭配使用的效果如下。

表 89: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **EMNIST** 数据集：

EMNIST 数据集由一组手写字符数字组成，源自 NIST 特别版数据库 19，并转换为与 MNIST 数据集直接匹配的 28x28 像素图像格式和数据集结构。有关数据集内容和转换过程的更多信息可在 <https://arxiv.org/abs/1702.05373v1> 上查阅。

EMNIST 按照不同的规则拆分成不同的子数据集的样本数和类数如下：

按类拆分：814,255 个样本和 62 个样本不平衡类。按合并拆分：814,255 个样本和 47 个样本不平衡类。平衡拆分：131,600 个样本和 47 个样本平衡类。按字母拆分：145,600 个样本和 26 个样本平衡类。按数字拆分：280,000 个样本和 10 个样本平衡类。**MNIST**: 70,000 个样本和 10 个样本平衡类。

以下是原始 EMNIST 数据集结构。可以将数据集文件解压缩到此目录结构中，并由 MindSpore 的 API 读取。

```
.
├── mnist_dataset_dir
│   ├── emnist-mnist-train-images-idx3-ubyte
│   ├── emnist-mnist-train-labels-idx1-ubyte
│   ├── emnist-mnist-test-images-idx3-ubyte
│   ├── emnist-mnist-test-labels-idx1-ubyte
│   └── ...
```

引用：

```
@article{cohen_afshar_tapson_schaik_2017,
title      = {EMNIST: Extending MNIST to handwritten letters},
DOI        = {10.1109/ijcnn.2017.7966217},
journal    = {2017 International Joint Conference on Neural Networks (IJCNN)},
author     = {Cohen, Gregory and Afshar, Saeed and Tapson, Jonathan and Schaik, Andre Van},
year       = {2017},
```

(下页继续)

(续上页)

```
howpublished = {https://www.westernsydney.edu.au/icns/reproducible_research/
                publication_support_materials/emnist}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.FakeImageDataset

```
class mindspore.dataset.FakeImageDataset (num_images=1000, image_size=(224, 224, 3), num_classes=10,
                                           base_seed=0, num_samples=None, num_parallel_workers=None,
                                           shuffle=None, sampler=None, num_shards=None, shard_id=None,
                                           cache=None)
```

生成虚假图像构建数据集。

生成的数据集有两列: `[image, label]`。 `image` 列的数据类型为 `uint8`。 `label` 列的数据类型为 `uint32`。

参数：

- **num_images** (int, 可选) - 要生成的虚假图像数。默认值：1000。
- **image_size** (tuple, 可选) - 虚假图像的尺寸。默认值：(224, 224, 3)。
- **num_classes** (int, 可选) - 数据集的类别数。默认值：10。
- **base_seed** (int, 可选) - 生成随机图像的随机种子。默认值：0。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> # Read 3 samples from FakeImage dataset
>>> dataset = ds.FakeImageDataset(num_images=1000, image_size=(224, 224, 3),
...                               num_classes=10, base_seed=0, num_samples=3)
```

教程样例：

- [使用数据 Pipeline 加载 & 处理数据集](#)

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 96: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <code>count</code> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <code>epoch</code> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <code>buffer_size</code> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <code>count</code> 条数据。

下页继续

表 97 – 续上页

<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.FashionMnistDataset

```
class mindspore.dataset.FashionMnistDataset (dataset_dir, usage=None, num_samples=None,
                                             num_parallel_workers=None, shuffle=None, sampler=None,
                                             num_shards=None, shard_id=None, cache=None)
```

Fashion-MNIST 数据集。

生成的数据集有两列: `[image, label]`。`image` 列的数据类型为 `uint8`。`label` 列的数据类型为 `uint32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 60,000 个训练样本，取值为 'test' 时将会读取 10,000 个测试样本，取值为 'all' 时将会读取全部 70,000 个样本。默认值：None，读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。

- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, \text{num\_shards})$ 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> fashion_mnist_dataset_dir = "/path/to/fashion_mnist_dataset_directory"
>>>
>>> # Read 3 samples from FASHIONMNIST dataset
>>> dataset = ds.FashionMnistDataset(dataset_dir=fashion_mnist_dataset_dir, num_samples=3)
>>>
>>> # Note: In FASHIONMNIST dataset, each dictionary has keys "image" and "label"
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器，其与入参 *sampler* 搭配使用的效果如下。

表 103: 参数 *sampler* 和 *num_samples*，*shuffle*，*num_shards*，*shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mindspore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray</i> , <i>list</i> , <i>tuple</i> , <i>int</i> 类型	<i>int</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **Fashion-MNIST** 数据集：

Fashion-MNIST 是网络电子商城 Zalando 推出的数据集，包括 60,000 个样本的训练集和 10,000 个样本的测试集。每个示例都是一个 28x28 灰度图像，分别与 10 个类的标签关联。Fashion-MNIST 是原始 MNIST 数据集的变种，用于对机器学习算法进行基准测试。它的训练集和测试集的图像尺寸和结构相同。

可以将数据集文件解压缩到此目录结构中，并由 MindSpore 的 API 读取。

```
.
├── fashionmnist_dataset_dir
│   ├── t10k-images-idx3-ubyte
│   ├── t10k-labels-idx1-ubyte
│   ├── train-images-idx3-ubyte
│   └── train-labels-idx1-ubyte
```

引用：

```
@online{xiao2017/online,
  author      = {Han Xiao and Kashif Rasul and Roland Vollgraf},
  title       = {Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms},
  date        = {2017-08-28},
  year        = {2017},
  eprintclass  = {cs.LG},
  eprinttype   = {arXiv},
  eprint      = {cs.LG/1708.07747},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.FlickrDataset

class mindspore.dataset.FlickrDataset (dataset_dir, annotation_file, num_samples=None, num_parallel_workers=None, shuffle=None, decode=None, sampler=None, num_shards=None, shard_id=None, cache=None)

Flickr8k 和 Flickr30k 数据集。

生成的数据集有两列: [image, annotation]。image 列的数据类型为 uint8。annotation 列是一个包含 5 个标注字符的张量, 如 ["a", "b", "c", "d", "e"]。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **annotation_file** (str) - 数据集标注 JSON 文件的路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None, 表 2 中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: None, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None, 表 2 中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (DatasetCache, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `annotation_file` 参数对应的文件不存在。

- **ValueError** - `dataset_dir` 参数路径不存在。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> flickr_dataset_dir = "/path/to/flickr_dataset_directory"
>>> annotation_file = "/path/to/flickr_annotation_file"
>>>
>>> # 1) Get all samples from FLICKR dataset in sequence
>>> dataset = ds.FlickerDataset(dataset_dir=flickr_dataset_dir,
...                             annotation_file=annotation_file,
...                             shuffle=False)
>>>
>>> # 2) Randomly select 350 samples from FLICKR dataset
>>> dataset = ds.FlickerDataset(dataset_dir=flickr_dataset_dir,
...                             annotation_file=annotation_file,
...                             num_samples=350,
...                             shuffle=True)
>>>
>>> # 3) Get samples from FLICKR dataset for shard 0 in a 2-way distributed training
>>> dataset = ds.FlickerDataset(dataset_dir=flickr_dataset_dir,
...                             annotation_file=annotation_file,
...                             num_shards=2,
...                             shard_id=0)
>>>
>>> # In FLICKR dataset, each dictionary has keys "image" and "annotation"
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 110: 参数 `sampler` 和 `num_samples`, `shuffle`, `num_shards`, `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **Flickr8k** 数据集:

Flickr8k 数据集由 8092 张彩色图像组成。Flickr8k.token.txt 中有 40460 个标注，每张图像有 5 个标注。

可以将数据集文件解压缩到以下目录结构中，并由 MindSpore 的 API 读取。

```
.
├── Flickr8k
│   ├── Flickr8k_Dataset
│   │   ├── 1000268201_693b08cb0e.jpg
│   │   ├── 1001773457_577c3a7d70.jpg
│   │   └── ...
│   └── Flickr8k.token.txt
```

引用:

```
@article{DBLP:journals/jair/HodoshYH13,
author   = {Micah Hodosh and Peter Young and Julia Hockenmaier},
title    = {Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics},
journal  = {J. Artif. Intell. Res.},
volume   = {47},
pages    = {853--899},
year     = {2013},
url      = {https://doi.org/10.1613/jair.3994},
doi      = {10.1613/jair.3994},
timestamp = {Mon, 21 Jan 2019 15:01:17 +0100},
biburl   = {https://dblp.org/rec/journals/jair/HodoshYH13.bib},
```

(下页继续)

(续上页)

```
bibsource = {dblp computer science bibliography, https://dblp.org}
}
```

关于 **Flickr30k** 数据集:

Flickr30k 数据集由 31783 张彩色图像组成。results_20130124.token 中有 158915 个标注, 每个图像有 5 个标注。
可以将数据集文件解压缩到以下目录结构中, 并由 MindSpore 的 API 读取。

```
.
└─ Flickr30k
    ├── flickr30k-images
    │   ├── 1000092795.jpg
    │   ├── 10002456.jpg
    │   └── ...
    └─ results_20130124.token
```

引用:

```
@article{DBLP:journals/tacl/YoungLHH14,
author   = {Peter Young and Alice Lai and Micah Hodosh and Julia Hockenmaier},
title    = {From image descriptions to visual denotations: New similarity metrics
            for semantic inference over event descriptions},
journal  = {Trans. Assoc. Comput. Linguistics},
volume   = {2},
pages    = {67--78},
year     = {2014},
url      = {https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/229},
timestamp = {Wed, 17 Feb 2021 21:55:25 +0100},
biburl   = {https://dblp.org/rec/journals/tacl/YoungLHH14.bib},
bibsource = {dblp computer science bibliography, https://dblp.org}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理, 并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表, 按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列, 并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。

下页继续

表 111 – 续上页

<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.Flowers102Dataset

`class mindspore.dataset.Flowers102Dataset (dataset_dir, task='Classification', usage='all', num_samples=None, num_parallel_workers=1, shuffle=None, decode=False, sampler=None, num_shards=None, shard_id=None)`

Oxford 102 Flower 数据集。

根据给定的 `task` 配置，生成数据集具有不同的输出列：

- `task` 为 'Classification'，输出列：`[image, dtype=uint8]`、`[label, dtype=uint32]`。
- `task` 为 'Segmentation'，输出列：`[image, dtype=uint8]`、`[segmentation, dtype=uint8]`、`[label, dtype=uint32]`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录的路径。
- **task** (str, 可选) - 指定读取数据的任务类型，支持 'Classification' 和 'Segmentation'。默认值：'Classification'。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'valid'、'test' 或 'all'。默认值：'all'，读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，所有图像样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作进程数。默认值：1。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- **sampler** (Union[Sampler, Iterable], 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。

异常:

- **RuntimeError** - *dataset_dir* 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, num\_shards)$ 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> flowers102_dataset_dir = "/path/to/flowers102_dataset_directory"
>>> dataset = ds.Flowers102Dataset(dataset_dir=flowers102_dataset_dir,
...                               task="Classification",
...                               usage="all",
...                               decode=True)
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

说明: 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器, 其与入参 *sampler* 搭配使用的效果如下。

表 117: 参数 *sampler* 和 *num_samples*, *shuffle*, *num_shards*, *shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray, list, tuple, int</i> 类型	<i>None</i>	<i>/</i>	<i>num_samples</i>	DatasetSampler(indices = sampler , num_samples = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	DatasetSampler(sampler = sampler , num_samples = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler(num_samples = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler(replacement = True , num_samples = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler(num_samples = num_samples)

关于 **Flowers102** 数据集:

Flowers102 数据集由 102 个花类别组成, 每个类由 40 到 258 张图像组成, 这些花常见于英国。

以下是原始的 Flowers102 数据集结构。可以将数据集文件解压缩到此目录结构中, 并通过 MindSpore 的 API 读取。

```

├── floweres102_dataset_dir
│   ├── imagelabels.mat
│   ├── setid.mat
│   ├── jpg
│   │   ├── image_00001.jpg
│   │   ├── image_00002.jpg
│   │   └── ...
│   └── segmim
│       ├── segmim_00001.jpg
│       ├── segmim_00002.jpg
│       └── ...

```

引用:

```

@InProceedings{Nilsback08,
  author      = "Maria-Elena Nilsback and Andrew Zisserman",
  title       = "Automated Flower Classification over a Large Number of Classes",
  booktitle   = "Indian Conference on Computer Vision, Graphics and Image Processing",
  month       = "Dec",
  year        = "2008",
}

```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.Food101Dataset

class mindspore.dataset.Food101Dataset (*dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, decode=False, sampler=None, num_shards=None, shard_id=None, cache=None*)

Food101 数据集。

生成的数据集有两列: [image, label]。image 列的数据类型为 uint8。label 列的数据类型为 string。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 75,750 个训练样本，取值为 'test' 时将会读取 25,250 个测试样本，取值为 'all' 时将会读取全部

101,000 个样本。默认值: `None` , 读取全部样本图片。

- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: `None` , 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: `None` , 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: `None` , 下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: `False` , 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: `None` , 下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: `None` 。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: `None` 。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#) 。默认值: `None` , 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - `usage` 参数取值不为 `'train'` 、 `'test'` 或 `'all'` 。
- **ValueError** - `dataset_dir` 指定的文件夹不存在。

样例:

```
>>> import mindspore.dataset as ds
>>> food101_dataset_dir = "/path/to/food101_dataset_directory"
>>>
>>> # Read 3 samples from Food101 dataset
>>> dataset = ds.Food101Dataset(dataset_dir=food101_dataset_dir, num_samples=3)
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

说明: 入参 `num_samples` 、 `shuffle` 、 `num_shards` 、 `shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 124: 参数 *sampler* 和 *num_samples* , *shuffle* , *num_shards* , *shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray</i> , <i>list</i> , <i>tuple</i> , <i>int</i> 类型	<i>None</i>	<i>/</i>	<i>num_samples</i>	SubsetSampler (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **Food101** 数据集:

Food101 是一个包含 101 种食品类别的数据集, 共 101000 张图片。每一个类别有 250 张测试图片和 750 张训练图片。所有图像都被重新缩放, 最大边长为 512 像素。

以下为原始 **Food101** 数据集的结构, 您可以将数据集文件解压得到如下的文件结构, 并通过 **MindSpore** 的 API 进行读取。

```
.
├── food101_dir
│   ├── images
│   │   ├── apple_pie
│   │   │   ├── 1005649.jpg
│   │   │   ├── 1014775.jpg
│   │   │   └── ...
│   │   ├── baby_back_rips
│   │   │   ├── 1005293.jpg
│   │   │   ├── 1007102.jpg
│   │   │   └── ...
│   │   └── ...
│   └── meta
│       ├── train.txt
│       ├── test.txt
│       ├── classes.txt
│       ├── train.json
│       ├── test.json
│       └── train.txt
```

引用:

```
@inproceedings{bossard14,
  title      = {Food-101 -- Mining Discriminative Components with Random Forests},
  author     = {Bossard, Lukas and Guillaumin, Matthieu and Van Gool, Luc},
  booktitle  = {European Conference on Computer Vision},
  year       = {2014}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.ImageFolderDataset

```
class mindspore.dataset.ImageFolderDataset (dataset_dir, num_samples=None, num_parallel_workers=None,
                                             shuffle=None, sampler=None, extensions=None,
                                             class_indexing=None, decode=False, num_shards=None,
                                             shard_id=None, cache=None, decrypt=None)
```

从树状结构的文件目录中读取图片构建源数据集。同一个文件夹中的所有图片将被分配相同的 label。

生成的数据集有两列：[image, label]。image 列的数据类型为 uint8。label 列的数据类型为 uint32。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录的路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **extensions** (list[str], 可选) - 指定文件的扩展名，仅读取与指定扩展名匹配的文件到数据集中。默认值：None。
- **class_indexing** (dict, 可选) - 指定文件夹名称到 label 索引的映射，要求映射规则为 string 到 int。文件夹名称将按字母顺序排列，索引值从 0 开始，并且要求每个文件夹名称对应的索引值唯一。默认值：None，不指定。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (DatasetCache, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。
- **decrypt** (callable, 可选) - 图像解密函数，接受加密的图片路径并返回 bytes 类型的解密数据。默认值：None，不进行解密。

异常：

- **RuntimeError** - `dataset_dir` 不包含任何数据文件。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **RuntimeError** - `class_indexing` 参数的类型不是 dict。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> image_folder_dataset_dir = "/path/to/image_folder_dataset_directory"
>>>
>>> # 1) Read all samples (image files) in image_folder_dataset_dir with 8 threads
>>> dataset = ds.ImageFolderDataset(dataset_dir=image_folder_dataset_dir,
...                                num_parallel_workers=8)
>>>
>>> # 2) Read all samples (image files) from folder cat and folder dog with label 0 and 1
>>> dataset = ds.ImageFolderDataset(dataset_dir=image_folder_dataset_dir,
...                                class_indexing={"cat":0, "dog":1})
>>>
>>> # 3) Read all samples (image files) in image_folder_dataset_dir with extensions .JPEG
>>> #      and .png (case sensitive)
>>> dataset = ds.ImageFolderDataset(dataset_dir=image_folder_dataset_dir,
...                                extensions=[".JPEG", ".png"])
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明:

- 如果 `decode` 参数的值为 `False` , 则得到的 `image` 列的 `shape` 为 `[undecoded_image_size]` , 如果为 `True` 则 `image` 列的 `shape` 为 `[H,W,C]`。
- 入参 `num_samples` 、 `shuffle` 、 `num_shards` 、 `shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 131: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **ImageFolderDataset**:

您可以将图片数据文件构建成如下目录结构，并通过 MindSpore 的 API 进行读取。

```
.
├── image_folder_dataset_directory
│   ├── class1
│   │   ├── 0000000000001.jpg
│   │   ├── 0000000000002.jpg
│   │   └── ...
│   ├── class2
│   │   ├── 0000000000001.jpg
│   │   ├── 0000000000002.jpg
│   │   └── ...
│   ├── class3
│   │   ├── 0000000000001.jpg
│   │   ├── 0000000000002.jpg
│   │   └── ...
│   ├── classN
│   └── ...
```

预处理操作

<code><i>mindspore.dataset.Dataset.apply</i></code>	对数据集对象执行给定操作函数。
<code><i>mindspore.dataset.Dataset.concat</i></code>	对传入的多个数据集对象进行拼接操作。
<code><i>mindspore.dataset.Dataset.filter</i></code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code><i>mindspore.dataset.Dataset.flat_map</i></code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code><i>mindspore.dataset.Dataset.map</i></code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code><i>mindspore.dataset.Dataset.project</i></code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code><i>mindspore.dataset.Dataset.rename</i></code>	对数据集对象按指定的列名进行重命名。
<code><i>mindspore.dataset.Dataset.repeat</i></code>	重复此数据集 <i>count</i> 次。
<code><i>mindspore.dataset.Dataset.reset</i></code>	重置下一个 <i>epoch</i> 的数据集对象。
<code><i>mindspore.dataset.Dataset.save</i></code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code><i>mindspore.dataset.Dataset.shuffle</i></code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code><i>mindspore.dataset.Dataset.skip</i></code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code><i>mindspore.dataset.Dataset.split</i></code>	将数据集拆分为多个不重叠的子数据集。
<code><i>mindspore.dataset.Dataset.take</i></code>	截取数据集的前指定条数据。
<code><i>mindspore.dataset.Dataset.zip</i></code>	将多个 dataset 对象按列进行合并压缩，多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.KITTI**Dataset**

class mindspore.dataset.**KITTI****Dataset** (*dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, decode=False, sampler=None, num_shards=None, shard_id=None, cache=None*)

KITTI 数据集。

当 *usage* 为 'train' 时, 生成的数据集有多列: [*image, label, truncated, occluded, alpha, bbox, dimensions, location, rotation_y*]; 当 *usage* 为 'test' 时, 生成的数据集只有一列 [*image*]。*image* 列的数据类型为 `uint8`。*label* 列的数据类型为 `uint32`。*truncated* 列的数据类型为 `float32`。*occluded* 列的数据类型为 `uint32`。*alpha* 列的数据类型为 `float32`。*bbox* 列的数据类型为 `float32`。*dimensions* 列的数据类型为 `float32`。*location* 列的数据类型为 `float32`。*rotation_y* 列的数据类型为 `float32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train' 或 'test'。取值为 'train' 时将会读取 7481 个训练样本, 取值为 'test' 时将会读取 7518 个不带标签的测试样本。默认值: `None`, 将使用 'train'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: `None`, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: `None`, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: `None`。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: `False`, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: `None`。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: `None`。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: `None`。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: `None`, 不使用缓存。

异常:

- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。

- **ValueError** - `dataset_dir` 对应目录不存在。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> kitti_dataset_dir = "/path/to/kitti_dataset_directory"
>>>
>>> # 1) Read all KITTI train dataset samples in kitti_dataset_dir in sequence
>>> dataset = ds.KITTIIDataset(dataset_dir=kitti_dataset_dir, usage="train")
>>>
>>> # 2) Read then decode all KITTI test dataset samples in kitti_dataset_dir in sequence
>>> dataset = ds.KITTIIDataset(dataset_dir=kitti_dataset_dir, usage="test",
...                             decode=True, shuffle=False)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 138: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler, num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler, num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards, shard_id = shard_id, shuffle = True, num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards, shard_id = shard_id, shuffle = False, num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True, num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **KITTI** 数据集：

KITTI（Karlsruhe Institute of Technology and Toyota Technological Institute）是移动机器人和自动驾驶领域的常用数据集之一。它由用高分辨率的 RGB、灰度立体相机和三维激光扫描仪等各种传感器记录的数小时的交通场景组成。尽管

它很常用，但该数据集本身并不包含用于语义分割的目标值。然而，许多研究人员已经对该数据集的部分内容进行了手工标注以适应需求。Álvarez 等人为道路检测挑战中的 323 幅图像生成了目标值，包含 3 个类别：道路、车辆和天空。Zhang 等人为追踪挑战中的 252 张（140 张用于训练，112 张用于测试）RGB 和激光扫描图像进行了标注，包含 10 个类别：建筑、天空、道路、植被、人行道、汽车、行人、自行车、标志/杆、栅栏。

您可以解压原始 KITTI 数据集文件构建成如下目录结构，并通过 MindSpore 的 API 进行读取。

```
.
├── kitti_dataset_directory
│   ├── data_object_image_2
│   │   ├── training
│   │   │   ├── image_2
│   │   │   │   ├── 0000000000001.jpg
│   │   │   │   ├── 0000000000002.jpg
│   │   │   │   └── ...
│   │   └── testing
│   │       ├── image_2
│   │       │   ├── 0000000000001.jpg
│   │       │   ├── 0000000000002.jpg
│   │       │   └── ...
│   └── data_object_label_2
│       ├── training
│       │   ├── label_2
│       │   │   ├── 0000000000001.jpg
│       │   │   ├── 0000000000002.jpg
│       │   │   └── ...
```

引用：

```
@INPROCEEDINGS{Geiger2012CVPR,
author={Andreas Geiger and Philip Lenz and Raquel Urtasun},
title={Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite},
booktitle={Conference on Computer Vision and Pattern Recognition (CVPR)},
year={2012}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 epoch 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。

下页继续

表 139 – 续上页

<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <code>buffer_size</code> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <code>count</code> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <code>dataset</code> 对象按列进行合并压缩, 多个 <code>dataset</code> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <code>batch_size</code> 条数据组合为一个批数据, 并可通过可选参数 <code>per_batch_map</code> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <code>batch_size</code> 条数据组合为一个批数据, 并可通过可选参数 <code>pad_info</code> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <code>epoch</code> 中的 <code>batch</code> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <code>RepeatDataset</code> 中定义的 <code>repeat</code> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <code>shape</code> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.KMnistDataset

```
class mindspore.dataset.KMnistDataset (dataset_dir, usage=None, num_samples=None,
                                       num_parallel_workers=None, shuffle=None, sampler=None,
                                       num_shards=None, shard_id=None, cache=None)
```

KMNIST（Kuzushiji-MNIST）数据集。

生成的数据集有两列: `[image, label]`。`image` 列的数据类型为 `uint8`。`label` 列的数据类型为 `uint32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 60,000 个训练样本，取值为 'test' 时将会读取 10,000 个测试样本，取值为 'all' 时将会读取全部 70,000 个样本。默认值：None，读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。

- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, \text{num\_shards})$ 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> kmnist_dataset_dir = "/path/to/kmnist_dataset_directory"
>>>
>>> # Read 3 samples from KMNIST dataset
>>> dataset = ds.KMnistDataset(dataset_dir=kmnist_dataset_dir, num_samples=3)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器，其与入参 *sampler* 搭配使用的效果如下。

表 145: 参数 *sampler* 和 *num_samples*，*shuffle*，*num_shards*，*shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray</i> , <i>list</i> , <i>tuple</i> , <i>int</i> 类型	<i>int</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **KMNIST** 数据集：

KMNIST 是一个数据集，改编自 Kuzushiji 数据集，作为 MNIST 数据集的替代数据集（MNIST 数据集是机器学习社区中著名的数据集）。以下是原始 KMNIST 数据集结构。可以将数据集文件解压缩到此目录结构中，并由 MindSpore 的 API 读取。

```
.
└─ kmnist_dataset_dir
    ├── t10k-images-idx3-ubyte
    ├── t10k-labels-idx1-ubyte
    ├── train-images-idx3-ubyte
    └─ train-labels-idx1-ubyte
```

引用:

```
@online{clanuwat2018deep,
  author      = {Tarin Clanuwat and Mikel Bober-Irizar and Asanobu Kitamoto and
                 Alex Lamb and Kazuaki Yamamoto and David Ha},
  title       = {Deep Learning for Classical Japanese Literature},
  date        = {2018-12-03},
  year        = {2018},
  eprintclass = {cs.CV},
  eprinttype  = {arXiv},
  eprint      = {cs.CV/1812.01718},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.LFWDataset

```
class mindspore.dataset.LFWDataset (dataset_dir, task=None, usage=None, image_set=None, num_samples=None,
                                     num_parallel_workers=None, shuffle=None, decode=False, sampler=None,
                                     num_shards=None, shard_id=None, cache=None)
```

LFW (Labeled Faces in the Wild) 数据集。

当 *task* 为 'people' 时，生成的数据集有两列: [*image*, *label*]；当 *task* 为 'pairs' 时，生成的数据集有三列: [*image1*, *image2*, *label*]。*image* 列的数据类型为 uint8。*image1* 列的数据类型为 uint8。*image2* 列的数据类型为 uint8。*label* 列的数据类型为 uint32。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **task** (str, 可选) - 指定读取 LFW 数据集的任务类型，支持 'people' 和 'pairs'。默认值：None，表示 'people'。
- **usage** (str, 可选) - 指定数据集的子集，支持 '10fold'、'train'、'test' 和 'all'。默认值：None，将读取 'train' 和 'test' 子集。
- **image_set** (str, 可选) - 指定读取子集的 Image Funneling 类型，支持 'original'、'funneled' 或 'deepfunneled'。默认值：None，将读取 'funneled' 子集。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部图像。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - *dataset_dir* 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。

- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> # 1) Read LFW People dataset
>>> lfw_people_dataset_dir = "/path/to/lfw_people_dataset_directory"
>>> dataset = ds.LFWDataset(dataset_dir=lfw_people_dataset_dir, task="people", usage="10fold",
...                          image_set="original")
>>>
>>> # 2) Read LFW Pairs dataset
>>> lfw_pairs_dataset_dir = "/path/to/lfw_pairs_dataset_directory"
>>> dataset = ds.LFWDataset(dataset_dir=lfw_pairs_dataset_dir, task="pairs", usage="test",
...                          image_set="funneled")
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 152: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **LFW** 数据集:

LFW (Labelled Faces in the Wild) 数据集是人脸识别领域最常用和广泛的开放数据集之一，它由美国马萨诸塞理工学院的 Gary B. Huang 等人于 2007 年发布。该数据集包含 13,233 个人的近 50,000 张图像，这些图像来自互联网上不同来源的人物照片，并包含了不同的姿势、光照、角度等不同环境因素。该数据集中大部分图像都是正面正视的，而且包含多种年龄、性别和人种。

你可以将原始的 LFW 数据集文件解压成以下目录结构，并通过 MindSpore 的 API 进行读取。

```
.
├── lfw_dataset_directory
│   ├── lfw
│   │   ├── Aaron_Eckhart
│   │   │   ├── Aaron_Eckhart_0001.jpg
│   │   │   └── ...
│   │   ├── Abbas_Kiarostami
│   │   │   ├── Abbas_Kiarostami_0001.jpg
│   │   │   └── ...
│   │   └── ...
│   ├── lfw-deepfunneled
│   │   ├── Aaron_Eckhart
│   │   │   ├── Aaron_Eckhart_0001.jpg
│   │   │   └── ...
│   │   ├── Abbas_Kiarostami
│   │   │   ├── Abbas_Kiarostami_0001.jpg
│   │   │   └── ...
│   │   └── ...
│   ├── lfw_funneled
│   │   ├── Aaron_Eckhart
│   │   │   ├── Aaron_Eckhart_0001.jpg
│   │   │   └── ...
│   │   ├── Abbas_Kiarostami
│   │   │   ├── Abbas_Kiarostami_0001.jpg
│   │   │   └── ...
│   │   └── ...
│   ├── lfw-names.txt
│   ├── pairs.txt
│   ├── pairsDevTest.txt
│   ├── pairsDevTrain.txt
│   ├── people.txt
│   ├── peopleDevTest.txt
│   └── peopleDevTrain.txt
```

引用：

```
@TechReport{LFWTech,
  title={LFW: A Database for Studying Recognition in Unconstrained Environments},
  author={Gary B. Huang and Manu Ramesh and Tamara Berg and Erik Learned-Miller},
  institution={University of Massachusetts, Amherst},
  year={2007}
  number={07-49},
  month={October},
  howpublished={http://vis-www.cs.umass.edu/lfw}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.LSUNDataset

class mindspore.dataset.LSUNDataset (*dataset_dir, usage=None, classes=None, num_samples=None, num_parallel_workers=None, shuffle=None, decode=False, sampler=None, num_shards=None, shard_id=None, cache=None*)

LSUN（Large-scale Scene UNderstanding）数据集。

生成的数据集有两列: *[image, label]*。 *image* 列的数据类型为 `uint8`。 *label* 列的数据类型为 `int32`。

参数：

- dataset_dir** (str) - 包含数据集文件的根目录路径。
- usage** (str, 可选) - 指定数据集的子集，可为 'train'、'test'、'valid' 或 'all'。默认值: None，将设置为 'all'。

- **classes** (Union[str, list[str]], 可选) - 读取数据集指定的类别。默认值: None, 表示读取所有类别。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部图像。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: False, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **ValueError** - `usage` 或 `classes` 参数错误 (不为可选的类别)。

样例:

```
>>> import mindspore.dataset as ds
>>> lsun_dataset_dir = "/path/to/lsun_dataset_directory"
>>>
>>> # 1) Read all samples (image files) in lsun_dataset_dir with 8 threads
>>> dataset = ds.LSUNDataset(dataset_dir=lsun_dataset_dir,
...                           num_parallel_workers=8)
>>>
>>> # 2) Read all train samples (image files) from folder "bedroom" and "classroom"
>>> dataset = ds.LSUNDataset(dataset_dir=lsun_dataset_dir, usage="train",
...                           classes=["bedroom", "classroom"])
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 159: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>DatasetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **LSUN** 数据集：

LSUN（**Large-Scale Scene Understanding**）是一个大规模数据集，用于室内场景理解。**LSUN** 最初是在 2015 年由斯坦福大学推出的，旨在为计算机视觉和机器学习领域的研究提供一个具有挑战性和多样性的数据集。该数据集的主要应用是室内场景分析。

该数据集包含了十种不同的场景类别，包括卧室、客厅、餐厅、起居室、书房、厨房、浴室、走廊、儿童房和室外。每种类别都包含了来自不同视角的数万张图像，并且这些图像都是高质量、高分辨率的真实世界图像。

您可以解压原始的数据集文件构建如下目录结构，并通过 **MindSpore** 的 API 进行读取。

```
.
├── lsun_dataset_directory
│   ├── test
│   │   ├── ...
│   ├── bedroom_train
│   │   ├── 1_1.jpg
│   │   ├── 1_2.jpg
│   ├── bedroom_val
│   │   ├── ...
│   ├── classroom_train
│   │   ├── ...
│   ├── classroom_val
│   │   ├── ...
```

引用：

```
article{yu15lsun,
  title={LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop},
  (下页继续)
```


(续上页)

```
author={Yu, Fisher and Zhang, Yinda and Song, Shuran and Seff, Ari and Xiao, Jianxiong},
journal={arXiv preprint arXiv:1506.03365},
year={2015}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

`mindspore.dataset.ManifestDataset`

```
class mindspore.dataset.ManifestDataset (dataset_file, usage='train', num_samples=None,  

num_parallel_workers=None, shuffle=None, sampler=None,  

class_indexing=None, decode=False, num_shards=None,  

shard_id=None, cache=None)
```

读取和解析 Manifest 数据文件构建数据集。

生成的数据集有两列: `[image, label]`。 `image` 列的数据类型为 `uint8` 类型。 `label` 列的数据类型为 `uint64` 类型。

参数:

- **dataset_file** (str) - 数据集文件的目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'eval' 或 'inference'。默认值: 'train'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **class_indexing** (dict, 可选) - 指定一个从 label 名称到 label 索引的映射, 要求映射规则为 string 到 int。索引值从 0 开始, 并且要求每个 label 名称对应的索引值唯一。默认值: None, 不指定。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: False, 不解码。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_files` 路径下不包含任何数据文件。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **RuntimeError** - `class_indexing` 参数的类型不是 dict。
- **ValueError** - `shard_id` 参数值错误 (小于 0 或者大于等于 `num_shards`)。

样例:

```
>>> import mindspore.dataset as ds
>>> manifest_dataset_dir = "/path/to/manifest_dataset_file"
>>>
>>> # 1) Read all samples specified in manifest_dataset_dir dataset with 8 threads for
    ↳ training
>>> dataset = ds.ManifestDataset(dataset_file=manifest_dataset_dir, usage="train", num_
    ↳ parallel_workers=8)
>>>
>>> # 2) Read samples (specified in manifest_file.manifest) for shard 0 in a 2-way
    ↳ distributed training setup
>>> dataset = ds.ManifestDataset(dataset_file=manifest_dataset_dir, num_shards=2, shard_id=0)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：

- 如果 *decode* 为 *False*，*image* 列返回图像的一维原始字节。否则，将返回 *shape* 为 *[H, W, C]* 的解码图像。
- 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器，其与入参 *sampler* 搭配使用的效果如下。

表 166: 参数 *sampler* 和 *num_samples*，*shuffle*，*num_shards*，*shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuf- fle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray, list, tuple, int</i> 类型	<i>int</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>indices</i> = sampler ， <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>sampler</i> = sampler ， <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards ， <i>shard_id</i> = shard_id ， <i>shuffle</i> = True ， <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards ， <i>shard_id</i> = shard_id ， <i>shuffle</i> = False ， <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True ， <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **Manifest** 数据集：

Manifest 文件包含数据集中包含的文件列表，包括文件名和文件 ID 等基本文件信息，以及扩展文件元数据。Manifest 是华为 ModelArts 支持的数据格式文件，详细说明请参见 [Manifest 文档](#)。

以下是原始 **Manifest** 数据集结构。可以将数据集文件解压缩到此目录结构中，并由 MindSpore 的 API 读取。

```
.
├── manifest_dataset_directory
│   ├── train
│   │   ├── 1.JPEG
│   │   ├── 2.JPEG
│   │   └── ...
│   └── eval
│       ├── 1.JPEG
│       ├── 2.JPEG
│       └── ...
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.MnistDataset

```
class mindspore.dataset.MnistDataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None,
                                         shuffle=None, sampler=None, num_shards=None, shard_id=None,
                                         cache=None)
```

MNIST 数据集。

生成的数据集有两列: `[image, label]`。 `image` 列的数据类型为 `uint8`。 `label` 列的数据类型为 `uint32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 60,000 个训练样本, 取值为 'test' 时将会读取 10,000 个测试样本, 取值为 'all' 时将会读取全部 70,000 个样本。默认值: None, 读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - `usage` 参数取值不为 'train'、'test' 或 'all'。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> mnist_dataset_dir = "/path/to/mnist_dataset_directory"
>>>
>>> # Read 3 samples from MNIST dataset
>>> dataset = ds.MnistDataset(dataset_dir=mnist_dataset_dir, num_samples=3)
>>>
>>> # Note: In mnist_dataset dataset, each dictionary has keys "image" and "label"
```


教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 173: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 MNIST 数据集：

MNIST 手写数字数据集是 NIST 数据集的子集，共有 60,000 个训练样本和 10,000 个测试样本。此数据集是 NIST 数据集的子集。数字已经预先进行了尺寸归一化和中心化处理。

以下为原始 MNIST 数据集的结构。您可以将数据集文件解压得到如下的文件结构，并通过 MindSpore 的 API 进行读取。

```
.
├── mnist_dataset_dir
│   ├── t10k-images-idx3-ubyte
│   ├── t10k-labels-idx1-ubyte
│   ├── train-images-idx3-ubyte
│   └── train-labels-idx1-ubyte
```

引用：

```
@article{lecun2010mnist,
title      = {MNIST handwritten digit database},
author     = {LeCun, Yann and Cortes, Corinna and Burges, CJ},
```

(下页继续)

(续上页)

```
journal      = {ATT Labs [Online]},
volume       = {2},
year         = {2010},
howpublished = {http://yann.lecun.com/exdb/mnist}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

`mindspore.dataset.OmniglotDataset`

```
class mindspore.dataset.OmniglotDataset (dataset_dir, background=None, num_samples=None,
                                         num_parallel_workers=None, shuffle=None, decode=False,
                                         sampler=None, num_shards=None, shard_id=None, cache=None)
```

Omniglot 数据集。

生成的数据集有两列: `[image, label]`。`image` 列的数据类型为 `uint8`。`label` 列的数据类型为 `int32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **background** (bool, 可选) - 是否使用 'background' 集来创建数据集，否则使用 'evaluation' 集创建数据集。默认值：None，将被设为 True。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> omniglot_dataset_dir = "/path/to/omniglot_dataset_directory"
>>> dataset = ds.OmniglotDataset(dataset_dir=omniglot_dataset_dir,
...                               num_parallel_workers=8)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 180: 参数 `sampler` 和 `num_samples`、`shuffle`、`num_shards`、`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **Omniglot** 数据集：

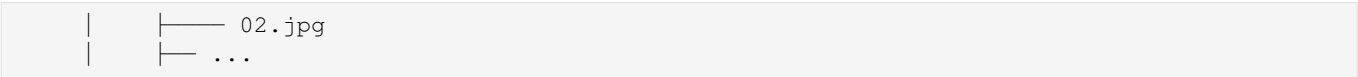
Omniglot 数据集是为开发更像人类的学习算法而设计的。它包含来自 50 个不同字母的 1623 个不同的手写字符。这 1623 个字符中的每一个都是由 20 个不同的人通过亚马逊的 Mechanical Turk 在线绘制的。每张图片都与一个笔画数数据配对，由形如 `[x, y, t]` 的坐标、时间序列表示，时间单位为毫秒。

您可以解压原始 Omniglot 数据集文件构建成如下目录结构，并通过 MindSpore 的 API 进行读取。

```
.
├── omniglot_dataset_directory
│   ├── images_background/
│   │   ├── character_class1/
│   │   │   ├── 01.jpg
│   │   │   ├── 02.jpg
│   │   │   └── ...
│   │   └── character_class2/
│   │       ├── 01.jpg
│   │       ├── 02.jpg
│   │       └── ...
│   └── images_evaluation/
│       ├── character_class1/
│       │   ├── 01.jpg
│       │   ├── 02.jpg
│       │   └── ...
│       └── character_class2/
│           ├── 01.jpg
│           ├── 02.jpg
│           └── ...
```

(下页继续)

(续上页)



引用:

```
@article{lake2015human,
  title={Human-level concept learning through probabilistic program induction},
  author={Lake, Brenden M and Salakhutdinov, Ruslan and Tenenbaum, Joshua B},
  journal={Science},
  volume={350},
  number={6266},
  pages={1332--1338},
  year={2015},
  publisher={American Association for the Advancement of Science}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。

下页继续

表 182 – 续上页

<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.PhotoTourDataset

class mindspore.dataset.**PhotoTourDataset** (*dataset_dir, name, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None*)

PhotoTour 数据集。

根据给定的 *usage* 配置, 生成数据集具有不同的输出列:

- *usage* = 'train', 输出列: [*image*, *dtype=uint8*] 。
- *usage* ≠ 'train', 输出列: [*image1*, *dtype=uint8*] 、 [*image2*, *dtype=uint8*] 、 [*matches*, *dtype=uint32*] 。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **name** (str) - 要加载的数据集内容名称, 可以取值为 'notredame' 、 'yosemite' 、 'liberty' 、 'notredame_harris' 、 'yosemite_harris' 或 'liberty_harris' 。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train' 或 'test'。默认值: None , 将被设置为 'train' 。取值为 'train' 时, 每个 *name* 的数据集样本数分别为 {'notredame': 468159, 'yosemite': 633587, 'liberty': 450092, 'liberty_harris': 379587, 'yosemite_harris': 450912, 'notredame_harris': 325295}。取值为 'test' 时, 将读取 100,000 个测试样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None , 读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None , 使用全局默认线程数 (8), 也可以通过`mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None 。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None 。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None 。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None 。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#) 。默认值: None , 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。

- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `dataset_dir` 不存在。
- **ValueError** - `usage` 不是 'train' 或 'test' 。
- **ValueError** - `name` 不是 'notredame' 、 'yosemite' 、 'liberty' 、 'notredame_harris' 、 'yosemite_harris' 或 'liberty_harris' 。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> # Read 3 samples from PhotoTour dataset.
>>> dataset = ds.PhotoTourDataset(dataset_dir="/path/to/photo_tour_dataset_directory",
...                               name='liberty', usage='train', num_samples=3)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 187: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **PhotoTour** 数据集：

数据取自许愿池（罗马）、巴黎圣母院（巴黎）和半圆顶（美国约塞米蒂国家公园）的旅游圣地照片。每个数据集包括一系列相应的图像块，是通过将旅游圣地的照片中的 3D 点投影回到原始图像而获得的。

数据集由 1024 x 1024 位图（.bmp）图像组成，每个图像都包含 16 x 16 的图像修补数组。每个图像块都以 64 x 64 灰度采样，具有规范的比例和方向。有关如何确定比例和方向的详细信息，请参见论文。关联的元数据文件 `info.txt` 包含匹配信息。`info.txt` 的每一行对应一个单独的图像块，图像块在每个位图图像中从左到右、从上到下顺序排列。`info.txt` 每行上的第一个数字是采样该图像块的 3D 点 ID——具有相同 3D 点 ID 的图像块从同一 3D 点投影（到不同的图像中）。`info.txt` 中的第二个数字代表图像块是从哪个原始图像采样得到，目前未使用。

可以将原始 **PhotoTour** 数据集文件解压缩到此目录结构中，并通过 MindSpore 的 API 读取。

```
.
├── photo_tour_dataset_directory
│   ├── liberty/
│   │   ├── info.txt           // two columns: 3D_point_ID, unused
│   │   ├── m50_100000_100000_0.txt // seven columns: patch_ID1, 3D_point_ID1, unused1,
│   │   │                               // patch_ID2, 3D_point_ID2, unused2, unused3
│   │   ├── patches0000.bmp      // 1024*1024 pixels, with 16 * 16 patches.
│   │   ├── patches0001.bmp
│   │   ├── ...
│   ├── yosemite/
│   │   ├── ...
│   ├── notredame/
│   │   ├── ...
│   ├── liberty_harris/
│   │   ├── ...
│   ├── yosemite_harris/
│   │   ├── ...
│   ├── notredame_harris/
│   │   ├── ...
```

引用：

```
@INPROCEEDINGS{4269996,
  author={Winder, Simon A. J. and Brown, Matthew},
  booktitle={2007 IEEE Conference on Computer Vision and Pattern Recognition},
  title={Learning Local Image Descriptors},
  year={2007},
  volume={},
  number={},
  pages={1-8},
  doi={10.1109/CVPR.2007.382971}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。

下页继续

表 188 – 续上页

<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。

下页继续

表 191 – 续上页

<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <code>repeat</code> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <code>shape</code> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器, 保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.Places365Dataset

```
class mindspore.dataset.Places365Dataset (dataset_dir, usage=None, small=True, decode=False,
                                           num_samples=None, num_parallel_workers=None, shuffle=None,
                                           sampler=None, num_shards=None, shard_id=None, cache=None)
```

Places365 数据集。

生成的数据集有两列: *[image, label]*。 *image* 列的数据类型为 `uint8`。 *label* 列的数据类型为 `uint32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train-standard'、'train-challenge' 或 'val'。默认值: None, 将使用 'train-standard'。
- **small** (bool, 可选) - 是否使用 256*256 的低分辨率图像 (True) 或高分辨率图像 (False)。默认值: True, 使用低分辨率图像。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: False, 不解码。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。

- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - *shard_id* 参数错误, 参数小于 0 或者大于等于 *num_shards*。
- **ValueError** - *usage* 不是 'train-standard'、'train-challenge' 或 'val'。

样例:

```
>>> import mindspore.dataset as ds
>>> place365_dataset_dir = "/path/to/place365_dataset_directory"
>>>
>>> # Read 3 samples from Places365 dataset
>>> dataset = ds.Places365Dataset(dataset_dir=place365_dataset_dir, usage='train-standard',
...                               small=True, decode=True, num_samples=3)
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

说明: 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器, 其与入参 *sampler* 搭配使用的效果如下。

表 194: 参数 *sampler* 和 *num_samples* , *shuffle* , *num_shards* , *shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray, list, tuple, int</i> 类型	<i>int</i>	<i>/</i>	<i>num_samples</i>	DatasetSampler (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IterSampler (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **Places365** 数据集:

在 **Places2** 数据库上训练的卷积神经网络 (CNN) 可用于场景识别, 也可用于视觉识别的通用深度场景特征。

Places 作者向公众发布了 **Places365-Standard** 数据集和 **Places365-Challenge** 数据集。**Places365-Standard** 数据集是 **Places2** 数据库的核心集, 该数据库已用于训练 **Places365-CNN**。**Places** 作者将在未来的 **Places365-Standard** 数据集上添加其他类型的标注。**Places365-Challenge** 数据集是 **Places2** 数据库的竞赛数据集, 与 **Places365-Standard** 数据集相比, 该数据库有 620 万张额外的图像。此数据集用于 2016 年的 **Places** 挑战赛。

可以将原始的 **Places365** 数据集文件解压缩到此目录结构中, 并由 **MindSpore** 的 API 读取。

```
.
├── categories_places365
│   ├── places365_train-standard.txt
│   ├── places365_train-challenge.txt
│   └── val_large/
│       ├── Places365_val_00000001.jpg
│       ├── Places365_val_00000002.jpg
│       ├── Places365_val_00000003.jpg
│       └── ...
│   └── val_256/
│       └── ...
├── data_large_standard/
│   └── ...
├── data_256_standard/
│   └── ...
├── data_large_challenge/
│   └── ...
```

(下页继续)

(续上页)

```
├─ data_256_challenge /
│   └─ ...
```

引用:

```
article{zhou2017places,
  title={Places: A 10 million Image Database for Scene Recognition},
  author={Zhou, Bolei and Lapedriza, Agata and Khosla, Aditya and Oliva, Aude and Torralba, Antonio},
  journal={IEEE Transactions on Pattern Analysis and Machine Intelligence},
  year={2017},
  publisher={IEEE}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。

下页继续

表 196 – 续上页

<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.QMnistDataset

```
class mindspore.dataset.QMnistDataset (dataset_dir, usage=None, compat=True, num_samples=None,
                                         num_parallel_workers=None, shuffle=None, sampler=None,
                                         num_shards=None, shard_id=None, cache=None)
```

QMnist 数据集。

生成的数据集有两列: `[image, label]`。 `image` 列的数据类型为 `uint8`。 `label` 列的数据类型为 `uint32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test'、'test10k'、'test50k'、'nist' 或 'all'。默认值: None, 读取所有子集。
- **compat** (bool, 可选) - 指定每个样本的标签信息。若为 True, 指定每个样本的标签是类别号, 否则指定标签是完整的 QMnist 信息。默认值: True, 标签为类别号。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> qmnist_dataset_dir = "/path/to/qmnist_dataset_directory"
>>>
>>> # Read 3 samples from QMNIST train dataset
>>> dataset = ds.QMnistDataset(dataset_dir=qmnist_dataset_dir, num_samples=3)
>>>
>>> # Note: In QMNIST dataset, each dictionary has keys "image" and "label"
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 201: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mindspore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>None</code>	<code>True</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
iterable 类型	<code>None</code>	<code>True</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 QMNIST 数据集:

QMNIST 数据集是从 NIST Special Database 19 中的原始数据生成的, 目的是尽可能地匹配 MNIST 预处理。研究人员试图生成额外的 50k 类似 MNIST 数据的图像。在 QMNIST 论文中, 作者给出了重建过程, 并使用匈牙利算法来找到原始 MNIST 样本与其重建样本之间的最佳匹配。

以下是原始的 QMNIST 数据集结构。可以将数据集文件解压缩到此目录结构中, 并通过 MindSpore 的 API 读取。

```
.
├── qmnist_dataset_dir
│   ├── qmnist-train-images-idx3-ubyte
│   ├── qmnist-train-labels-idx2-int
│   ├── qmnist-test-images-idx3-ubyte
│   ├── qmnist-test-labels-idx2-int
│   ├── xnist-images-idx3-ubyte
│   └── xnist-labels-idx2-int
```

引用:

```
@incollection{qmnist-2019,
  title = "Cold Case: The Lost MNIST Digits",
  author = "Chhavi Yadav and L\{'e}on Bottou", \
  booktitle = {Advances in Neural Information Processing Systems 32},
  year = {2019},
  publisher = {Curran Associates, Inc.},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.RenderedSST2Dataset

class mindspore.dataset.**RenderedSST2Dataset** (*dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, decode=False, sampler=None, num_shards=None, shard_id=None, cache=None*)

RenderedSST2 (Rendered Stanford Sentiment Treebank v2) 数据集。

生成的数据集有两列 `[image, label]`。 `image` 列的数据类型为 `uint8`。 `label` 列的数据类型为 `uint32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'val'、'test' 或 'all'。默认值: None, 读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None, 下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: False, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None, 下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `usage` 参数取值不为 'train'、'val'、'test' 或 'all'。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。

- **ValueError** - *shard_id* 参数值错误, 小于 0 或者大于等于 *num_shards* 。

样例:

```
>>> import mindspore.dataset as ds
>>> rendered_sst2_dataset_dir = "/path/to/rendered_sst2_dataset_directory"
>>>
>>> # 1) Read all samples (image files) in rendered_sst2_dataset_dir with 8 threads
>>> dataset = ds.RenderedSST2Dataset(dataset_dir=rendered_sst2_dataset_dir,
...                                 usage="all", num_parallel_workers=8)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器, 其与入参 *sampler* 搭配使用的效果如下。

表 208: 参数 *sampler* 和 *num_samples*, *shuffle*, *num_shards*, *shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray</i> , <i>list</i> , <i>tuple</i> , <i>int</i> 类型	<i>None</i>	<i>/</i>	<i>num_samples</i>	SubsetSampler (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **RenderedSST2** 数据集:

Rendered SST2 是一个图像分类数据集, 它是由 SST2 数据集 中的数据生成的。数据集被分割成三份, 每一份包含有两类 (positive 和 negative): 在 train 这一份下共有 6920 张图像 (3610 张 positive, 3310 张 negative), 在 validation 这一份下共有 872 张图像 (444 张 positive, 428 张 negative), 在 test 这一份下共有 1821 张图像 (909 张 positive, 912 张 negative)。

以下为原始 RenderedSST2 数据集的结构, 您可以将数据集文件解压得到如下的文件结构, 并通过 MindSpore 的 API 进行读取。

```

└─ rendered_sst2_dataset_directory
    └─ train
        └─ negative
            ├── 0001.jpg
            ├── 0002.jpg
            └─ ...
        └─ positive
            ├── 0001.jpg
            ├── 0002.jpg
            └─ ...
    └─ test
        └─ negative
            ├── 0001.jpg
            ├── 0002.jpg
            └─ ...
        └─ positive
            ├── 0001.jpg
            ├── 0002.jpg
            └─ ...
    └─ valid
        └─ negative
            ├── 0001.jpg
            ├── 0002.jpg
            └─ ...
        └─ positive
            ├── 0001.jpg
            ├── 0002.jpg
            └─ ...

```

引用:

```

@inproceedings{socher-etal-2013-recursive,
  title      = {Recursive Deep Models for Semantic Compositionality Over a Sentiment-  

  ↳Treebank},
  author     = {Socher, Richard and Perelygin, Alex and Wu, Jean and Chuang, Jason and   

  ↳Manning, Christopher D. and Ng, Andrew and Potts, Christopher},
  booktitle  = {Proceedings of the 2013 Conference on Empirical Methods in Natural Language-  

  ↳Processing},
  month      = oct,
  year       = {2013},
  address    = {Seattle, Washington, USA},
  publisher  = {Association for Computational Linguistics},
  url        = {https://www.aclweb.org/anthology/D13-1170},
  pages      = {1631--1642},
}

```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.SBDataset

class mindspore.dataset.SBDataset (*dataset_dir, task='Boundaries', usage='all', num_samples=None, num_parallel_workers=1, shuffle=None, decode=None, sampler=None, num_shards=None, shard_id=None*)

SB（Semantic Boundaries）数据集。

通过配置 *task* 参数，生成的数据集具有不同的输出列：

- task* 为 'Boundaries'，有两个输出列：*image* 列的数据类型为 uint8，*label* 列包含 1 个的数据类型为 uint8 的图像。
- task* 为 'Segmentation'，有两个输出列：*image* 列的数据类型为 uint8。*label* 列包含 20 个的数据类型为 uint8 的图像。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录的路径。
- **task** (str, 可选) - 指定读取 SB 数据集的任务类型, 支持 'Boundaries' 和 'Segmentation'。默认值: 'Boundaries'。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'val'、'train_noval' 和 'all'。默认值: 'all'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 所有图像样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作进程数。默认值: 1。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: None, 默认为 False, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。

异常:

- **RuntimeError** - *dataset_dir* 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *dataset_dir* 不存在。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - *task* 不是 'Boundaries' 或 'Segmentation'。
- **ValueError** - *usage* 不是 'train'、'val'、'train_noval' 或 'all'。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, \text{num\_shards})$ 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> sb_dataset_dir = "/path/to/sb_dataset_directory"
>>>
>>> # 1) Get all samples from Semantic Boundaries Dataset in sequence
>>> dataset = ds.SBDataset(dataset_dir=sb_dataset_dir, shuffle=False)
>>>
>>> # 2) Randomly select 350 samples from Semantic Boundaries Dataset
>>> dataset = ds.SBDataset(dataset_dir=sb_dataset_dir, num_samples=350, shuffle=True)
>>>
>>> # 3) Get samples from Semantic Boundaries Dataset for shard 0 in a 2-way distributed_
↪ training
>>> dataset = ds.SBDataset(dataset_dir=sb_dataset_dir, num_shards=2, shard_id=0)
>>>
>>> # In Semantic Boundaries Dataset, each dictionary has keys "image" and "task"
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 215: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **Semantic Boundaries** 数据集：

Semantic Boundaries（语义边界）数据集由 11355 张彩色图像组成。train.txt 中有 8498 个图像，val.txt 中有 2857 个图像，train_noval.txt 中有 5623 个图像。目录 `cls` 中包含类别的分割和边界标注，目录 `inst` 中包含实例级的分割和边界标注。

可以将数据集文件解压缩为以下结构，并通过 MindSpore 的 API 读取：

```
.
├── benchmark_RELEASE
│   ├── dataset
│   ├── img
│   │   ├── 2008_000002.jpg
│   │   ├── 2008_000003.jpg
│   │   └── ...
│   ├── cls
│   │   ├── 2008_000002.mat
│   │   ├── 2008_000003.mat
│   │   └── ...
│   └── inst
```

(下页继续)

(续上页)

```
|      |— 2008_000002.mat
|      |— 2008_000003.mat
|      |— ...
|— train.txt
|— val.txt
```

引用:

```
@InProceedings{BharathICCV2011,
  author      = "Bharath Hariharan and Pablo Arbelaez and Lubomir Bourdev and
                Subhransu Maji and Jitendra Malik",
  title       = "Semantic Contours from Inverse Detectors",
  booktitle   = "International Conference on Computer Vision (ICCV)",
  year        = "2011",
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。

下页继续

表 217 – 续上页

<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.SBUDataset

```
class mindspore.dataset.SBUDataset (dataset_dir, num_samples=None, num_parallel_workers=None, shuffle=None,
                                     decode=False, sampler=None, num_shards=None, shard_id=None,
                                     cache=None)
```

SBU（SBU Captioned Photo）数据集。

生成的数据集有两列：`[image, caption]`。`image` 列的数据类型为 `uint8`。`caption` 列的数据类型为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录的路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，所有图像样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过`mindspore.dataset.config.set_num_parallel_workers()`配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值：False，不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> sbu_dataset_dir = "/path/to/sbu_dataset_directory"
>>> # Read 3 samples from SBU dataset
>>> dataset = ds.SBUDataset(dataset_dir=sbu_dataset_dir, num_samples=3)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 222: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mindspore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>True</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
iterable 类型	<code>int</code>	<code>True</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 SBU 数据集：

SBU 数据集是一个带字幕的大型照片集。它包含一百万张带有视觉相关标注的图像。

你需要使用官方的 `download.m` 手动下载图片，将 `'urls{i}(24, end)'` 替换为 `'urls{i}(24:1:end)'`，并将目录保持如下。

```
.
├── dataset_dir
│   ├── SBU_captioned_photo_dataset_captions.txt
│   ├── SBU_captioned_photo_dataset_urls.txt
│   └── sbu_images
│       ├── m_3326_3596303505_3ce4c20529.jpg
│       ├── .....
│       └── m_2522_4182181099_c3c23ab1cc.jpg
```

引用:

```
@inproceedings{Ordonez:2011:im2text,
  Author      = {Vicente Ordonez and Girish Kulkarni and Tamara L. Berg},
  Title       = {Im2Text: Describing Images Using 1 Million Captioned Photographs},
  Booktitle   = {Neural Information Processing Systems ({NIPS})},
  Year        = {2011},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.SemeionDataset

```
class mindspore.dataset.SemeionDataset (dataset_dir, num_samples=None, num_parallel_workers=None,
                                         shuffle=None, sampler=None, num_shards=None, shard_id=None,
                                         cache=None)
```

Semeion 数据集。

生成的数据集有两列: `[image, label]`。 `image` 列的数据类型为 `uint8`。 `label` 列的数据类型为 `uint32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录的路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: `None`, 所有图像样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: `None`, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: `None`。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: `None`。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: `None`。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: `None`。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: `None`, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> semeion_dataset_dir = "/path/to/semeion_dataset_directory"
>>>
>>> # 1) Get all samples from SEMEION dataset in sequence
>>> dataset = ds.SemeionDataset(dataset_dir=semeion_dataset_dir, shuffle=False)
>>>
>>> # 2) Randomly select 10 samples from SEMEION dataset
>>> dataset = ds.SemeionDataset(dataset_dir=semeion_dataset_dir, num_samples=10,
↪ shuffle=True)
>>>
>>> # 3) Get samples from SEMEION dataset for shard 0 in a 2-way distributed training
>>> dataset = ds.SemeionDataset(dataset_dir=semeion_dataset_dir, num_shards=2, shard_id=0)
>>>
>>> # In SEMEION dataset, each dictionary has keys: image, label.
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 229: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **SEMEION** 数据集：

该数据集由意大利布雷西亚 Tactile Srl 创建 (<http://www.tattil.it>)，并于 1994 年捐赠给意大利罗马 Semeion 通信科学研究中心 (<http://www.semeion.it>)，用于机器学习研究。此数据集由 1593 条样本记录（行）和 256 个属性（列）组成。每条记录代表一个手写数字，最初扫描的分辨率为 256 灰度。数据集拉伸了每个原始扫描图像的每个像素，然后在 0 和 1 之间缩放（将值低于灰度值 127 的每个像素（包括 127）设置为 0，并将灰度值超过 127 的每个像素设置为 1）。最后，每个二进制图像再次缩放为一个 16x16 的方形图像。

```
.
├── semeion_dataset_dir
│   ├── semeion.data
│   └── semeion.names
```

引用：

```
@article{
  title={The Theory of Independent Judges, in Substance Use & Misuse 33(2)1998, pp 439-461},
  author={M Buscema, MetaNet},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.STL10Dataset

```
class mindspore.dataset.STL10Dataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None)
```

STL-10 数据集。

生成的数据集有两列：[image, label]。image 列的数据类型是 uint8。label 列的数据类型是 uint32。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test'、'unlabeled'、'train+unlabeled' 或 'all'。取值为 'train' 时将会读取 5,000 个样本，取值为 'test' 时将

会读取 8,000 个样本, 取值为 'unlabeled' 时将会读取 100,000 个样本, 取值为 'train+unlabeled' 时将会读取 10,5000 个样本, 取值为 'all' 时将会读取全部类型的样本。默认值: None, 读取全部样本图片。

- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `usage` 参数无效。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> stl10_dataset_dir = "/path/to/stl10_dataset_directory"
>>>
>>> # 1) Get all samples from STL10 dataset in sequence
>>> dataset = ds.STL10Dataset(dataset_dir=stl10_dataset_dir, shuffle=False)
>>>
>>> # 2) Randomly select 350 samples from STL10 dataset
>>> dataset = ds.STL10Dataset(dataset_dir=stl10_dataset_dir, num_samples=350, shuffle=True)
>>>
>>> # 3) Get samples from STL10 dataset for shard 0 in a 2-way distributed training
>>> dataset = ds.STL10Dataset(dataset_dir=stl10_dataset_dir, num_shards=2, shard_id=0)
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

说明: 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 236: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **STL10** 数据集：

STL10 数据集由 10 类组成：飞机、鸟、汽车、猫、鹿、狗、马、猴子、船、卡车。数据集样本均为 96x96 的彩色图像。每个类别分别有 500 张训练图像和 800 张测试图像，以及 100000 张没有标签的图像。标签索引从 0 开始标记，没有标签的的图像以-1 作为标记。

以下是原始 STL10 数据集结构。可以将数据集文件解压缩到此目录结构中，并由 MindSpore 的 API 读取。

```
.
├── stl10_dataset_dir
│   ├── train_X.bin
│   ├── train_y.bin
│   ├── test_X.bin
│   ├── test_y.bin
│   └── unlabeled_X.bin
```

引用：

```
@techreport{Coates10,
author      = {Adam Coates},
title       = {Learning multiple layers of features from tiny images},
year        = {20010},
howpublished = {https://cs.stanford.edu/~acoates/stl10/},
description  = {The STL-10 dataset consists of 96x96 RGB images in 10 classes,
                with 500 training images and 800 testing images per class.
                There are 5000 training images and 8000 test images.
                It also has 100000 unlabeled images for unsupervised learning.
                These examples are extracted from a similar but broader distribution of
                images.
            }
```

(下页继续)

(续上页)

```
        }  
    }
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩，多个 dataset 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

`mindspore.dataset.SUN397Dataset`

```
class mindspore.dataset.SUN397Dataset (dataset_dir, num_samples=None, num_parallel_workers=None,
                                         shuffle=None, decode=False, sampler=None, num_shards=None,
                                         shard_id=None, cache=None)
```

SUN397 (Scene UNDERstanding) 数据集。

生成的数据集有两列: `[image, label]`。 `image` 列的数据类型是 `uint8`。 `label` 列的数据类型是 `uint32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: `None`, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: `None`, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: `None`, 下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: `False`, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: `None`, 下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: `None`。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: `None`。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: `None`, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> sun397_dataset_dir = "/path/to/sun397_dataset_directory"
>>>
>>> # 1) Read all samples (image files) in sun397_dataset_dir with 8 threads
>>> dataset = ds.SUN397Dataset(dataset_dir=sun397_dataset_dir, num_parallel_workers=8)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 243: 参数 `sampler` 和 `num_samples`、`shuffle`、`num_shards`、`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **SUN397** 数据集：

SUN397 是一个用于场景识别的数据集，包括 397 个类别，有 108,754 张图像。不同类别的图像数量不同，但每个类别至少有 100 张。图片为 `jpg`、`png` 或 `gif` 格式。

以下是原始 SUN397 数据集结构。可以将数据集文件解压缩到此目录结构中，并由 MindSpore 的 API 读取。

```
.
├── sun397_dataset_directory
│   ├── ClassName.txt
│   ├── README.txt
│   ├── a
│   │   ├── abbey
│   │   │   ├── sun_aaaulhwrhqgejnyt.jpg
│   │   │   ├── sun_aacphuqehdodwawg.jpg
│   │   │   └── ...
│   │   ├── apartment_building
│   │   │   └── outdoor
│   │   │       ├── sun_aamyhslnsnomjzue.jpg
│   │   │       ├── sun_abbjzfrsalhqivis.jpg
│   │   │       └── ...
│   │   └── ...
│   └── b
│       └── badlands
```

(下页继续)

(续上页)

```

├── sun_aabtemlmesogqbbp.jpg
├── sun_afbsfeexggdhzshd.jpg
├── ...
├── balcony
│   ├── exterior
│   │   ├── sun_aaxzaiuznwquburq.jpg
│   │   ├── sun_baajuldidvlyzhv.jpg
│   │   └── ...
│   └── interior
│       ├── sun_babkzjntjfarengi.jpg
│       ├── sun_bagjvjynskmonnbv.jpg
│       └── ...
└── ...

```

引用:

```
@inproceedings{xiao2010sun,
title      = {Sun database: Large-scale scene recognition from abbey to zoo},
author     = {Xiao, Jianxiong and Hays, James and Ehinger, Krista A and Oliva, Aude and
Torralba, Antonio},
booktitle  = {2010 IEEE computer society conference on computer vision and pattern
recognition},
pages      = {3485--3492},
year       = {2010},
organization = {IEEE}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 epoch 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩，多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.SVHNDataset

class mindspore.dataset.**SVHNDataset** (*dataset_dir, usage=None, num_samples=None, num_parallel_workers=1, shuffle=None, sampler=None, num_shards=None, shard_id=None*)

SVHN (Street View House Numbers) 数据集。

生成的数据集有两列: `[image, label]`。`image` 列的数据类型是 `uint8`。`label` 列的数据类型是 `uint32`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test'、'extra' 或 'all'。默认值: None, 读取全部样本图片。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数, 可以小于数据集总数。默认值: None, 读取全部样本图片。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作进程数。默认值: 1。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `usage` 参数无效。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> svhn_dataset_dir = "/path/to/svhn_dataset_directory"
>>> dataset = ds.SVHNDataset(dataset_dir=svhn_dataset_dir, usage="train")
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 250: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>DistributedSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IterableSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **SVHN** 数据集：

SVHN 数据集是从谷歌街景图像中的门牌号码中获得的，由 10 位数字组成。

以下是原始 **SVHN** 数据集结构。可以将数据集文件解压缩到此目录结构中，并由 **MindSpore** 的 API 读取。

```
.
├── svhn_dataset_dir
│   ├── train_32x32.mat
│   ├── test_32x32.mat
│   └── extra_32x32.mat
```

引用：

```
@article{
  title={Reading Digits in Natural Images with Unsupervised Feature Learning},
  author={Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng},
  conference={NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.},
  year={2011},
  publisher={NIPS}
```

(下页继续)

(续上页)

```
url={http://ufldl.stanford.edu/housenumbers}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩，多个 dataset 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.USPSDataset

class mindspore.dataset.USPSDataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, cache=None)

USPS (U.S. Postal Service) 数据集。

生成的数据集有两列: [image, label]。image 列的数据类型为 uint8。label 列的数据类型为 uint32。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 7,291 个样本, 取值为 'test' 时将会读取 2,007 个测试样本, 取值为 'all' 时将会读取全部 9,298 个样本。默认值: None, 读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取所有数据集。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, Shuffle], 可选) - 每个 epoch 中数据混洗的模式, 支持传入 bool 类型与枚举类型进行指定。默认值: Shuffle.GLOBAL。如果 shuffle 为 False, 则不混洗, 如果 shuffle 为 True, 等同于将 shuffle 设置为 mindspore.dataset.Shuffle.GLOBAL。通过传入枚举变量设置数据混洗的模式:
 - Shuffle.GLOBAL: 混洗文件和样本。
 - Shuffle.FILES: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, num_samples 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 num_shards 时才能指定此参数。
- **cache** (DatasetCache, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - dataset_dir 路径下不包含数据文件。
- **RuntimeError** - 指定了 num_shards 参数, 但是未指定 shard_id 参数。
- **RuntimeError** - 指定了 shard_id 参数, 但是未指定 num_shards 参数。
- **ValueError** - usage 参数无效。
- **ValueError** - num_parallel_workers 参数超过系统最大线程数。
- **ValueError** - 如果 shard_id 取值不在 [0, num_shards) 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> usps_dataset_dir = "/path/to/usps_dataset_directory"
>>>
>>> # Read 3 samples from USPS dataset
>>> dataset = ds.USPSDataset(dataset_dir=usps_dataset_dir, num_samples=3)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

关于 USPS 数据集：

USPS 是美国邮政服务公司从信封中自动扫描的数字数据集，包含总共 9,298 个 16×16 像素灰度样本。数据集中的图片内容已被预处理为居中和归一化，并集中了多种样式的字体。

以下是原始的 USPS 数据集结构。可以将数据集文件下载并解压缩到此目录结构中，并通过 MindSpore 的 API 读取。

```
.
└─ usps_dataset_dir
    ├── usps
    └── usps.t
```

引用：

```
@article{hull1994database,
  title={A database for handwritten text recognition research},
  author={Hull, Jonathan J.},
  journal={IEEE Transactions on pattern analysis and machine intelligence},
  volume={16},
  number={5},
  pages={550--554},
  year={1994},
  publisher={IEEE}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.VOCDataset

class mindspore.dataset.VOCDataset (*dataset_dir, task='Segmentation', usage='train', class_indexing=None, num_samples=None, num_parallel_workers=None, shuffle=None, decode=False, sampler=None, num_shards=None, shard_id=None, cache=None, extra_metadata=False, decrypt=None*)

VOC (Visual Object Classes) 数据集。

根据给定的 *task* 配置, 生成数据集具有不同的输出列:

- *task* 为 'Detection', 输出列: `[image, dtype=uint8], [bbox, dtype=float32], [label, dtype=uint32], [difficult, dtype=uint32], [truncate, dtype=uint32]`。
- *task* 为 'Segmentation', 输出列: `[image, dtype=uint8], [target, dtype=uint8]`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录的路径。
- **task** (str, 可选) - 指定读取 VOC 数据的任务类型, 现在只支持 'Segmentation' 和 'Detection'。默认值: 'Segmentation'。
- **usage** (str, 可选) - 指定数据集的子集。默认值: 'train'。
 - 如果 *task* 的值为 'Segmentation', 则读取 'ImageSets/Segmentation/' 目录下定义的图片 and label 信息;
 - 如果 *task* 的值为 'Detection', 则读取 'ImageSets/Main/' 目录下定义的图片 and label 信息。
- **class_indexing** (dict, 可选) - 指定一个从 label 名称到 label 索引的映射, 要求映射规则为 string 到 int。索引值从 0 开始, 并且要求每个 label 名称对应的索引值唯一。仅在 'Detection' 任务中有效。默认值: None, 不指定。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 所有图像样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: False, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

- **extra_metadata** (bool, 可选) - 用于指定是否额外输出一个数据列用于表示图片元信息。如果为 True, 则将额外输出一个名为 `[_meta-filename, dtype=string]` 的数据列。默认值: False。
- **decrypt** (callable, 可选) - 图像解密函数, 接受加密的图片路径并返回 bytes 类型的解密数据。默认值: None, 不进行解密。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 读取的 xml 文件格式异常或无效。
- **RuntimeError** - 读取的 xml 文件缺失 `object` 属性。
- **RuntimeError** - 读取的 xml 文件缺失 `bndbox` 属性。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 指定的任务不为 'Segmentation' 或 'Detection'。
- **ValueError** - 指定任务为 'Segmentation' 时, `class_indexing` 参数不为 None。
- **ValueError** - 与 `usage` 参数相关的 txt 文件不存在。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> voc_dataset_dir = "/path/to/voc_dataset_directory"
>>>
>>> # 1) Read VOC data for segmentation training
>>> dataset = ds.VOCDataset(dataset_dir=voc_dataset_dir, task="Segmentation", usage="train")
>>>
>>> # 2) Read VOC data for detection training
>>> dataset = ds.VOCDataset(dataset_dir=voc_dataset_dir, task="Detection", usage="train")
>>>
>>> # 3) Read all VOC dataset samples in voc_dataset_dir with 8 threads in random order
>>> dataset = ds.VOCDataset(dataset_dir=voc_dataset_dir, task="Detection", usage="train",
...                          num_parallel_workers=8)
>>>
>>> # 4) Read then decode all VOC dataset samples in voc_dataset_dir in sequence
>>> dataset = ds.VOCDataset(dataset_dir=voc_dataset_dir, task="Detection", usage="train",
...                          decode=True, shuffle=False)
>>>
>>> # In VOC dataset, if task='Segmentation', each dictionary has keys "image" and "target"
>>> # In VOC dataset, if task='Detection', each dictionary has keys "image" and "annotation"
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明:

- 当参数 `extra_metadata` 为 `True` 时, 还需使用 `rename` 操作删除额外数据列 `'_meta-filename'` 的前缀 `'_meta-'`, 否则迭代得到的数据行中不会出现此额外数据列。
- 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器, 其与入参 `sampler` 搭配使用的效果如下。

表 263: 参数 `sampler` 和 `num_samples`, `shuffle`, `num_shards`, `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **VOC** 数据集：

PASCAL Visual Object Classes（VOC）是视觉目标识别和检测的挑战赛，它为视觉和机器学习社区提供了图像和标注的标准数据集，称为 **VOC** 数据集。

您可以解压缩原始 **VOC-2012** 数据集文件到如下目录结构，并通过 **MindSpore** 的 **API** 进行读取。

```
.
├── voc2012_dataset_dir
│   ├── Annotations
│   │   ├── 2007_000027.xml
│   │   ├── 2007_000032.xml
│   │   └── ...
│   ├── ImageSets
│   │   ├── Action
│   │   ├── Layout
│   │   ├── Main
│   │   └── Segmentation
│   ├── JPEGImages
│   │   ├── 2007_000027.jpg
│   │   ├── 2007_000032.jpg
│   │   └── ...
```

(下页继续)

(续上页)

```
├─ SegmentationClass
│   ├── 2007_000032.png
│   ├── 2007_000033.png
│   └── ...
└─ SegmentationObject
    ├── 2007_000032.png
    ├── 2007_000033.png
    └── ...
```

引用:

```
@article{Everingham10,
author      = {Everingham, M. and Van~Gool, L. and Williams, C. K. I. and Winn, J. and
↪Zisserman, A.},
title       = {The Pascal Visual Object Classes (VOC) Challenge},
journal     = {International Journal of Computer Vision},
volume      = {88},
year        = {2012},
number      = {2},
month       = {jun},
pages       = {303--338},
biburl      = {http://host.robots.ox.ac.uk/pascal/VOC/pubs/everingham10.html#bibtex},
howpublished = {http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.WIDERFaceDataset

```
class mindspore.dataset.WIDERFaceDataset (dataset_dir, usage=None, num_samples=None,
                                           num_parallel_workers=None, shuffle=None, decode=False,
                                           sampler=None, num_shards=None, shard_id=None, cache=None)
```

WIDERFace 数据集。

当 `usage` 为“train”、“valid”或“all”时, 生成的数据集有八列 `["image", "bbox", "blur", "expression", "illumination", "occlusion", "pose", "invalid"]`。其中 `image` 列的数据类型为 `uint8`, 其他列均为 `uint32`。当 `usage` 为“test”时, 生成的数据集只有一列 `["image"]`, 数据类型为 `uint8`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test'、'valid' 或 'all'。取值为 'train' 时将会读取 12,880 个样本, 取值为 'test' 时将会读取 16,097 个样本, 取值为 'valid' 时将会读取 3,226 个样本, 取值为 'all' 时将会读取全部类别样本。默认值: None, 读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **decode** (bool, 可选) - 是否对读取的图片进行解码操作。默认值: False, 不解码。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 数据并行模式训练 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 单节点数据缓存。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

- **ValueError** - *usage* 不为 'train'、'test'、'valid' 或 'all'。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - *annotation_file* 不存在。
- **ValueError** - *dataset_dir* 不存在。

样例:

```
>>> import mindspore.dataset as ds
>>> wider_face_dir = "/path/to/wider_face_dataset"
>>>
>>> # Read 3 samples from WIDERFace dataset
>>> dataset = ds.WIDERFaceDataset(dataset_dir=wider_face_dir, num_samples=3)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明: 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器, 其与入参 *sampler* 搭配使用的效果如下。

表 270: 参数 *sampler* 和 *num_samples* , *shuffle* , *num_shards* , *shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<i>mind-spore.dataset.Sampler</i> 类型	<i>None</i>	<i>None</i>	<i>None</i>	sampler
<i>numpy.ndarray</i> , <i>list</i> , <i>tuple</i> , <i>int</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>indices</i> = sampler , <i>num_samples</i> = num_samples)
<i>iterable</i> 类型	<i>/</i>	<i>/</i>	<i>num_samples</i>	IndexSampler (<i>sampler</i> = sampler , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>num_shards</i> / <i>shard_id</i>	<i>False</i>	<i>num_samples</i>	DistributedSampler (<i>num_shards</i> = num_shards , <i>shard_id</i> = shard_id , <i>shuffle</i> = False , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>None</i>	RandomSampler (<i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>None</i> / <i>True</i>	<i>num_samples</i>	RandomSampler (<i>replacement</i> = True , <i>num_samples</i> = num_samples)
<i>None</i>	<i>None</i>	<i>False</i>	<i>num_samples</i>	SequentialSampler (<i>num_samples</i> = num_samples)

关于 **WIDERFace** 数据集:

WIDER FACE 数据集具有 12,880 个训练样本, 16,097 个测试样本, 以及 3,226 个验证样本。此数据集是 **WIDER** 数据集的子集。其中图片已经预先进行了尺寸归一化和人像中心化处理。

以下是原始的 **WIDERFace** 数据集结构。可以将数据集文件解压缩到此目录结构中, 并由 **MindSpore** 的 API 读取。



引用:

```
@inproceedings{2016WIDER,
  title={WIDERFACE: A Detection Benchmark},
  author={Yang, S. and Luo, P. and Loy, C. C. and Tang, X.},
  booktitle={IEEE},
  pages={5525-5533},
  year={2016},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。

下页继续

表 271 – 续上页

<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

12.5.2 文本数据集

<code>mindspore.dataset.AGNewsDataset</code>	AG News 数据集。
<code>mindspore.dataset.AmazonReviewDataset</code>	Amazon Review Full 和 Amazon Review Polarity 数据集。
<code>mindspore.dataset.CLUEDataset</code>	CLUE (Chinese Language Understanding Evaluation) 数据集。
<code>mindspore.dataset.CSVDataset</code>	CSV 文件数据集。
<code>mindspore.dataset.CoNLL2000Dataset</code>	CoNLL-2000 (Conference on Computational Natural Language Learning) 分块数据集。
<code>mindspore.dataset.DBpediaDataset</code>	DBpedia 数据集。
<code>mindspore.dataset.EnWik9Dataset</code>	EnWik9 数据集。
<code>mindspore.dataset.IMDBDataset</code>	IMDb (Internet Movie Database) 数据集。
<code>mindspore.dataset.IWSLT2016Dataset</code>	IWSLT2016 (International Workshop on Spoken Language Translation) 数据集。
<code>mindspore.dataset.IWSLT2017Dataset</code>	IWSLT2017 (International Workshop on Spoken Language Translation) 数据集。
<code>mindspore.dataset.Multi30kDataset</code>	Multi30k 数据集。
<code>mindspore.dataset.PennTreebankDataset</code>	PennTreebank 数据集。
<code>mindspore.dataset.SogouNewsDataset</code>	SogouNews 数据集。
<code>mindspore.dataset.SQuADDataset</code>	SQuAD 1.1 和 SQuAD 2.0 数据集。
<code>mindspore.dataset.SST2Dataset</code>	SST2 (Stanford Sentiment Treebank v2) 数据集。
<code>mindspore.dataset.TextFileDataset</code>	读取和解析文本文件构建数据集。
<code>mindspore.dataset.UDPOSDataset</code>	UDPOS (Universal Dependencies dataset for Part of Speech) 数据集。
<code>mindspore.dataset.WikiTextDataset</code>	WikiText2 和 WikiText103 数据集。
<code>mindspore.dataset.YahooAnswersDataset</code>	YahooAnswers 数据集。
<code>mindspore.dataset.YelpReviewDataset</code>	Yelp Review Full 和 Yelp Review Polarity 数据集。

mindspore.dataset.AGNewsDataset

```
class mindspore.dataset.AGNewsDataset (dataset_dir, usage=None, num_samples=None,
                                         num_parallel_workers=None, shuffle=Shuffle.GLOBAL, num_shards=None,
                                         shard_id=None, cache=None)
```

AG News 数据集。

生成的数据集有三列 `[index, title, description]`，三列的数据类型均为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。默认值：None，读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, [Shuffle](#)], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：Shuffle.GLOBAL。如果 `shuffle` 为 False，则不混洗，如果 `shuffle` 为 True，等同于将 `shuffle` 设置为 Shuffle.GLOBAL。通过传入枚举变量设置数据混洗的模式：
 - Shuffle.GLOBAL：混洗文件和样本。
 - Shuffle.FILES：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。

样例：

```
>>> import mindspore.dataset as ds
>>> ag_news_dataset_dir = "/path/to/ag_news_dataset_file"
>>> dataset = ds.AGNewsDataset(dataset_dir=ag_news_dataset_dir, usage='all')
```

教程样例：

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 AG News 数据集：

AG News 是一个大型合集，具有超过 100 万篇新闻文章。这些新闻文章是由 ComeToMyHead 在持续 1 年多的活动中，从 2000 多个新闻来源收集的。ComeToMyHead 是一个学术新闻搜索引擎，自 2004 年 7 月以来一直在运营。数据集由学者提供，用于研究目的，如数据挖掘（聚类、分类等）、信息检索（排名、搜索等）、xml、数据压缩、数据流和

任何其他非商业活动。AG 的新闻主题类别来自于原始语料库中四个最大的类别。每个分类包含 30000 个训练样本和 1900 个测试样本。`train.csv` 中的训练样本总数为 12 万，`test.csv` 中的测试样本总数为 7600。

可以将数据集文件解压缩到以下结构中，并通过 MindSpore 的 API 读取：

```
.
├── ag_news_dataset_dir
│   ├── classes.txt
│   ├── train.csv
│   ├── test.csv
│   └── readme.txt
```

引用：

```
@misc{zhang2015characterlevel,
title={Character-level Convolutional Networks for Text Classification},
author={Xiang Zhang and Junbo Zhao and Yann LeCun},
year={2015},
eprint={1509.01626},
archivePrefix={arXiv},
primaryClass={cs.LG}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.AmazonReviewDataset

```
class mindspore.dataset.AmazonReviewDataset (dataset_dir, usage=None, num_samples=None,
                                              num_parallel_workers=None, shuffle=Shuffle.GLOBAL,
                                              num_shards=None, shard_id=None, cache=None)
```

Amazon Review Full 和 Amazon Review Polarity 数据集。
生成的数据集有三列 [label, title, content]，三列的数据类型均为 string。
参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。对于 Polarity 数据集，'train' 将读取 360 万个训练样本，'test' 将读取 40 万个测试样本，'all' 将读取所有 400 万个样本。对于 Full 数据集，'train' 将读取 300 万个训练样本，'test' 将读取 65 万个测试样本，'all' 将读取所有 365 万个样本。默认值：None，读取所有样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：Shuffle.GLOBAL。如果 *shuffle* 为 False，则不混洗，如果 *shuffle* 为 True，等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式：
 - Shuffle.GLOBAL：混洗文件和样本。
 - Shuffle.FILES：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 数据并行模式训练 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 单节点数据缓存。默认值：None，不使用缓存。

异常：

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例：

```
>>> import mindspore.dataset as ds
>>> amazon_review_dataset_dir = "/path/to/amazon_review_dataset_dir"
>>> dataset = ds.AmazonReviewDataset(dataset_dir=amazon_review_dataset_dir, usage='all')
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

关于 **AmazonReview** 数据集：

Amazon Review Full 数据集包括来自亚马逊的评论数据。这些数据跨越 18 年，包括截止至 2013 年 3 月的约 3500 万条评论。评论数据包括产品和用户信息、产品评级和产品评论。数据集主要用于文本分类，给定内容和标题，预测正确的星级评定。

Amazon Review Polarity 数据集对产品评分进行了分级，评论分数 1 和 2 视为负面评论，4 和 5 视为正面评论。评分 3 的样本则被忽略。

Amazon Reviews Polarity 和 Amazon Reviews Full datasets 具有相同的目录结构。可以将数据集文件解压缩到以下结构，并通过 MindSpore 的 API 读取：

```
.
├── amazon_review_dir
│   ├── train.csv
│   ├── test.csv
│   └── readme.txt
```

引用：

```
@article{zhang2015character,
  title={Character-level convolutional networks for text classification},
  author={Zhang, Xiang and Zhao, Junbo and LeCun, Yann},
  journal={Advances in neural information processing systems},
  volume={28},
  pages={649--657},
  year={2015}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 epoch 的数据集对象。

下页继续

表 284 – 续上页

<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <code>buffer_size</code> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <code>count</code> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <code>dataset</code> 对象按列进行合并压缩, 多个 <code>dataset</code> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <code>batch_size</code> 条数据组合为一个批数据, 并可通过可选参数 <code>per_batch_map</code> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <code>batch_size</code> 条数据组合为一个批数据, 并可通过可选参数 <code>pad_info</code> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <code>epoch</code> 中的 <code>batch</code> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <code>RepeatDataset</code> 中定义的 <code>repeat</code> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <code>shape</code> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.CLUEDataset

```
class mindspore.dataset.CLUEDataset (dataset_files, task='AFQMC', usage='train', num_samples=None,
                                     num_parallel_workers=None, shuffle=Shuffle.GLOBAL, num_shards=None,
                                     shard_id=None, cache=None)
```

CLUE（Chinese Language Understanding Evaluation）数据集。

目前支持的 CLUE 分类任务包括：'AFQMC'、'TNEWS'、'IFLYTEK'、'CMNLI'、'WSC' 和 'CSL'。更多 CLUE 数据集的说明详见 [CLUE GitHub](#)。

参数：

- **dataset_files** (Union[str, list[str]]) - 数据集文件路径，支持单文件路径字符串、多文件路径字符串列表或可被 glob 库模式匹配的字符串，文件列表将在内部进行字典排序。
- **task** (str, 可选) - 任务类型，可取值为 'AFQMC'、'TNEWS'、'IFLYTEK'、'CMNLI'、'WSC' 或 'CSL'。默认值：'AFQMC'。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'eval'。默认值：'train'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：*Shuffle.GLOBAL*。如果 *shuffle* 为 False，则不混洗，如果 *shuffle* 为 True，等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式：
 - *Shuffle.GLOBAL*：混洗文件和样本。
 - *Shuffle.FILES*：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。

- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

根据给定的 *task* 参数和 *usage* 配置，数据集会生成不同的输出列：

<i>task</i>	<i>usage</i>	输出列
AFQMC	train	[sentence1, dtype=string] [sentence2, dtype=string] [label, dtype=string]
	test	[id, dtype=uint32] [sentence1, dtype=string] [sentence2, dtype=string]
	eval	[sentence1, dtype=string] [sentence2, dtype=string] [label, dtype=string]
TNEWS	train	[label, dtype=string] [label_des, dtype=string] [sentence, dtype=string] [keywords, dtype=string]
	test	[label, dtype=uint32] [keywords, dtype=string] [sentence, dtype=string]
	eval	[label, dtype=string] [label_des, dtype=string] [sentence, dtype=string] [keywords, dtype=string]
IFLY-TEK	train	[label, dtype=string] [label_des, dtype=string] [sentence, dtype=string]
	test	[id, dtype=uint32] [sentence, dtype=string]
	eval	[label, dtype=string] [label_des, dtype=string] [sentence, dtype=string]
CMNLI	train	[sentence1, dtype=string] [sentence2, dtype=string] [label, dtype=string]
	test	[id, dtype=uint32] [sentence1, dtype=string] [sentence2, dtype=string]
	eval	[sentence1, dtype=string] [sentence2, dtype=string] [label, dtype=string]
WSC	train	[span1_index, dtype=uint32] [span2_index, dtype=uint32] [span1_text, dtype=string] [span2_text, dtype=string] [idx, dtype=uint32] [text, dtype=string] [label, dtype=string]
	test	[span1_index, dtype=uint32] [span2_index, dtype=uint32] [span1_text, dtype=string] [span2_text, dtype=string] [idx, dtype=uint32] [text, dtype=string]
	eval	[span1_index, dtype=uint32] [span2_index, dtype=uint32] [span1_text, dtype=string]
1780		[span2_text, dtype=string] [idx, dtype=uint32] [text, dtype=string] [label, dtype=string]

异常:

- **ValueError** - `dataset_files` 参数所指向的文件无效或不存在。
- **ValueError** - `task` 参数不为 'AFQMC'、'TNEWS'、'IFLYTEK'、'CMNLI'、'WSC' 或 'CSL'。
- **ValueError** - `usage` 参数不为 'train'、'test' 或 'eval'。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。

样例:

```
>>> import mindspore.dataset as ds
>>> clue_dataset_dir = ["/path/to/clue_dataset_file"] # contains 1 or multiple clue files
>>> dataset = ds.CLUEDataset(dataset_files=clue_dataset_dir, task='AFQMC', usage='train')
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

关于 CLUE 数据集:

CLUE, 又名中文语言理解测评基准, 包含许多有代表性的数据集, 涵盖单句分类、句对分类和机器阅读理解等任务。您可以将数据集解压成如下的文件结构, 并通过 MindSpore 的 API 进行读取, 以 'afqmc' 数据集为例:

```
.
├── afqmc_public
│   ├── train.json
│   ├── test.json
│   └── dev.json
```

引用:

```
@article{CLUEbenchmark,
title = {CLUE: A Chinese Language Understanding Evaluation Benchmark},
author = {Liang Xu, Xuanwei Zhang, Lu Li, Hai Hu, Chenjie Cao, Weitang Liu, Junyi Li,
→Yudong Li,
→Kai Sun, Yechen Xu, Yiming Cui, Cong Yu, Qianqian Dong, Yin Tian, Dian Yu, Bo Shi,
→Jun Zeng,
→Rongzhao Wang, Weijian Xie, Yanting Li, Yina Patterson, Zuoyu Tian, Yiwen Zhang, He
→Zhou,
→Shaowei Hua Liu, Qipeng Zhao, Cong Yue, Xinrui Zhang, Zhengliang Yang, Zhenzhong Lan},
journal = {arXiv preprint arXiv:2004.05986},
year = {2020},
howpublished = {https://github.com/CLUEbenchmark/CLUE}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.CSVDataset

class mindspore.dataset.CSVDataset (*dataset_files, field_delim=',', column_defaults=None, column_names=None, num_samples=None, num_parallel_workers=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, cache=None*)

CSV 文件数据集。

生成的数据集的列名和列类型取决于输入的 CSV 文件。

参数：

- dataset_files** (Union[str, list[str]]) - 数据集文件路径，支持单文件路径字符串、多文件路径字符串列表或可被 glob 库模式匹配的字符串，文件列表将在内部进行字典排序。
- field_delim** (str, 可选) - 指定用于分隔字段的分隔符。默认值：','。

- **column_defaults** (list, 可选) - 指定每个数据列的数据类型, 有效的类型包括 float、int 或 string。默认值: None, 不指定。如果未指定该参数, 则所有列的数据类型将被视为 string。
- **column_names** (list[str], 可选) - 指定数据集生成的列名。默认值: None, 不指定。如果未指定该列表, 则将 CSV 文件首行提供的字段作为列名生成。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式, 支持传入 bool 类型与枚举类型进行指定。默认值: *Shuffle*.GLOBAL。如果 *shuffle* 为 False, 则不混洗, 如果 *shuffle* 为 True, 等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式:
 - *Shuffle*.GLOBAL: 混洗文件和文件中的数据。
 - *Shuffle*.FILES: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_files* 参数所指向的文件无效或不存在。
- **ValueError** - *field_delim* 参数无效。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> csv_dataset_dir = ["/path/to/csv_dataset_file"] # contains 1 or multiple csv files
>>> dataset = ds.CSVDataset(dataset_files=csv_dataset_dir, column_names=['col1', 'col2',
↪ 'col3', 'col4'])
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.CoNLL2000Dataset

class mindspore.dataset.CoNLL2000Dataset (*dataset_dir, usage=None, num_samples=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, num_parallel_workers=None, cache=None*)

CoNLL-2000（Conference on Computational Natural Language Learning）分块数据集。

生成的数据集有三列 `[word, pos_tag, chunk_tag]`。三列的数据类型均为 `string`。

参数：

- dataset_dir** (str) - 包含 CoNLL2000 分块数据集文件的根目录路径。
- usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。'train' 将读取 8936 个训练样本，'test' 将读取 2,012 个测试样本中，'all' 将读取所有 1,0948 个样本。默认值：None，读取

全部样本。

- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: `None` , 读取所有样本。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 `epoch` 中数据混洗的模式。默认值: `Shuffle.GLOBAL` 。如果 *shuffle* 为 `False` , 则不混洗。如果 *shuffle* 为 `True` , 执行全局混洗。总共有三种混洗模式, 通过枚举变量 *mindspore.dataset.Shuffle* 指定。
 - `Shuffle.GLOBAL` : 混洗文件和样本, 与设置为 `True` 效果相同。
 - `Shuffle.FILES` : 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。指定此参数后, *num_samples* 表示每个分片的最大样本数。默认值: `None` 。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。只有当指定了 *num_shards* 时才能指定此参数。默认值: `None` 。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: `None` , 使用全局默认线程数 (8), 也可以通过 *mindspore.dataset.config.set_num_parallel_workers()* 配置全局线程数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#) 。默认值: `None` , 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。

样例:

```
>>> import mindspore.dataset as ds
>>> conll2000_dataset_dir = "/path/to/conll2000_dataset_dir"
>>> dataset = ds.CoNLL2000Dataset(dataset_dir=conll2000_dataset_dir, usage='all')
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 **CoNLL2000** 数据集:

CoNLL2000 分块数据集由华尔街日报语料库第 15-20 节的文本组成。文本使用 IOB 表示法进行分块, 分块类型有 NP, VP, PP, ADJP 和 ADVP。数据集由通过空格分隔的三列组成。第一列包含当前单词, 第二列是由 Brill 标注器派生的词性标注, 第三列是由华尔街语料库派生的分块标注。文本分块旨在将文本划分为单词的句法的相关组成部分。

可以将数据集文件解压缩到以下结构, 并通过 MindSpore 的 API 读取:

```
.
├── conll2000_dataset_dir
│   ├── train.txt
│   ├── test.txt
│   └── readme.txt
```

引用:

```
@inproceedings{tksbuchholz2000conll,
  author      = {Tjong Kim Sang, Erik F. and Sabine Buchholz},
  title       = {Introduction to the CoNLL-2000 Shared Task: Chunking},
  editor      = {Claire Cardie and Walter Daelemans and Claire Nedellec and Tjong Kim Sang, and Erik},
  booktitle   = {Proceedings of CoNLL-2000 and LLL-2000},
  publisher   = {Lisbon, Portugal},
  pages       = {127--132},
  year        = {2000}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.DBpediaDataset

```
class mindspore.dataset.DBpediaDataset (dataset_dir, usage=None, num_samples=None,
                                         num_parallel_workers=None, shuffle=Shuffle.GLOBAL,
                                         num_shards=None, shard_id=None, cache=None)
```

DBpedia 数据集。

生成的数据集有三列 `[class, title, content]`，三列的数据类型均为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'，'test' 或 'all'。'train' 将读取 560,000 个训练样本，'test' 将读取 70,000 个测试样本中，'all' 将读取所有 630,000 个样本。默认值：None，读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, [Shuffle](#)], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：Shuffle.GLOBAL。如果 `shuffle` 为 False，则不混洗，如果 `shuffle` 为 True，等同于将 `shuffle` 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式：
 - `Shuffle.GLOBAL`：混洗文件和样本。
 - `Shuffle.FILES`：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** ([DatasetCache](#), 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> dbpedia_dataset_dir = "/path/to/dbpedia_dataset_directory"
>>>
>>> # 1) Read 3 samples from DBpedia dataset
>>> dataset = ds.DBpediaDataset(dataset_dir=dbpedia_dataset_dir, num_samples=3)
>>>
>>> # 2) Read train samples from DBpedia dataset
>>> dataset = ds.DBpediaDataset(dataset_dir=dbpedia_dataset_dir, usage="train")
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

关于 DBpedia 数据集：

DBpedia 数据集包括 14 个类，超过 63 万个文本样本，train.csv 中有 56 万样本，test.csv 中有 7 万测试样本。14 个不同的类别分别是：公司、教育学院、艺术家、运动员、文员，交通，建筑，自然场所，村庄，动物，植物，专辑，电影，书面工作。

以下是原始 DBpedia 数据集结构。可以将数据集文件解压缩到此目录结构中，并通过 Mindspore 的 API 读取。

```
.
├── dbpedia_dataset_dir
│   ├── train.csv
│   ├── test.csv
│   ├── classes.txt
│   └── readme.txt
```

引用：

```
@article{DBpedia,
title   = {DBpedia Ontology Classification Dataset},
author  = {Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas,
          Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef,
          Sören Auer, Christian Bizer},
year    = {2015},
howpublished = {http://dbpedia.org}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.EnWik9Dataset

class mindspore.dataset.**EnWik9Dataset** (*dataset_dir, num_samples=None, num_parallel_workers=None, shuffle=True, num_shards=None, shard_id=None, cache=None*)

EnWik9 数据集。

生成的数据集有一列 *[text]*, 数据类型为 `string`。

参数:

- **dataset_dir** (`str`) - 包含数据集文件的根目录路径。
- **num_samples** (`int`, 可选) - 指定从数据集中读取的样本数。默认值: `None`, 读取所有样本。
- **num_parallel_workers** (`int`, 可选) - 指定读取数据的工作线程数。默认值: `None`, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (`Union[bool, Shuffle]`, 可选) - 每个 `epoch` 中数据混洗的模式, 支持传入 `bool` 类型与枚举类型进行指定。默认值: `True`。如果 *shuffle* 为 `False`, 则不混洗, 如果 *shuffle* 为 `True`, 等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式:
 - `Shuffle.GLOBAL`: 混洗文件和样本。
 - `Shuffle.FILES`: 仅混洗文件。
- **num_shards** (`int`, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: `None`。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (`int`, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: `None`。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: `None`, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> en_wik9_dataset_dir = "/path/to/en_wik9_dataset"
>>> dataset2 = ds.EnWik9Dataset(dataset_dir=en_wik9_dataset_dir, num_samples=2,
...                             shuffle=True)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

关于 **EnWik9** 数据集：

EnWik9 的数据是一系列 UTF-8 编码的 XML，主要由英文文本组成。数据集包含 243,426 篇文章标题，其中 85,560 个被重定向以修复丢失的网页链接，其余是常规文章。

数据是 UTF-8 格式。所有字符都在 U'0000 到 U'10FFFF 范围内，有效编码为 1 到 4 字节。字节值 0xC0、0xC1 和 0xF5-0xFF 从未出现。此外，在维基百科转储中，除了 0x09（制表符）和 0x0A（换行符）外，没有范围为 0x00-0x1F 的控制字符。断行符只出现在段落边界上，因此整体是有语义目的。

可以将数据集文件解压缩到以下目录结构中，并由 MindSpore 的 API 读取。

```
.
└─ EnWik9
    └─ enwik9
```

引用：

```
@NetworkResource{Hutter_prize,
author      = {English Wikipedia},
url         = "https://matmahoney.net/dc/textdata.html",
month       = {March},
year        = {2006}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.IMDBDataset

```
class mindspore.dataset.IMDBDataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None)
```

IMDb（Internet Movie Database）数据集。

生成的数据集有两列 `[text, label]`。`text` 列的数据类型是 `string`。`label` 列的数据类型是 `uint32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。默认值：None，读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> imdb_dataset_dir = "/path/to/imdb_dataset_directory"
>>>
>>> # 1) Read all samples (text files) in imdb_dataset_dir with 8 threads
>>> dataset = ds.IMDBDataset(dataset_dir=imdb_dataset_dir, num_parallel_workers=8)
>>>
>>> # 2) Read train samples (text files).
>>> dataset = ds.IMDBDataset(dataset_dir=imdb_dataset_dir, usage="train")
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 320: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **IMDB** 数据集：

IMDB 数据集包含来自互联网电影数据库 (IMDB) 的 50000 条高度两极分化的评论。数据集分为 25,000 条用于训练的评论和 25,000 条用于测试的评论，训练集和测试集都包含 50% 的积极评论和 50% 的消极评论。训练标签和测试标签分别是 0 和 1，其中 0 代表负样本，1 代表正样本。

可以将数据集文件解压缩到此目录结构中，并通过 MindSpore 的 API 读取。



引用:

```
@InProceedings{maas-EtAl:2011:ACL-HLT2011,
  author      = {Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan
    and Ng, Andrew Y. and Potts, Christopher},
  title       = {Learning Word Vectors for Sentiment Analysis},
  booktitle   = {Proceedings of the 49th Annual Meeting of the Association for Computational
    Linguistics:
      Human Language Technologies},
  month       = {June},
  year        = {2011},
  address     = {Portland, Oregon, USA},
  publisher   = {Association for Computational Linguistics},
  pages       = {142--150},
  url         = {http://www.aclweb.org/anthology/P11-1015}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 epoch 的数据集对象。

下页继续

表 321 – 续上页

<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.IWSLT2016Dataset

```
class mindspore.dataset.IWSLT2016Dataset (dataset_dir, usage=None, language_pair=None, valid_set=None,
                                           test_set=None, num_samples=None, shuffle=Shuffle.GLOBAL,
                                           num_shards=None, shard_id=None, num_parallel_workers=None,
                                           cache=None)
```

IWSLT2016（International Workshop on Spoken Language Translation）数据集。
生成的数据集有两列 `[text, translation]`。 `text` 列的数据类型是 `string`。 `translation` 列的数据类型是 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'valid'、'test' 或 'all'。默认值: None，读取全部样本。
- **language_pair** (sequence, 可选) - 包含源语言和目标语言的序列，支持的值为 ('en', 'fr')、('en', 'de')、('en', 'cs')、('en', 'ar')、('de', 'en')、('cs', 'en')、('ar', 'en')。默认值: None，默认为 ('de', 'en')。
- **valid_set** (str, 可选) - 标识验证集的字符串，支持的值为 'dev2010'、'tst2010'、'tst2011'、'tst2012'、'tst2013' 和 'tst2014'。默认值: None，默认为 'tst2013'。
- **test_set** (str, 可选) - 标识测试集的字符串，支持的值为 'dev2010'、'tst2010'、'tst2011'、'tst2012'、'tst2013' 和 'tst2014'。默认值: None，默认为 'tst2014'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None，读取所有样本。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值: *Shuffle.GLOBAL*。如果 *shuffle* 为 False，则不混洗，如果 *shuffle* 为 True，等同于将 *shuffle* 设置为 *mindspore.dataset.Shuffle.GLOBAL*。通过传入枚举变量设置数据混洗的模式：
 - *Shuffle.GLOBAL*：混洗文件和样本。
 - *Shuffle.FILES*：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。

- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> iwslt2016_dataset_dir = "/path/to/iwslt2016_dataset_dir"
>>> dataset = ds.IWSLT2016Dataset(dataset_dir=iwslt2016_dataset_dir, usage='all',
...                               language_pair=('de', 'en'), valid_set='tst2013', test_set=
...                               'tst2014')
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 **IWSLT2016** 数据集:

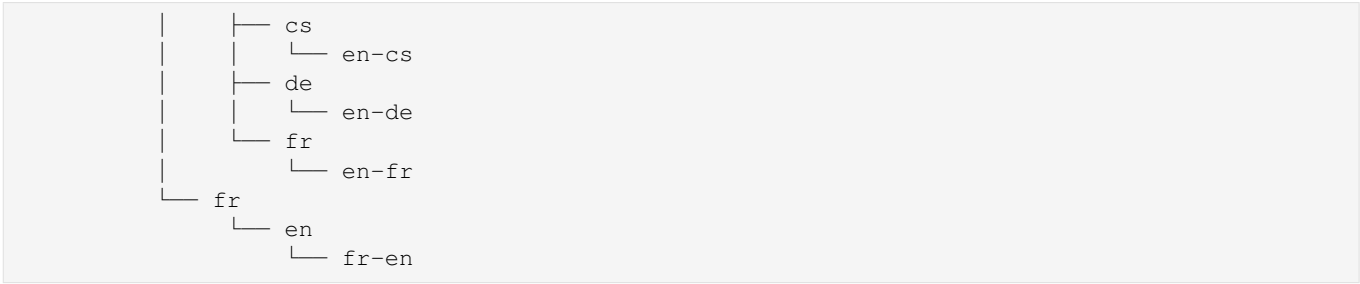
IWSLT 是一个专门讨论口译各个方面的重要年度科学会议。**IWSLT** 评估活动中的 **MT** 任务被构成一个数据集, 该数据集可通过 [wit3](#) 公开获取。**IWSLT2016** 数据集包括从英语到阿拉伯语、捷克、法语和德语的翻译, 以及从阿拉伯语、捷克、法语和德语到英语的翻译。

可以将原始 **IWSLT2016** 数据集文件解压缩到此目录结构中, 并由 **MindSpore** 的 API 读取。解压后, 还需要将要读取的数据集解压到指定文件夹中。例如, 如果要读取 **de-en** 的数据集, 则需要解压缩 **de/en** 目录下的 **tgz** 文件, 数据集位于解压缩文件夹中。

```
.
├── iwslt2016_dataset_directory
│   ├── subeval_files
│   └── texts
│       ├── ar
│       │   └── en
│       │       └── ar-en
│       ├── cs
│       │   └── en
│       │       └── cs-en
│       ├── de
│       │   └── en
│       │       ├── de-en
│       │       │   ├── IWSLT16.TED.dev2010.de-en.de.xml
│       │       │   ├── train.tags.de-en.de
│       │       │   └── ...
│       └── en
│           ├── ar
│           └── en-ar
```

(下页继续)

(续上页)



引用:

```
@inproceedings{cettoloEtAl:EAMT2012,
Address = {Trento, Italy},
Author = {Mauro Cettolo and Christian Girardi and Marcello Federico},
Booktitle = {Proceedings of the 16th Conference of the European Association for Machine
↪ Translation
(EAMT)},
Date = {28-30},
Month = {May},
Pages = {261--268},
Title = {WIT3: Web Inventory of Transcribed and Translated Talks},
Year = {2012}}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.IWSLT2017Dataset

class mindspore.dataset.IWSLT2017Dataset (dataset_dir, usage=None, language_pair=None, num_samples=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, num_parallel_workers=None, cache=None)

IWSLT2017 (International Workshop on Spoken Language Translation) 数据集。

生成的数据集有两列 [text, translation]。text 列和 translation 列的数据类型均为 string。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'valid'、'test' 或 'all'。默认值: None, 读取全部样本。
- **language_pair** (sequence, 可选) - 包含源语和目标语的语言列表, 支持的语言对有 ('en', 'nl')、('en', 'de')、('en', 'it')、('en', 'ro')、('nl', 'en')、('nl', 'de')、('nl', 'it')、('nl', 'ro')、('de', 'en')、('de', 'nl')、('de', 'it')、('de', 'ro')、('it', 'en')、('it', 'nl')、('it', 'de')、('it', 'ro')、('ro', 'en')、('ro', 'nl')、('ro', 'de')、('ro', 'it')。默认值: None, 默认为 ('de', 'en')。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取所有样本。
- **shuffle** (Union[bool, Shuffle], 可选) - 每个 epoch 中数据混洗的模式, 支持传入 bool 类型与枚举类型进行指定。默认值: Shuffle.GLOBAL。如果 shuffle 为 False, 则不混洗, 如果 shuffle 为 True, 等同于将 shuffle 设置为 mindspore.dataset.Shuffle.GLOBAL。通过传入枚举变量设置数据混洗的模式:
 - Shuffle.GLOBAL: 混洗文件和样本。
 - Shuffle.FILES: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, num_samples 表示每个分片的最大样本数。一般在 数据并行模式训练 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 num_shards 时才能指定此参数。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **cache** (DatasetCache, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 单节点数据缓存。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - dataset_dir 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 num_shards 参数, 但是未指定 shard_id 参数。
- **RuntimeError** - 指定了 shard_id 参数, 但是未指定 num_shards 参数。

- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> iwslt2017_dataset_dir = "/path/to/iwslt2017_dataset_dir"
>>> dataset = ds.IWslt2017Dataset(dataset_dir=iwslt2017_dataset_dir, usage='all', language_
    ↳pair=('de', 'en'))
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

关于 **IWSLT2017** 数据集:

IWSLT 是一个专门讨论口译各个方面的重要年度科学会议。IWSLT 评估活动中的 MT 任务被构成一个数据集, 该数据集可通过 [wit3](#) 公开获取。IWSLT2017 数据集集中有德语、英语、意大利语、荷兰语和罗马尼亚语, 数据集包括其中任何两种语言的翻译。

可以将原始 IWSLT2017 数据集文件解压缩到此目录结构中, 并由 MindSpore 的 API 读取。解压后, 还需要将要读取的数据集解压到指定文件夹中。例如, 如果要读取 de-en 的数据集, 则需要解压缩 de/en 目录下的 tgz 文件, 数据集位于解压缩文件夹中。

```
.
├── iwslt2017_dataset_directory
│   ├── subeval_files
│   └── texts
│       ├── ar
│       │   └── en
│       │       └── ar-en
│       ├── cs
│       │   └── en
│       │       └── cs-en
│       ├── de
│       │   └── en
│       │       ├── de-en
│       │       ├── IWslt16.TED.dev2010.de-en.de.xml
│       │       ├── train.tags.de-en.de
│       │       └── ...
│       ├── en
│       │   ├── ar
│       │   │   └── en-ar
│       │   ├── cs
│       │   │   └── en-cs
│       │   ├── de
│       │   │   └── en-de
│       │   ├── fr
│       │   │   └── en-fr
│       └── fr
│           └── en
│               └── fr-en
```

引用:

```
@inproceedings{cettoloEtAl:EAMT2012,
Address = {Trento, Italy},
Author = {Mauro Cettolo and Christian Girardi and Marcello Federico},
Booktitle = {Proceedings of the 16th Conference of the European Association for Machine
    ↳Translation
```

(下页继续)

(续上页)

```
        (EAMT) },
Date = {28-30},
Month = {May},
Pages = {261--268},
Title = {WIT$^3$: Web Inventory of Transcribed and Translated Talks},
Year = {2012}}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.Multi30kDataset

class mindspore.dataset.**Multi30kDataset** (*dataset_dir, usage=None, language_pair=None, num_samples=None, num_parallel_workers=None, shuffle=None, num_shards=None, shard_id=None, cache=None*)

Multi30k 数据集。

生成的数据集有两列: *[text, translation]*。text 列的数据类型为 string。label 列的数据类型为 string。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test'、'valid' 或 'all'。默认值: None, 读取全部样本图片。
- **language_pair** (Sequence[str, str], 可选) - 源语言与目标语言类别, 可取值为 ['en', 'de'] 或 ['de', 'en']。默认值: None, 表示 ['en', 'de']。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 是否混洗数据集。默认值: None, 表示 `mindspore.dataset.Shuffle.GLOBAL`。如果输入 False, 将不进行混洗。如果输入 True, 效果与设置 `mindspore.dataset.Shuffle.GLOBAL` 相同。如果输入 *Shuffle* 枚举值, 效果如下表所示:
 - *Shuffle.GLOBAL*: 混洗文件和文件中的数据。
 - *Shuffle.FILES*: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 路径下不包含数据文件。
- **ValueError** - *usage* 参数取值不为 'train'、'test'、'valid' 或 'all'。
- **TypeError** - 如果 *language_pair* 不为 Sequence[str, str] 类型。
- **RuntimeError** - 如果 *num_samples* 小于 0。
- **RuntimeError** - *num_parallel_workers* 参数超过系统最大线程数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> multi30k_dataset_dir = "/path/to/multi30k_dataset_directory"
>>> data = ds.Multi30kDataset(dataset_dir=multi30k_dataset_dir, usage='all', language_pair=[
    ↪ 'de', 'en'])
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

关于 **Multi30k** 数据集：

Multi30k 是一个多语言的计算机视觉数据集，包含了约 3.1 万个以多种语言描述的标准图像。这些图像来自 **Flickr** 数据集，每个图像都配有英语和德语的描述，以及其他多种语言。**Multi30k** 常用在图像描述生成、机器翻译、视觉问答等任务的训练和测试中。

您可以将数据集解压并构建成以下目录结构，并通过 **MindSpore** 的 API 进行读取。

```
└─ multi30k_dataset_directory
   │
   ├── training
   │ │   ├── train.de
   │ │   └── train.en
   │
   ├── validation
   │ │   ├── val.de
   │ │   └── val.en
   │
   └── mmt16_task1_test
       ├── val.de
       └── val.en
```

引用：

```
@article{elliott-EtAl:2016:VL16,
author   = {{Elliott}, D. and {Frank}, S. and {Sima'an}, K. and {Specia}, L.},
title    = {Multi30K: Multilingual English-German Image Descriptions},
booktitle = {Proceedings of the 5th Workshop on Vision and Language},
year     = {2016},
pages    = {70--74},
year     = 2016
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。

下页继续

表 339 – 续上页

<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.PennTreebankDataset

`class mindspore.dataset.PennTreebankDataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, cache=None)`

PennTreebank 数据集。

生成的数据集有一列 `[text]`。`text` 列的数据类型为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test'、'valid' 或 'all'。取值为 'train' 将读取 42,068 个样本，'test' 将读取 3,370 个样本，'valid' 将读取 3,761 个样本，'all' 将读取所有 49,199 个样本。默认值：None，读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：*Shuffle.GLOBAL*。如果 *shuffle* 为 False，则不混洗；如果 *shuffle* 为 True，等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式：
 - *Shuffle.GLOBAL*：混洗文件和样本。
 - *Shuffle.FILES*：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时，数据集划分的片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时，使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时，才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 参数所指向的文件目录不存在或缺少数数据集文件。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> penn_treebank_dataset_dir = "/path/to/penn_treebank_dataset_directory"
>>> dataset = ds.PennTreebankDataset(dataset_dir=penn_treebank_dataset_dir, usage='all')
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

关于 **PennTreebank** 数据集:

Penn Treebank (PTB) 数据集, 广泛用于 NLP (自然语言处理) 的机器学习研究。PTB 不包含大写字母、数字和标点符号, 其词汇表上限为 10k 个不重复词, 与大多数现代数据集相比相对较小, 可能会导致出现大量超出词汇表外的 token。

以下是原始的 PennTreebank 数据集结构。可以将数据集文件解压缩到此目录结构中, 并通过 MindSpore 的 API 读取。

```
.
├── PennTreebank_dataset_dir
│   ├── ptb.test.txt
│   ├── ptb.train.txt
│   └── ptb.valid.txt
```

引用:

```
@techreport{Santorini1990,
  added-at = {2014-03-26T23:25:56.000+0100},
  author = {Santorini, Beatrice},
  biburl = {https://www.bibsonomy.org/bibtex/234cdf6ddadd89376090e7dada2fc18ec/butonic},
  file = {:Santorini - Penn Treebank tag definitions.pdf:PDF},
  institution = {Department of Computer and Information Science, University of Pennsylvania},
  interhash = {818e72efd9e4b5fae3e51e88848100a0},
  intrahash = {34cdf6ddadd89376090e7dada2fc18ec},
  keywords = {dis pos tagging treebank},
  number = {MS-CIS-90-47},
  timestamp = {2014-03-26T23:25:56.000+0100},
  title = {Part-of-speech tagging guidelines for the {P}enn {T}reebank {P}roject},
  url = {ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz},
  year = 1990
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.SogouNewsDataset

```
class mindspore.dataset.SogouNewsDataset (dataset_dir, usage=None, num_samples=None,
                                           shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None,
                                           num_parallel_workers=None, cache=None)
```

SogouNews 数据集。

生成的数据集有三列 `[index, title, content]`，三列的数据类型均为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。默认值：None，读取全部样本。取值为 'train' 时将会读取 45 万个训练样本，取值为 'test' 时将会读取 6 万个测试样本，取

值为 'all' 时将会读取全部 51 万个样本。

- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式, 支持传入 bool 类型与枚举类型进行指定。默认值: *Shuffle*.GLOBAL。如果 *shuffle* 为 False, 则不混洗, 如果 *shuffle* 为 True, 等同于将 *shuffle* 设置为 *mindspore.dataset.Shuffle*.GLOBAL。通过传入枚举变量设置数据混洗的模式:
 - *Shuffle*.GLOBAL: 混洗文件和样本。
 - *Shuffle*.FILES: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 *mindspore.dataset.config.set_num_parallel_workers()* 配置全局线程数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> sogou_news_dataset_dir = "/path/to/sogou_news_dataset_dir"
>>> dataset = ds.SogouNewsDataset(dataset_dir=sogou_news_dataset_dir, usage='all')
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 **SogouNews** 数据集:

SogouNews 数据集包括 3 列, 分别对应类别索引 (1 到 5)、标题和内容。标题和内容使用双引号 (") 进行转义, 任何内部双引号都使用 2 个双引号 (") 进行转义。新行使用反斜杠进行转义, 后跟 "n" 字符, 即 "\n"。

以下是原始 **SogouNews** 数据集结构, 可以将数据集文件解压缩到此目录结构中, 并由 MindSpore 的 API 读取:

```
.
├── sogou_news_dir
│   ├── classes.txt
│   ├── readme.txt
│   ├── test.csv
│   └── train.csv
```

引用:

```
@misc{zhang2015characterlevel,
  title={Character-level Convolutional Networks for Text Classification},
  author={Xiang Zhang and Junbo Zhao and Yann LeCun},
  year={2015},
  eprint={1509.01626},
  archivePrefix={arXiv},
  primaryClass={cs.LG}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.SQuADDataset

class mindspore.dataset.SQuADDataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, cache=None)

SQuAD 1.1 和 SQuAD 2.0 数据集。

不同版本和子集生成的数据集具有相同的列: [context, question, text, answer_start]。context 列的数据类型为 string。question 列的数据类型为 string。text 列为上下文中的回答, 数据类型为 string。answer_start 列为上下文中回答的起始索引, 数据类型为 uint32。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'dev' 或 'all'。默认值: None, 读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, Shuffle], 可选) - 是否混洗数据集。默认值: Shuffle.GLOBAL。如果输入 False, 将不进行混洗。如果输入 True, 效果与设置 `mindspore.dataset.Shuffle.GLOBAL` 相同。如果输入 `Shuffle` 枚举值, 效果如下表所示:
 - `Shuffle.GLOBAL`: 混洗文件和文件中的数据。
 - `Shuffle.FILES`: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> squad_dataset_dir = "/path/to/squad_dataset_file"
>>> dataset = ds.SQuADDataset(dataset_dir=squad_dataset_dir, usage='all')
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 SQuAD 数据集:

SQuAD (Stanford Question Answering Dataset) 是一个阅读理解数据集, 由众人对一组维基百科文章提出的问题组成, 每个问题的答案都是相应阅读段落中的一段文字或范围, 否则问题可能无法回答。

SQuAD 1.1, 即 SQuAD 数据集的前一个版本, 包含 500 多篇文章的 100,000 多个问题-答案对。SQuAD 2.0 除包含 SQuAD 1.1 中的 100,000 个问题外, 还补充了超过 50,000 个由贡献者编写的不可回答的对抗性问题, 它们看起来与可回答的问题类似。为了在 SQuAD 2.0 中取得好成绩, 系统不仅要尽量回答可回答的问题, 而且要能够在段落中不存在答案时放弃回答。

您可以将数据集解压并构建以下目录结构, 并通过 MindSpore 的 API 进行读取。

SQuAD 1.1:

```
.
└─ SQuAD1
   └─ train-v1.1.json
   └─ dev-v1.1.json
```

SQuAD 2.0:

```
.
└─ SQuAD2
   └─ train-v2.0.json
   └─ dev-v2.0.json
```

引用:

```
@misc{rajpurkar2016squad,
  title = {SQuAD: 100,000+ Questions for Machine Comprehension of Text},
  author = {Pranav Rajpurkar and Jian Zhang and Konstantin Lopyrev and Percy Liang},
  year = {2016},
  eprint = {1606.05250},
  archivePrefix = {arXiv},
  primaryClass = {cs.CL}
}

@misc{rajpurkar2018know,
  title = {Know What You Don't Know: Unanswerable Questions for SQuAD},
  author = {Pranav Rajpurkar and Robin Jia and Percy Liang},
  year = {2018},
  eprint = {1806.03822},
  archivePrefix = {arXiv},
  primaryClass = {cs.CL}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理, 并将结果展开。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表, 按顺序将数据增强作用在数据集对象上。

下页继续

表 357 – 续上页

<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。

下页继续

表 360 – 续上页

<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <code>shape</code> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.SST2Dataset

class `mindspore.dataset.SST2Dataset` (`dataset_dir`, `usage=None`, `num_samples=None`, `num_parallel_workers=None`, `shuffle=Shuffle.GLOBAL`, `num_shards=None`, `shard_id=None`, `cache=None`)

SST2 (Stanford Sentiment Treebank v2) 数据集。

数据集中 `train.tsv` 文件和 `dev.tsv` 有两列 [`sentence`, `label`]。数据集中 `test.tsv` 文件中有一列 [`sentence`]。`sentence` 列和 `label` 列的数据类型都是 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'dev'。取值为 'train' 时将会读取 67,349 个训练样本，取值为 'test' 时将会读取 1,821 个测试样本，取值为 'dev' 时将会读取 872 个样本。默认值：None，读取 `train` 中样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, `Shuffle`], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：`Shuffle.GLOBAL`。如果 `shuffle` 为 False，则不混洗，如果 `shuffle` 为 True，等同于将 `shuffle` 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式：
 - `Shuffle.GLOBAL`：混洗文件和样本。
 - `Shuffle.FILES`：仅混洗文件。

- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *shard_id* 参数值错误, 小于 0 或者大于等于 *num_shards*。

样例:

```
>>> import mindspore.dataset as ds
>>> sst2_dataset_dir = "/path/to/sst2_dataset_directory"
>>>
>>> # 1) Read 3 samples from SST2 dataset
>>> dataset = ds.SST2Dataset(dataset_dir=sst2_dataset_dir, num_samples=3)
>>>
>>> # 2) Read train samples from SST2 dataset
>>> dataset = ds.SST2Dataset(dataset_dir=sst2_dataset_dir, usage="train")
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 SST2 数据集:

Stanford Sentiment Treebank 是一个具有完全标记解析树的语料库, 可以对语言中情感的合成效果进行完整的分析。语料库基于 Pang 和 Lee (2005) 介绍的数据集, 由 11855 个从电影评论中提取的句子组成。它是用斯坦福解析器解析的, 共包含 215154 个来自这些解析树的独特短语, 每个短语都由 3 个人类评委进行注释。

以下为原始 SST2 数据集的结构, 您可以将数据集文件解压得到如下的文件结构, 并通过 MindSpore 的 API 进行读取。

```
.
├── sst2_dataset_dir
│   ├── train.tsv
│   ├── test.tsv
│   ├── dev.tsv
│   └── original
```

引用:

```
@inproceedings{socher-etal-2013-recursive,
  title = {Recursive Deep Models for Semantic Compositionality Over a Sentiment_
↪Treebank},
  author = {Socher, Richard and Perelygin, Alex and Wu, Jean and Chuang, Jason and_
↪Manning, Christopher D. and Ng, Andrew and Potts, Christopher},
  booktitle = {Proceedings of the 2013 Conference on Empirical Methods in Natural Language_
↪Processing},
```

(下页继续)

(续上页)

```
month      = oct,
year       = {2013},
address    = {Seattle, Washington, USA},
publisher  = {Association for Computational Linguistics},
url        = {https://www.aclweb.org/anthology/D13-1170},
pages      = {1631--1642},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

`mindspore.dataset.TextFileDataset`

```
class mindspore.dataset.TextFileDataset (dataset_files, num_samples=None, num_parallel_workers=None,  
                                           shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None,  
                                           cache=None)
```

读取和解析文本文件构建数据集。生成的数据集有一个数据列：`[text]`，类型为 `string`。

参数：

- **dataset_files** (`Union[str, list[str]]`) - 数据集文件路径，支持单文件路径字符串、多文件路径字符串列表或可被 `glob` 库模式匹配的字符串，文件列表将在内部进行字典排序。
- **num_samples** (`int`, 可选) - 指定从数据集中读取的样本数。默认值：`None`，读取所有样本。
- **num_parallel_workers** (`int`, 可选) - 指定读取数据的工作线程数。默认值：`None`，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (`Union[bool, Shuffle]`, 可选) - 每个 `epoch` 中数据混洗的模式，支持传入 `bool` 类型与枚举类型进行指定。默认值：`Shuffle.GLOBAL`。如果 `shuffle` 为 `False`，则不混洗，如果 `shuffle` 为 `True`，等同于将 `shuffle` 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式：
 - `Shuffle.GLOBAL`：混洗文件和样本。
 - `Shuffle.FILES`：仅混洗文件。
- **num_shards** (`int`, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：`None`。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (`int`, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：`None`。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：`None`，不使用缓存。

异常：

- **ValueError** - `dataset_files` 参数所指向的文件无效或不存在。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> text_file_list = ["/path/to/text_file_dataset_file"] # contains 1 or multiple text files
>>> dataset = ds.TextFileDataset(dataset_files=text_file_list)
```

教程样例：

- [使用数据 Pipeline 加载 & 处理数据集](#)

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.UDPOSDataset

class mindspore.dataset.UDPOSDataset (*dataset_dir, usage=None, num_samples=None, shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None, num_parallel_workers=None, cache=None*)

UDPOS（Universal Dependencies dataset for Part of Speech）数据集。

生成的数据集有三列 [*word, universal, stanford*]，三列的数据类型均为 string。

参数：

- dataset_dir** (str) - 包含数据集文件的根目录路径。
- usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test'、'valid' 或 'all'。取值为 'train' 时将会读取 12,543 个样本，取值为 'test' 时将会读取 2,077 个测试样本，取值为 'valid' 时将会读取

2,002 个样本，取值为 'all' 时将会读取全部 16,622 个样本。默认值：None，读取全部样本。

- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部样本。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：*Shuffle.GLOBAL*。如果 *shuffle* 为 False，则不混洗，如果 *shuffle* 为 True，等同于将 *shuffle* 设置为 *mindspore.dataset.Shuffle.GLOBAL*。通过传入枚举变量设置数据混洗的模式：
 - *Shuffle.GLOBAL*：混洗文件和样本。
 - *Shuffle.FILES*：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 *mindspore.dataset.config.set_num_parallel_workers()* 配置全局线程数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例：

```
>>> import mindspore.dataset as ds
>>> udpos_dataset_dir = "/path/to/udpos_dataset_dir"
>>> dataset = ds.UDPOSDataset(dataset_dir=udpos_dataset_dir, usage='all')
```

教程样例：

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 UDPOS 数据集：

UDPOS 是一个解析的文本语料库数据集，用于阐明句法或者语义句子结构。该语料库包含 254,830 个单词和 16,622 个句子，取自各种网络媒体，包括博客、新闻组、电子邮件和评论。

引用：

```
@inproceedings{silveira14gold,
  year = {2014},
  author = {Natalia Silveira and Timothy Dozat and Marie-Catherine de Marneffe and Samuel Bowman
    and Miriam Connor and John Bauer and Christopher D. Manning},
  title = {A Gold Standard Dependency Corpus for {E}nglish},
  booktitle = {Proceedings of the Ninth International Conference on Language
    Resources and Evaluation (LREC-2014)}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.WikiTextDataset

```
class mindspore.dataset.WikiTextDataset (dataset_dir, usage=None, num_samples=None,
                                         num_parallel_workers=None, shuffle=Shuffle.GLOBAL,
                                         num_shards=None, shard_id=None, cache=None)
```

WikiText2 和 WikiText103 数据集。

生成的数据集有一列 `[text]`，数据类型为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test'、'valid' 或 'all'。默认值: None，读取全部样本。

- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式, 支持传入 bool 类型与枚举类型进行指定。默认值: *Shuffle.GLOBAL*。如果 *shuffle* 为 False, 则不混洗, 如果 *shuffle* 为 True, 等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式:
 - *Shuffle.GLOBAL*: 混洗文件和样本。
 - *Shuffle.FILES*: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, \text{num\_shards})$ 范围。
- **ValueError** - *num_samples* 参数值错误, 小于 0。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> wiki_text_dataset_dir = "/path/to/wiki_text_dataset_directory"
>>> dataset = ds.WikiTextDataset(dataset_dir=wiki_text_dataset_dir, usage='all')
```

教程样例:

- [使用数据 Pipeline 加载 & 处理数据集](#)

关于 **WikiText** 数据集:

WikiText 数据集是一个包含 1 亿字的英语词典。这些样本术语来自维基百科的高级和基础文章, 包括 Wikitext2 和 Wikitext103 的版本。对于 WikiText2, 分别在 `wiki.train.tokens` 中有 36718 个样本, 在 `wiki.test.tokens` 中有 4358 个样本, 在 `wiki.valid.tokens` 中有 3760 个样本。对于 WikiText103, 分别在 `wiki.train.tokens` 中有 1801350 个样本, `wiki.test.tokens` 中的 4358 个样本, `Wiki.valid.tokens` 中的 3760 个样本。

以下是原始的 WikiText 数据集结构。可以将数据集文件解压缩到此目录结构中, 并由 MindSpore 的 API 读取。

```
.
├── WikiText2/WikiText103
│   ├── wiki.train.tokens
│   ├── wiki.test.tokens
│   └── wiki.valid.tokens
```

引用:

```
@article{merity2016pointer,
  title={Pointer sentinel mixture models},
  author={Merity, Stephen and Xiong, Caiming and Bradbury, James and Socher, Richard},
  journal={arXiv preprint arXiv:1609.07843},
  year={2016}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.YahooAnswersDataset

```
class mindspore.dataset.YahooAnswersDataset (dataset_dir, usage=None, num_samples=None,
                                             num_parallel_workers=None, shuffle=Shuffle.GLOBAL,
                                             num_shards=None, shard_id=None, cache=None)
```

YahooAnswers 数据集。

生成的数据集有四列 `[class, title, content, answer]`，数据类型均为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'test' 或 'all'。取值为 'train' 时将会读取 1,400,000 个训练样本，取值为 'test' 时将会读取 60,000 个测试样本，取值为 'all' 时将会读取全部 1,460,000 个样本。默认值：None，读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式，支持传入 bool 类型与枚举类型进行指定。默认值：Shuffle.GLOBAL。如果 *shuffle* 为 False，则不混洗，如果 *shuffle* 为 True，等同于将 *shuffle* 设置为 `mindspore.dataset.Shuffle.GLOBAL`。通过传入枚举变量设置数据混洗的模式：
 - Shuffle.GLOBAL：混洗文件和样本。
 - Shuffle.FILES：仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 `[0, num_shards)` 范围。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例：

```
>>> import mindspore.dataset as ds
>>> yahoo_answers_dataset_dir = "/path/to/yahoo_answers_dataset_directory"
>>>
>>> # 1) Read 3 samples from YahooAnswers dataset
>>> dataset = ds.YahooAnswersDataset(dataset_dir=yahoo_answers_dataset_dir, num_samples=3)
>>>
>>> # 2) Read train samples from YahooAnswers dataset
>>> dataset = ds.YahooAnswersDataset(dataset_dir=yahoo_answers_dataset_dir, usage="train")
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

关于 **YahooAnswers** 数据集：

YahooAnswers 数据集包含 10 个类的 63 万个文本样本。**train.csv** 中有 56 万个样本，**test.csv** 中有 7 万个样本。这 10 个不同的类代表社会与文化、科学与数学、健康、教育与参考、计算机与互联网、体育、商业与金融、娱乐与音乐、家庭与关系、政治与政府。

以下是原始的 **YahooAnswers** 数据集结构，可以将数据集文件解压缩到此目录结构中，并由 **Mindspore** 的 **API** 读取。

```
.
├── yahoo_answers_dataset_dir
│   ├── train.csv
│   ├── test.csv
│   ├── classes.txt
│   └── readme.txt
```

引用：

```
@article{YahooAnswers,
title    = {Yahoo! Answers Topic Classification Dataset},
author   = {Xiang Zhang},
year     = {2015},
howpublished = {}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 epoch 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩，多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

mindspore.dataset.YelpReviewDataset

```
class mindspore.dataset.YelpReviewDataset (dataset_dir, usage=None, num_samples=None,
                                           shuffle=Shuffle.GLOBAL, num_shards=None, shard_id=None,
                                           num_parallel_workers=None, cache=None)
```

Yelp Review Full 和 Yelp Review Polarity 数据集。
生成的数据集有两列 `[label, text]`, 两列的数据类型均为 `string`。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 'train'、'test' 或 'all'。默认值: None, 读取所有样本。
 - 对于 Polarity 数据集, 'train' 将读取 560,000 个训练样本, 'test' 将读取 38,000 个测试样本, 'all' 将读取所有 598,000 个样本。
 - 对于 Full 数据集, 'train' 将读取 650,000 个训练样本, 'test' 将读取 50,000 个测试样本, 'all' 将读取所有 700,000 个样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **shuffle** (Union[bool, *Shuffle*], 可选) - 每个 epoch 中数据混洗的模式, 支持传入 bool 类型与枚举类型进行指定。默认值: *Shuffle.GLOBAL*。如果 *shuffle* 为 False, 则不混洗, 如果 *shuffle* 为 True, 等同于将 *shuffle* 设置为 *mindspore.dataset.Shuffle.GLOBAL*。通过传入枚举变量设置数据混洗的模式:
 - *Shuffle.GLOBAL*: 混洗文件和样本。
 - *Shuffle.FILES*: 仅混洗文件。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, *num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 *num_shards* 时才能指定此参数。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - *dataset_dir* 参数所指向的文件目录不存在或缺少数据集文件。
- **RuntimeError** - 指定了 *num_shards* 参数, 但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数, 但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。

样例:

```
>>> import mindspore.dataset as ds
>>> yelp_review_dataset_dir = "/path/to/yelp_review_dataset_dir"
>>> dataset = ds.YelpReviewDataset(dataset_dir=yelp_review_dataset_dir, usage='all')
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

关于 **YelpReview** 数据集:

Yelp Review Full 数据集包括来自 Yelp 的评论数据。这些数据是从 2015 年的 Yelp 数据集挑战赛数据中提取的, 主要用于文本分类。

Yelp Review Polarity 数据集在 Full 数据集的基础上, 对产品评分进行了分级, 评论分数 1 和 2 视为负面评论, 4 和 5 视为正面评论。

Yelp Reviews Polarity 和 Yelp Reviews Full datasets 具有相同的目录结构。可以将数据集文件解压缩到以下结构, 并通过 MindSpore 的 API 读取:

```
.
├── yelp_review_dir
│   ├── train.csv
│   ├── test.csv
│   └── readme.txt
```

引用:

```
@article{zhangCharacterlevelConvolutionalNetworks2015,
  archivePrefix = {arXiv},
  eprinttype = {arxiv},
  eprint = {1509.01626},
  primaryClass = {cs},
  title = {Character-Level {{Convolutional Networks}} for {{Text Classification}}},
  abstract = {This article offers an empirical exploration on the use of character-level
↪convolutional networks
              (ConvNets) for text classification. We constructed several large-scale
↪datasets to show that
              character-level convolutional networks could achieve state-of-the-art or
↪competitive results.
              Comparisons are offered against traditional models such as bag of words, n-
↪grams and their TFIDF
              variants, and deep learning models such as word-based ConvNets and recurrent
↪neural networks.},
  journal = {arXiv:1509.01626 [cs]},
  author = {Zhang, Xiang and Zhao, Junbo and LeCun, Yann},
  month = sep,
  year = {2015},
}
```

```
@article{zhangCharacterlevelConvolutionalNetworks2015,
  archivePrefix = {arXiv},
  eprinttype = {arxiv},
  eprint = {1509.01626},
  primaryClass = {cs},
  title = {Character-Level {{Convolutional Networks}} for {{Text Classification}}},
  abstract = {This article offers an empirical exploration on the use of character-level
↪convolutional networks
```

(下页继续)

(续上页)

```
        (ConvNets) for text classification. We constructed several large-scale_
↪datasets to show that
        character-level convolutional networks could achieve state-of-the-art or_
↪competitive results.
        Comparisons are offered against traditional models such as bag of words, n-
↪grams and their TFIDF
        variants, and deep learning models such as word-based ConvNets and recurrent_
↪neural networks.},
    journal = {arXiv:1509.01626 [cs]},
    author = {Zhang, Xiang and Zhao, Junbo and LeCun, Yann},
    month = sep,
    year = {2015},
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

12.5.3 音频数据集

<code>mindspore.dataset.CMUArcticDataset</code>	CMU Arctic 数据集。
<code>mindspore.dataset.GTZANDataset</code>	GTZAN 数据集。
<code>mindspore.dataset.LibriTTSDataset</code>	LibriTTS 数据集。
<code>mindspore.dataset.LJSpeechDataset</code>	LJSpeech 数据集。
<code>mindspore.dataset.SpeechCommandsDataset</code>	Speech Commands 数据集。
<code>mindspore.dataset.TedliumDataset</code>	Tedlium 数据集。
<code>mindspore.dataset.YesNoDataset</code>	YesNo 数据集。

mindspore.dataset.CMUArcticDataset

class mindspore.dataset.CMUArcticDataset (dataset_dir, name=None, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None)

CMU Arctic 数据集。

生成的数据集有四列 [waveform, sample_rate, transcript, utterance_id]。waveform 列的数据类型为 float32。sample_rate 列的数据类型为 uint32。transcript 列的数据类型为 string。utterance_id 列的数据类型为 string。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **name** (str, 可选) - 指定读取的数据集子集，可为 'aew'、'ahw'、'aup'、'awb'、'axb'、'bd1'、'clb'、'eey'、'fem'、'gka'、'jmk'、'ksp'、'ljm'、'lnh'、'rms'、'rxr'、'slp' 或 'slt'。默认值：None，表示 'aew'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部音频。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None，下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None，下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None，不进行分片。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None，将使用 0。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (DatasetCache, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。

- **ValueError** - 如果 *shard_id* 取值不在 $[0, num\_shards)$ 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> cmu_arctic_dataset_directory = "/path/to/cmu_arctic_dataset_directory"
>>>
>>> # 1) Read 500 samples (audio files) in cmu_arctic_dataset_directory
>>> dataset = ds.CMUArcticDataset(cmu_arctic_dataset_directory, name="ahw", num_samples=500)
>>>
>>> # 2) Read all samples (audio files) in cmu_arctic_dataset_directory
>>> dataset = ds.CMUArcticDataset(cmu_arctic_dataset_directory)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

说明:

- 暂不支持指定 *sampler* 参数为 `mindspore.dataset.PKSampler`。
- 入参 *num_samples*、*shuffle*、*num_shards*、*shard_id* 可用于控制数据集所使用的采样器，其与入参 *sampler* 搭配使用的效果如下。

表 400: 参数 *sampler* 和 *num_samples*，*shuffle*，*num_shards*，*shard_id* 的不同组合得到的采样器

参数 <i>sampler</i>	参数 <i>num_shards</i> / <i>shard_id</i>	参数 <i>shuffle</i>	参数 <i>num_samples</i>	使用的采样器
<code>mindspore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 CMUArctic 数据集:

CMU Arctic 数据集是为语音合成研究而设计的。这些单人语音数据是在演播室条件下精心录制的，由大约 1200 个语音平衡的英语语料组成。除了音频文件外，数据集还为 Festival 语音合成系统提供了完整的支持，包括可按原样使

用的预建语音。整个软件包是作为免费软件发布的，不限制商业或非商业使用。
您可以将 CMUArctic 数据集构建以下目录结构，并通过 MindSpore 的 API 进行读取。

```
.
├── cmu_arctic_dataset_directory
│   ├── cmu_us_aew_arctic
│   │   ├── wav
│   │   │   ├── arctic_a0001.wav
│   │   │   ├── arctic_a0002.wav
│   │   │   └── ...
│   │   └── etc
│   │       └── txt.done.data
│   ├── cmu_us_ahw_arctic
│   │   ├── wav
│   │   │   ├── arctic_a0001.wav
│   │   │   ├── arctic_a0002.wav
│   │   │   └── ...
│   │   └── etc
│   │       └── txt.done.data
│   └── ...
```

引用:

```
@article{LTI2003CMUArctic,
title      = {CMU ARCTIC databases for speech synthesis},
author     = {John Kominek and Alan W Black},
journal    = {Language Technologies Institute [Online]},
year       = {2003}
howpublished = {http://www.festvox.org/cmu_arctic/}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。

下页继续

表 401 – 续上页

<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩, 多个 dataset 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.GTZANDataset

`class mindspore.dataset.GTZANDataset (dataset_dir, usage=None, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None)`

GTZAN 数据集。

生成的数据集有三列 `[waveform, sample_rate, label]`。 `waveform` 列的数据类型为 `float32`。 `sample_rate` 列的数据类型为 `uint32`。 `label` 列的数据类型为 `string`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'train'、'valid'、'test' 或 'all'。默认值: None，表示读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None，读取全部音频。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None，下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None，下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后，`num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值: None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。

- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> gtzan_dataset_directory = "/path/to/gtzan_dataset_directory"
>>>
>>> # 1) Read 500 samples (audio files) in gtzan_dataset_directory
>>> dataset = ds.GTZANDataset(gtzan_dataset_directory, usage="all", num_samples=500)
>>>
>>> # 2) Read all samples (audio files) in gtzan_dataset_directory
>>> dataset = ds.GTZANDataset(gtzan_dataset_directory)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：

- 暂不支持指定 `sampler` 参数为 `mindspore.dataset.PKSampler`。
 - 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。
-

表 407: 参数 `sampler` 和 `num_samples`, `shuffle`, `num_shards`, `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>DatasetSampler(indices = sampler , num_samples = num_samples)</code>
iterable 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **GTZAN** 数据集：

GTZAN 数据集至少出现在 100 个已发表的工作中，是机器听觉研究中最常用的公共数据集，用于音乐曲风识别。它由 1000 条音轨组成，每条音轨的长度为 30 秒。它包含 10 种流派（蓝调、古典、乡村、迪斯科、hiphop、爵士、金属、流行、雷鬼和摇滚），每一种曲风各 100 条音轨。这些音轨都是 22050Hz 的单声道 16 位.wav 格式音频文件。

您可以将 **GTZAN** 数据集构建成以下目录结构，并通过 **MindSpore** 的 API 进行读取。

```
.
├── gtzan_dataset_directory
│   ├── blues
│   │   ├── blues.00000.wav
│   │   ├── blues.00001.wav
│   │   ├── blues.00002.wav
│   │   └── ...
│   ├── disco
│   │   ├── disco.00000.wav
│   │   ├── disco.00001.wav
│   │   ├── disco.00002.wav
│   │   └── ...
│   └── ...
```

引用：

```
@misc{tzanetakis_essl_cook_2001,
author   = "Tzanetakis, George and Essl, Georg and Cook, Perry",
title    = "Automatic Musical Genre Classification Of Audio Signals",
url      = "http://ismir2001.ismir.net/pdf/tzanetakis.pdf",
```

(下页继续)

(续上页)

```
publisher = "The International Society for Music Information Retrieval",
year      = "2001"
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.LibriTTSDataSet

```
class mindspore.dataset.LibriTTSDataSet (dataset_dir, usage=None, num_samples=None,
                                         num_parallel_workers=None, shuffle=None, sampler=None,
                                         num_shards=None, shard_id=None, cache=None)
```

LibriTTS 数据集。

生成的数据集有七列 [*waveform*, *sample_rate*, *original_text*, *normalized_text*, *speaker_id*, *chapter_id*, *utterance_id*]。*waveform* 列的数据类型为 float32。*sample_rate* 列的数据类型为 uint32。*original_text* 列的数据类型为 string。*normalized_text* 列的数据类型为 string。*speaker_id* 列的数据类型为 uint32。*chapter_id* 列的数据类型为 uint32。*utterance_id* 列的数据类型为 string。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集，可取值为 'dev-clean'、'dev-other'、'test-clean'、'test-other'、'train-clean-100'、'train-clean-360'、'train-other-500' 或 'all'。默认值：None，表示读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部音频。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None，下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None，下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - *dataset_dir* 路径下不包含任何数据文件。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 $[0, \text{num\_shards})$ 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> libri_tts_dataset_dir = "/path/to/libri_tts_dataset_directory"
>>>
>>> # 1) Read 500 samples (audio files) in libri_tts_dataset_directory
>>> dataset = ds.LibriTTSDataSet(libri_tts_dataset_dir, usage="train-clean-100", num_
    ↳ samples=500)
>>>
```

(下页继续)

(续上页)

```
>>> # 2) Read all samples (audio files) in libri_tts_dataset_directory
>>> dataset = ds.LibriTTSDataset(libri_tts_dataset_dir)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：

- 暂不支持指定 `sampler` 参数为 `mindspore.dataset.PKSampler`。
- 入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 414: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mindspore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	<code>sampler</code>
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>None</code>	<code>True</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>None</code>	<code>True</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 LibriTTS 数据集：

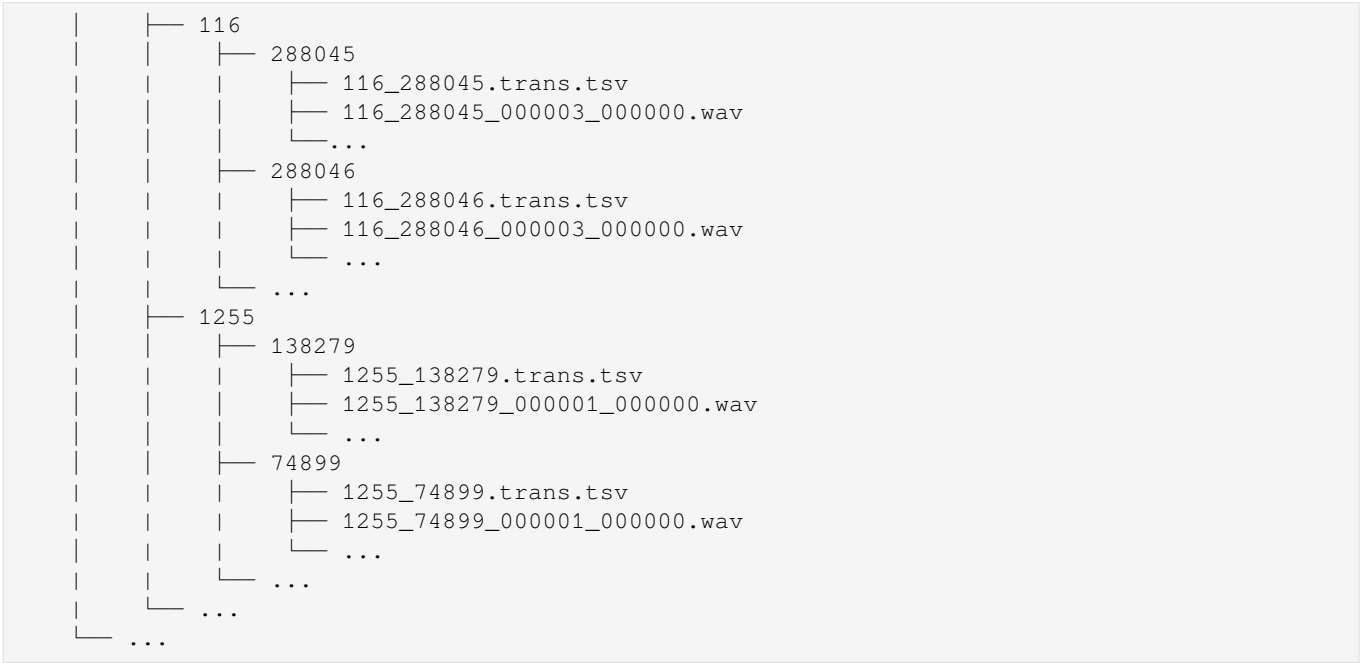
LibriTTS 是由 Heiga Zen 在谷歌语音和谷歌大脑团队成员的协助下准备的一个多语者英语语料库，包括了大约 585 小时的 24kHz 采样率的英语语音。LibriTTS 语料库是为 TTS 研究设计的。它是由 LibriSpeech 语料库的原始语料（来自 LibriVox 的 mp3 音频文件和来自 Project Gutenberg 的文本文件）衍生而来。

您可以将 LibriTTS 数据集构建以下目录结构，并通过 MindSpore 的 API 进行读取。

```
.
├── libri_tts_dataset_directory
│   └── dev-clean
```

(下页继续)

(续上页)



引用:

```
@article{lecun2010mnist,
title      = {LIBRITTS handwritten digit database},
author     = {zpw, NBU},
journal    = {ATT Labs [Online]},
volume     = {2},
year       = {2010},
howpublished = {http://www.openslr.org/resources/60/},
description = {The LibriSpeech ASR corpus (http://www.openslr.org/12/) [1] has been used in
various research projects. However, as it was originally designed for ASR,
↪research,
there are some undesired properties when using for TTS research}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。

下页继续

表 415 – 续上页

<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <code>epoch</code> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <code>buffer_size</code> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <code>count</code> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <code>dataset</code> 对象按列进行合并压缩, 多个 <code>dataset</code> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <code>batch_size</code> 条数据组合为一个批数据, 并可通过可选参数 <code>per_batch_map</code> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <code>batch_size</code> 条数据组合为一个批数据, 并可通过可选参数 <code>pad_info</code> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <code>epoch</code> 中的 <code>batch</code> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <code>RepeatDataset</code> 中定义的 <code>repeat</code> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <code>shape</code> 。

下页继续

表 418 – 续上页

<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.LJSpeechDataset

```
class mindspore.dataset.LJSpeechDataset (dataset_dir, num_samples=None, num_parallel_workers=None,
                                         shuffle=None, sampler=None, num_shards=None, shard_id=None,
                                         cache=None)
```

LJSpeech 数据集。

生成的数据集有四列: [waveform, sample_rate, transcription, normalized_transcript] 。 waveform 列的数据类型为 float32。 sample_rate 列的数据类型为 int32。 transcription 列的数据类型为 string。 normalized_transcript 列的数据类型为 string。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数，可以小于数据集总数。默认值：None，读取全部样本音频。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过`mindspore.dataset.config.set_num_parallel_workers()`配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，`num_samples`表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数，但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数，但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> lj_speech_dataset_dir = "/path/to/lj_speech_dataset_directory"
>>>
>>> # 1) Get all samples from LJSPEECH dataset in sequence
>>> dataset = ds.LJSpeechDataset(dataset_dir=lj_speech_dataset_dir, shuffle=False)
>>>
>>> # 2) Randomly select 350 samples from LJSPEECH dataset
>>> dataset = ds.LJSpeechDataset(dataset_dir=lj_speech_dataset_dir, num_samples=350,
↳ shuffle=True)
>>>
>>> # 3) Get samples from LJSPEECH dataset for shard 0 in a 2-way distributed training
>>> dataset = ds.LJSpeechDataset(dataset_dir=lj_speech_dataset_dir, num_shards=2, shard_id=0)
>>>
>>> # In LJSPEECH dataset, each dictionary has keys "waveform", "sample_rate", "transcription
↳ "
>>> # and "normalized_transcript"
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 421: 参数 `sampler` 和 `num_samples` , `shuffle` , `num_shards` , `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **LJSPEECH** 数据集：

LJSPEECH 是一个公共领域的语音数据集，由 13,100 个来自 7 部非小说类书籍的段落短音频片段组成。为每个剪辑片段都进行转录。剪辑的长度从 1 秒到 10 秒不等，总长度约为 24 小时。

这些被阅读的文本于 1884 年至 1964 年间出版，属于公共领域。这些音频由 LibriVox 项目于 2016-17 年录制。

以下是原始的 **LJSPEECH** 数据集结构。可以将数据集文件解压缩到以下目录结构中，并由 MindSpore 的 API 读取。

```
.
├── LJSpeech-1.1
│   ├── README
│   ├── metadata.csv
│   └── wavs
│       ├── LJ001-0001.wav
│       ├── LJ001-0002.wav
│       ├── LJ001-0003.wav
│       ├── LJ001-0004.wav
│       ├── LJ001-0005.wav
│       ├── LJ001-0006.wav
│       ├── LJ001-0007.wav
│       ├── LJ001-0008.wav
│       ├── ...
│       ├── LJ050-0277.wav
│       └── LJ050-0278.wav
```

引用：

```
@misc{lj_speech17,  
author      = {Keith Ito and Linda Johnson},  
title       = {The LJ Speech Dataset},  
howpublished = {url{https://keithito.com/LJ-Speech-Dataset}},  
year        = 2017  
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.SpeechCommandsDataset

```
class mindspore.dataset.SpeechCommandsDataset (dataset_dir, usage=None, num_samples=None,
                                              num_parallel_workers=None, shuffle=None, sampler=None,
                                              num_shards=None, shard_id=None, cache=None)
```

Speech Commands 数据集。

生成的数据集有五列 `[waveform, sample_rate, label, speaker_id, utterance_number]`。列 `waveform` 的数据类型为 `float32`。列 `sample_rate` 的数据类型为 `int32`。列 `label` 的数据类型为 `string`。列 `speaker_id` 的数据类型为 `string`。列 `utterance_number` 的数据类型为 `int32`。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **usage** (str, 可选) - 指定数据集的子集, 可取值为 `'train'`、`'test'`、`'valid'` 或 `'all'`。默认值: `None`，读取全部样本。取值为 `'train'` 时将会读取 84,843 个训练样本, 取值为 `'test'` 时将会读取 11,005 个测试样本, 取值为 `'valid'` 时将会读取 9,981 个测试样本, 取值为 `'all'` 时将会读取全部 105,829 个样本。默认值: `None`，读取全部样本。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: `None`，读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: `None`，使用全局默认线程数 (8)，也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: `None`。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: `None`。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: `None`。指定此参数后, `num_samples` 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: `None`。只有当指定了 `num_shards` 时才能指定此参数。
- **cache** (`DatasetCache`, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: `None`，不使用缓存。

异常：

- **RuntimeError** - `dataset_dir` 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **RuntimeError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **RuntimeError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **RuntimeError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> speech_commands_dataset_dir = "/path/to/speech_commands_dataset_directory"
>>>
>>> # Read 3 samples from SpeechCommands dataset
>>> dataset = ds.SpeechCommandsDataset(dataset_dir=speech_commands_dataset_dir, num_
↳ samples=3)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 428: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>None</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **SpeechCommands** 数据集：

SpeechCommands（语音命令）数据是用于有限词汇语音识别的数据集，包含 105,829 个 '.wav' 格式的音频样本。以下是原始 **SpeechCommands** 的数据集结构。可以将数据集文件解压缩成此目录结构，并由 MindSpore 的 API 读取。

```
.
├── speech_commands_dataset_dir
│   ├── cat
│   │   ├── b433eff_nohash_0.wav
│   │   ├── 5a33edf_nohash_1.wav
│   │   └── ....
│   ├── dog
│   │   ├── b433w2w_nohash_0.wav
│   │   └── ....
│   ├── four
│   └── ....
```

引用：

```
@article{2018Speech,
title={Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition},
author={Warden, P.},
year={2018}
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.TedliumDataset

```
class mindspore.dataset.TedliumDataset (dataset_dir, release, usage=None, extensions=None, num_samples=None,
                                         num_parallel_workers=None, shuffle=None, sampler=None,
                                         num_shards=None, shard_id=None, cache=None)
```

Tedlium 数据集。生成的数据集的列取决于源 SPH 文件和相应的 STM 文件。

生成的数据集有六列 [waveform, sample_rate, transcript, talk_id, speaker_id, identifier]。列 waveform 的数据类型为 float32, 列 sample_rate 的数据类型为 int32, 列 transcript、列 talk_id、列 speaker_id 和列 identifier 的数据类型为 string。

参数:

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **release** (str) - 指定数据集的发布版本, 可以取值为 'release1'、'release2' 或 'release3'。
- **usage** (str, 可选) - 指定数据集的子集。对于 release 为 'release1' 或 'release2', usage 可以是 'train'、'test'、'dev' 或 'all'。对于 release 为 'release3', usage 只能是 'all'。默认值: None, 读取全部样本。
- **extensions** (str, 可选) - 指定 SPH 文件的扩展名。默认值: None, 默认指定为 '.sph'。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值: None, 读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值: None, 使用全局默认线程数 (8), 也可以通过 `mindspore.dataset.config.set_num_parallel_workers()` 配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值: None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值: None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值: None。指定此参数后, num_samples 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值: None。只有当指定了 num_shards 时才能指定此参数。
- **cache** (DatasetCache, 可选) - 单节点数据缓存服务, 用于加快数据集处理, 详情请阅读 [单节点数据缓存](#)。默认值: None, 不使用缓存。

异常:

- **RuntimeError** - dataset_dir 路径下不包含任何数据文件。
- **RuntimeError** - 同时指定了 sampler 和 shuffle 参数。
- **RuntimeError** - 同时指定了 sampler 和 num_shards 参数或同时指定了 sampler 和 shard_id 参数。
- **RuntimeError** - 指定了 num_shards 参数, 但是未指定 shard_id 参数。
- **RuntimeError** - 指定了 shard_id 参数, 但是未指定 num_shards 参数。
- **ValueError** - num_parallel_workers 参数超过系统最大线程数。
- **ValueError** - 如果 shard_id 取值不在 [0, num_shards) 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> # 1) Get all train samples from TEDLIUM_release1 dataset in sequence.
>>> dataset = ds.TedliumDataset(dataset_dir="/path/to/tedlium1_dataset_directory",
...                             release="release1", shuffle=False)
>>>
>>> # 2) Randomly select 10 samples from TEDLIUM_release2 dataset.
```

(下页继续)

(续上页)

```
>>> dataset = ds.TedliumDataset(dataset_dir="/path/to/tedlium2_dataset_directory",
...                             release="release2", num_samples=10, shuffle=True)
>>>
>>> # 3) Get samples from TEDLIUM_release-3 dataset for shard 0 in a 2-way distributed
    ↪ training.
>>> dataset = ds.TedliumDataset(dataset_dir="/path/to/tedlium3_dataset_directory",
...                             release="release3", num_shards=2, shard_id=0)
>>>
>>> # In TEDLIUM dataset, each dictionary has keys : waveform, sample_rate, transcript, talk_
    ↪ id,
>>> # speaker_id and identifier.
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 435: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **TEDLIUM** 数据集：

TEDLIUM_release1 数据集：TED-LUM 语料库是英语 TED 演讲，有转录，采样频率为 16kHz。包含了大约 118 小时的演讲。

TEDLIUM_release2 数据集：这是 TED-LIUM 语料库版本 2，根据知识共享 BY-NC-ND 3.0 授权。所有会谈和文本均

为 TED 会议有限责任公司的财产。TED-LIUM 语料库由音频谈话及其转录组成，由 TED 网站提供。这些数据为训练声学模型参加 2011 年口语翻译国际研讨会（LIUM 英语/法语 SLT 系统在 SLT 任务中排名第一）而准备和过滤。

TEDLIUM_release-3 数据集：这是 TED-LIUM 语料库版本 3，根据知识共享 BY-NC-ND 3.0 授权。所有会谈和文本均为 TED 会议有限责任公司的财产。这个新的 TED-LIUM 版本是通过 Ubiquis 公司和 LIUM（法国勒芒大学）的合作发布的。

可以将数据集文件解压缩到以下目录结构中，并由 MindSpore 的 API 读取。

TEDLIUM release1 与 TEDLIUM release2 的结构相同，只是数据不同。

```
.
├── TEDLIUM_release1
│   ├── dev
│   │   ├── sph
│   │   │   ├── AlGore_2009.sph
│   │   │   └── BarrySchwartz_2005G.sph
│   │   └── stm
│   │       ├── AlGore_2009.stm
│   │       └── BarrySchwartz_2005G.stm
│   ├── test
│   │   ├── sph
│   │   │   ├── AimeeMullins_2009P.sph
│   │   │   └── BillGates_2010.sph
│   │   └── stm
│   │       ├── AimeeMullins_2009P.stm
│   │       └── BillGates_2010.stm
│   ├── train
│   │   ├── sph
│   │   │   ├── AaronHuey_2010X.sph
│   │   │   └── AdamGrosser_2007.sph
│   │   └── stm
│   │       ├── AaronHuey_2010X.stm
│   │       └── AdamGrosser_2007.stm
│   ├── readme
│   └── TEDLIUM.150k.dic
```

TEDLIUM release3 目录结构稍有不同。

```
.
├── TEDLIUM_release-3
│   ├── data
│   │   ├── ctl
│   │   ├── sph
│   │   │   ├── 911Mothers_2010W.sph
│   │   │   └── AalaElKhani.sph
│   │   └── stm
│   │       ├── 911Mothers_2010W.stm
│   │       └── AalaElKhani.stm
│   ├── doc
│   ├── legacy
│   ├── LM
│   ├── speaker-adaptation
│   ├── readme
│   └── TEDLIUM.150k.dic
```

引用：

```

@article{
  title={TED-LIUM: an automatic speech recognition dedicated corpus},
  author={A. Rousseau, P. Deléglise, Y. Estève},
  journal={Proceedings of the Eighth International Conference on Language Resources and
  Evaluation (LREC'12)},
  year={May 2012},
  biburl={https://www.openslr.org/7/}
}

@article{
  title={Enhancing the TED-LIUM Corpus with Selected Data for Language Modeling and More TED
  Talks},
  author={A. Rousseau, P. Deléglise, and Y. Estève},
  journal={Proceedings of the Eighth International Conference on Language Resources and
  Evaluation (LREC'12)},
  year={May 2014},
  biburl={https://www.openslr.org/19/}
}

@article{
  title={TED-LIUM 3: twice as much data and corpus repartition for experiments on speaker
  adaptation},
  author={François Hernandez, Vincent Nguyen, Sahar Ghannay, Natalia Tomashenko, and Yannick
  Estève},
  journal={the 20th International Conference on Speech and Computer (SPECOM 2018)},
  year={September 2018},
  biburl={https://www.openslr.org/51/}
}

```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 <i>epoch</i> 中的 <i>batch</i> 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 <i>repeat</i> 操作的次数。
<code>mindspore.dataset.Dataset.input_indexes</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 <i>shape</i> 。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

mindspore.dataset.YesNoDataset

class mindspore.dataset.**YesNoDataset** (*dataset_dir, num_samples=None, num_parallel_workers=None, shuffle=None, sampler=None, num_shards=None, shard_id=None, cache=None*)

YesNo 数据集。

生成的数据集有三列 [*waveform, sample_rate, labels*]。列 *waveform* 的数据类型为 float32。列 *sample_rate* 的数据类型为 int32。列 *labels* 的数据类型为 int32。

参数：

- **dataset_dir** (str) - 包含数据集文件的根目录路径。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取全部样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过`mindspore.dataset.config.set_num_parallel_workers()`配置全局线程数。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **sampler** (Sampler, 可选) - 指定从数据集中选取样本的采样器。默认值：None。下表中会展示不同配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 数据并行模式训练 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 单节点数据缓存。默认值：None，不使用缓存。

异常：

- **RuntimeError** - *dataset_dir* 路径下不包含任何数据文件。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **RuntimeError** - 同时指定了 *sampler* 和 *shuffle* 参数。
- **RuntimeError** - 同时指定了 *sampler* 和 *num_shards* 参数或同时指定了 *sampler* 和 *shard_id* 参数。
- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - 如果 *shard_id* 取值不在 [0, *num_shards*) 范围。

样例：

```
>>> import mindspore.dataset as ds
>>> yes_no_dataset_dir = "/path/to/yes_no_dataset_directory"
>>>
>>> # Read 3 samples from YesNo dataset
>>> dataset = ds.YesNoDataset(dataset_dir=yes_no_dataset_dir, num_samples=3)
>>>
>>> # Note: In YesNo dataset, each dictionary has keys "waveform", "sample_rate", "label"
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

说明：入参 `num_samples`、`shuffle`、`num_shards`、`shard_id` 可用于控制数据集所使用的采样器，其与入参 `sampler` 搭配使用的效果如下。

表 442: 参数 `sampler` 和 `num_samples`，`shuffle`，`num_shards`，`shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray</code> , <code>list</code> , <code>tuple</code> , <code>int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(indices = sampler , num_samples = num_samples)</code>
<code>iterable</code> 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>IndexSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

关于 **YesNo** 数据集：

Yesno 是一个音频数据集，由 60 个录音组成，由一个人用希伯来语说是或不是；每个录音都有 8 个字长。以下是原始的 **YesNo** 数据集结构。可以将数据集文件解压缩到此目录结构中，并由 MindSpore 的 API 读取。

```
.
├── yes_no_dataset_dir
│   ├── 1_1_0_0_1_1_0_0.wav
│   └── 1_0_0_0_1_1_0_0.wav
```

(下页继续)

(续上页)

```
|— 1_1_0_0_1_1_0_0.wav
|— ....
```

引用:

```
@NetworkResource{Kaldi_audio_project,
author      = {anonymous},
url         = "http://www.openslr.org/1/"
}
```

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩，多个 dataset 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

表 450: 参数 `sampler` 和 `num_samples`, `shuffle`, `num_shards`, `shard_id` 的不同组合得到的采样器

参数 <code>sampler</code>	参数 <code>num_shards</code> / <code>shard_id</code>	参数 <code>shuffle</code>	参数 <code>num_samples</code>	使用的采样器
<code>mind-spore.dataset.Sampler</code> 类型	<code>None</code>	<code>None</code>	<code>None</code>	sampler
<code>numpy.ndarray, list, tuple, int</code> 类型	<code>int</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(indices = sampler , num_samples = num_samples)</code>
iterable 类型	<code>/</code>	<code>/</code>	<code>num_samples</code>	<code>SubsetSampler(sampler = sampler , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = True , num_samples = num_samples)</code>
<code>None</code>	<code>num_shards</code> / <code>shard_id</code>	<code>False</code>	<code>num_samples</code>	<code>DistributedSampler(num_shards = num_shards , shard_id = shard_id , shuffle = False , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>None</code>	<code>RandomSampler(num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>None</code> / <code>True</code>	<code>num_samples</code>	<code>RandomSampler(replacement = True , num_samples = num_samples)</code>
<code>None</code>	<code>None</code>	<code>False</code>	<code>num_samples</code>	<code>SequentialSampler(num_samples = num_samples)</code>

异常:

- **RuntimeError** - `column_names` 列表的长度与数据的输出列表长度不匹配。
- **ValueError** - `num_parallel_workers` 参数超过系统最大线程数。
- **ValueError** - 同时指定了 `sampler` 和 `shuffle` 参数。
- **ValueError** - 同时指定了 `sampler` 和 `num_shards` 参数或同时指定了 `sampler` 和 `shard_id` 参数。
- **ValueError** - 指定了 `num_shards` 参数, 但是未指定 `shard_id` 参数。
- **ValueError** - 指定了 `shard_id` 参数, 但是未指定 `num_shards` 参数。
- **ValueError** - 如果 `shard_id` 取值不在 `[0, num_shards)` 范围。

样例:

```
>>> import mindspore.dataset as ds
>>> # 1) Input data can be a list
>>> data = [1, 2, 3]
>>> dataset = ds.NumpySlicesDataset(data=data, column_names=["column_1"])
>>>
>>> # 2) Input data can be a dictionary, and column_names will be its keys
>>> data = {"a": [1, 2], "b": [3, 4]}
>>> dataset = ds.NumpySlicesDataset(data=data)
>>>
>>> # 3) Input data can be a tuple of lists (or NumPy arrays), each tuple element refers to
↳data in each column
>>> data = ([1, 2], [3, 4], [5, 6])
```

(下页继续)

(续上页)

```
>>> dataset = ds.NumpySlicesDataset(data=data, column_names=["column_1", "column_2", "column_
↪3"])
>>>
>>> # 4) Load data from CSV file
>>> import pandas as pd
>>> df = pd.read_csv(filepath_or_buffer=csv_dataset_dir[0])
>>> dataset = ds.NumpySlicesDataset(data=dict(df), shuffle=False)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器, 保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串, 如果提供了文件名, 则转储到文件中。

12.6.2 mindspore.dataset.PaddedDataset

class mindspore.dataset.**PaddedDataset** (*padded_samples*)
由用户提供的填充数据构建数据集。可用于在分布式训练时给原始数据集添加样本，使数据集样本能平均分配给不同的分片。

参数：

- **padded_samples** (list(dict)) - 用户提供的样本数据。

异常：

- **TypeError** - *padded_samples* 的类型不为 list。
- **TypeError** - *padded_samples* 的元素类型不为 dict。
- **ValueError** - *padded_samples* 为空的 list。

样例：

```
>>> import mindspore.dataset as ds
>>> import numpy as np
>>> data = [{'image': np.zeros(1, np.uint8)}, {'image': np.zeros(2, np.uint8)}]
>>> dataset = ds.PaddedDataset(padded_samples=data)
```

教程样例：

- 使用数据 Pipeline 加载 & 处理数据集

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 <i>dataset</i> 对象按列进行合并压缩，多个 <i>dataset</i> 对象不能有相同的列名。

Batch (批操作)

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据, 并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器, 返回字典形式的样本, 其中键为列名, 值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器, 返回列表形式的样本, 其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小, 即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引, 它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器, 保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

12.6.3 mindspore.dataset.RandomDataset

```
class mindspore.dataset.RandomDataset (total_rows=None, schema=None, columns_list=None, num_samples=None,
                                         num_parallel_workers=None, cache=None, shuffle=None,
                                         num_shards=None, shard_id=None)
```

生成随机数据的源数据集。

参数：

- **total_rows** (int, 可选) - 随机生成样本数据的数量。默认值：None，生成随机数量的样本。
- **schema** (Union[str, *Schema*], 可选) - 数据格式策略，用于指定读取数据列的数据类型、数据维度等信息。支持传入 JSON 文件路径或*mindspore.dataset.Schema* 构造的对象。默认值：None。
- **columns_list** (list[str], 可选) - 指定生成数据集的列名。默认值：None，生成的数据列将以“c0”、“c1”、“c2”...“cn”的规则命名。
- **num_samples** (int, 可选) - 指定从数据集中读取的样本数。默认值：None，读取所有样本。
- **num_parallel_workers** (int, 可选) - 指定读取数据的工作线程数。默认值：None，使用全局默认线程数 (8)，也可以通过*mindspore.dataset.config.set_num_parallel_workers()* 配置全局线程数。
- **cache** (*DatasetCache*, 可选) - 单节点数据缓存服务，用于加快数据集处理，详情请阅读 [单节点数据缓存](#)。默认值：None，不使用缓存。
- **shuffle** (bool, 可选) - 是否混洗数据集。默认值：None。下表中会展示不同参数配置的预期行为。
- **num_shards** (int, 可选) - 指定分布式训练时将数据集进行划分的分片数。默认值：None。指定此参数后，*num_samples* 表示每个分片的最大样本数。一般在 [数据并行模式训练](#) 的时候使用。
- **shard_id** (int, 可选) - 指定分布式训练时使用的分片 ID 号。默认值：None。只有当指定了 *num_shards* 时才能指定此参数。

异常：

- **RuntimeError** - 指定了 *num_shards* 参数，但是未指定 *shard_id* 参数。
- **RuntimeError** - 指定了 *shard_id* 参数，但是未指定 *num_shards* 参数。
- **ValueError** - *num_parallel_workers* 参数超过系统最大线程数。
- **ValueError** - 如果 *shard_id* 取值不在 [0, *num_shards*) 范围。
- **TypeError** - *total_rows* 的类型不是 int。
- **TypeError** - *num_shards* 的类型不是 int。
- **TypeError** - *num_parallel_workers* 的类型不是 int。
- **TypeError** - *shuffle* 的类型不是 bool。
- **TypeError** - *columns_list* 的类型不是 list。

样例：


```
>>> from mindspore import dtype as mstype
>>> import mindspore.dataset as ds
>>>
>>> schema = ds.Schema()
>>> schema.add_column('image', de_type=mstype.uint8, shape=[2])
>>> schema.add_column('label', de_type=mstype.uint8, shape=[1])
>>> # apply dataset operations
>>> ds1 = ds.RandomDataset(schema=schema, total_rows=50, num_parallel_workers=4)
```

教程样例:

- 使用数据 Pipeline 加载 & 处理数据集

预处理操作

<code>mindspore.dataset.Dataset.apply</code>	对数据集对象执行给定操作函数。
<code>mindspore.dataset.Dataset.concat</code>	对传入的多个数据集对象进行拼接操作。
<code>mindspore.dataset.Dataset.filter</code>	通过自定义判断条件对数据集对象中的数据进行过滤。
<code>mindspore.dataset.Dataset.flat_map</code>	对数据集对象中每一条数据执行给定的数据处理，并将结果展平。
<code>mindspore.dataset.Dataset.map</code>	给定一组数据增强列表，按顺序将数据增强作用在数据集对象上。
<code>mindspore.dataset.Dataset.project</code>	从数据集对象中选择需要的列，并按给定的列名的顺序进行排序。
<code>mindspore.dataset.Dataset.rename</code>	对数据集对象按指定的列名进行重命名。
<code>mindspore.dataset.Dataset.repeat</code>	重复此数据集 <i>count</i> 次。
<code>mindspore.dataset.Dataset.reset</code>	重置下一个 <i>epoch</i> 的数据集对象。
<code>mindspore.dataset.Dataset.save</code>	将数据处理管道中正处理的数据保存为通用的数据集格式。
<code>mindspore.dataset.Dataset.shuffle</code>	通过创建 <i>buffer_size</i> 大小的缓存来混洗该数据集。
<code>mindspore.dataset.Dataset.skip</code>	跳过此数据集对象的前 <i>count</i> 条数据。
<code>mindspore.dataset.Dataset.split</code>	将数据集拆分为多个不重叠的子数据集。
<code>mindspore.dataset.Dataset.take</code>	截取数据集的前指定条数据。
<code>mindspore.dataset.Dataset.zip</code>	将多个 dataset 对象按列进行合并压缩，多个 dataset 对象不能有相同的列名。

Batch（批操作）

<code>mindspore.dataset.Dataset.batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>per_batch_map</i> 指定组合前要进行的预处理操作。
<code>mindspore.dataset.Dataset.bucket_batch_by_length</code>	根据数据的长度进行分桶。
<code>mindspore.dataset.Dataset.padded_batch</code>	将数据集中连续 <i>batch_size</i> 条数据组合为一个批数据，并可通过可选参数 <i>pad_info</i> 预先将样本补齐。

迭代器

<code>mindspore.dataset.Dataset.create_dict_iterator</code>	创建数据集迭代器，返回字典形式的样本，其中键为列名，值为数据。
<code>mindspore.dataset.Dataset.create_tuple_iterator</code>	创建数据集迭代器，返回列表形式的样本，其中的元素为各列数据。

数据集属性

<code>mindspore.dataset.Dataset.get_batch_size</code>	获得数据集对象定义的批处理大小，即一个批处理数据中包含的数据条数。
<code>mindspore.dataset.Dataset.get_class_indexing</code>	获取类别名称到类别索引的映射字典。
<code>mindspore.dataset.Dataset.get_col_names</code>	返回数据集对象中包含的列名。
<code>mindspore.dataset.Dataset.get_dataset_size</code>	返回一个 epoch 中的 batch 数。
<code>mindspore.dataset.Dataset.get_repeat_count</code>	获取 <i>RepeatDataset</i> 中定义的 repeat 操作的次数。
<code>mindspore.dataset.Dataset.input_indexs</code>	获取/设置数据列索引，它表示使用下沉模式时数据列映射至网络中的对应关系。
<code>mindspore.dataset.Dataset.num_classes</code>	获取数据集对象中所有样本的类别数目。
<code>mindspore.dataset.Dataset.output_shapes</code>	获取数据集对象中每列数据的 shape。
<code>mindspore.dataset.Dataset.output_types</code>	获取数据集对象中每列数据的数据类型。

应用采样方法

<code>mindspore.dataset.MappableDataset.add_sampler</code>	为当前数据集添加子采样器。
<code>mindspore.dataset.MappableDataset.use_sampler</code>	替换当前数据集的最末子采样器，保持父采样器不变。

其他方法

<code>mindspore.dataset.Dataset.sync_update</code>	释放阻塞条件并使用给定数据触发回调函数。
<code>mindspore.dataset.Dataset.sync_wait</code>	为同步操作在数据集对象上添加阻塞条件。
<code>mindspore.dataset.Dataset.to_json</code>	将数据处理管道序列化为 JSON 字符串，如果提供了文件名，则转储到文件中。

12.7 采样器

<code>mindspore.dataset.DistributedSampler</code>	分布式采样器，将数据集进行分片用于分布式训练。
<code>mindspore.dataset.PKSampler</code>	为数据集中每 P 个类别各采样 K 个样本。
<code>mindspore.dataset.RandomSampler</code>	随机采样器。
<code>mindspore.dataset.SequentialSampler</code>	按数据集的读取顺序采样数据集样本，相当于不使用采样器。
<code>mindspore.dataset.SubsetRandomSampler</code>	给定样本的索引序列，从序列中随机获取索引对数据集进行采样。
<code>mindspore.dataset.SubsetSampler</code>	给定样本的索引序列，对数据集采样指定索引的样本。
<code>mindspore.dataset.WeightedRandomSampler</code>	给定样本的权重列表，根据权重决定样本的采样概率，随机采样 <code>[0, len(weights) - 1]</code> 中的样本。

12.7.1 mindspore.dataset.DistributedSampler

class mindspore.dataset.DistributedSampler (num_shards, shard_id, shuffle=True, num_samples=None, offset=-1)
分布式采样器，将数据集进行分片用于分布式训练。

说明：不同数据集的混洗模式支持如下：

表 470: 混洗模式支持

混洗模式	MindDataset	TFRecordDataset	其他数据集
Shuffle.ADAPTIVE	支持	不支持	不支持
Shuffle.GLOBAL	支持	支持	支持
Shuffle.PARTIAL	支持	不支持	不支持
Shuffle.FILES	支持	支持	不支持
Shuffle.INFILE	支持	不支持	不支持

参数：

- num_shards** (int) - 数据集分片数量。
- shard_id** (int) - 当前分片的分片 ID，应在 `[0, num_shards-1]` 范围内。
- shuffle** (Union[bool, Shuffle], 可选) - 是否混洗采样得到的样本。默认值：True，采用 `mindspore.dataset.Shuffle.GLOBAL` 混洗样本。如果 `shuffle` 为 False，则不混洗。通过传入枚举变量设置数据混洗的模式，枚举变量参考链接[mindspore.dataset.Shuffle](#)：
 - `Shuffle.ADAPTIVE`：当数据集样本小于等于 1 亿时，采用 `Shuffle.GLOBAL`，当大于 1 亿时，采用局部 `Shuffle.PARTIAL`，每 100 万样本混洗一次。
 - `Shuffle.GLOBAL`：执行全局混洗，一次性混洗数据集中所有样本。
 - `Shuffle.PARTIAL`：执行局部混洗，每 100 万个样本混洗一次。
 - `Shuffle.FILES`：仅混洗文件序列，不混洗文件中的数据。
 - `Shuffle.INFILE`：保持读入文件的序列，仅混洗每个文件中的数据。
- num_samples** (int, 可选) - 获取的样本数，可用于获取部分采样得到的样本。默认值：None，获取采样到的所有样本。

- **offset** (int, 可选) - 分布式采样结果进行分配时的起始分片 ID 号, 值不能大于参数 *num_shards*。从不同的分片 ID 开始分配数据可能会影响每个分片的最终样本数。仅当 `ConcatDataset` 以 `mindspore.dataset.DistributedSampler` 为采样器时, 此参数才有效。默认值: -1, 每个分片具有相同的样本数。

异常:

- **TypeError** - *num_shards* 的类型不是 int。
- **TypeError** - *shard_id* 的类型不是 int。
- **TypeError** - *shuffle* 的类型不是 bool 或者 `Shuffle`。
- **TypeError** - *num_samples* 的类型不是 int。
- **TypeError** - *offset* 的类型不是 int。
- **ValueError** - *num_samples* 为负值。
- **RuntimeError** - *num_shards* 不是正值。
- **RuntimeError** - *shard_id* 小于 0 或大于等于 *num_shards*。
- **RuntimeError** - *offset* 大于 *num_shards*。

样例:

```
>>> import mindspore.dataset as ds
>>> # creates a distributed sampler with 10 shards in total. This shard is shard 5.
>>> sampler = ds.DistributedSampler(10, 5)
>>> dataset = ds.ImageFolderDataset(image_folder_dataset_dir,
...                                num_parallel_workers=8,
...                                sampler=sampler)
```

add_child(sampler)

为给定采样器添加子采样器。父采样器接收子采样器输出数据作为输入, 并应用其采样逻辑返回新的采样结果。

说明: 被添加的子 `sampler` 如果有 *shuffle* 属性, 其值不能是 `Shuffle.PARTIAL`, 且父 `sampler` 的 *shuffle* 属性值必须是 `Shuffle.GLOBAL`。

参数:

- **sampler** (Sampler) - 用于从数据集中选择样本的对象。仅支持内置采样器 (`mindspore.dataset.DistributedSampler`、`mindspore.dataset.PKSampler`、`mindspore.dataset.RandomSampler`、`mindspore.dataset.SequentialSampler`、`mindspore.dataset.SubsetRandomSampler`、`mindspore.dataset.WeightedRandomSampler`)。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=4))
>>> dataset = ds.Cifar10Dataset(cifar10_dataset_dir, sampler=sampler)
```

get_child()

获取给定采样器的子采样器。

返回:

`Sampler`, 给定采样器的子采样器。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=2))
>>> child_sampler = sampler.get_child()
```

get_num_samples()

获取当前采样器实例的 *num_samples* 参数值。此参数在定义 **Sampler** 时, 可以选择性传入 (默认为 `None`)。此方法将返回 *num_samples* 的值。如果当前采样器有子采样器, 会继续访问子采样器, 并根据一定的规则处理获取值。

下表显示了各种可能的组合, 以及最终返回的结果。

子采样器	num_samples	child_samples	结果
T	x	y	min(x, y)
T	x	None	x
T	None	y	y
T	None	None	None
None	x	n/a	x
None	None	n/a	None

返回:

`int`, 样本数, 可为 `None`。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> num_samplers = sampler.get_num_samples()
```

12.7.2 mindspore.dataset.PKSampler

`class mindspore.dataset.PKSampler (num_val, num_class=None, shuffle=False, class_column='label', num_samples=None)`

为数据集中每 **P** 个类别各采样 **K** 个样本。

参数:

- **num_val** (`int`) - 每个类要采样的元素数量。
- **num_class** (`int`, 可选) - 要采样的类数量。默认值为 `None`, 采样所有类。当前不支持指定该参数。
- **shuffle** (`bool`, 可选) - 是否混洗采样得到的样本。默认值: `False`, 不混洗样本。
- **class_column** (`str`, 可选) - 指定 `label` 所属数据列的名称, 将基于此列作为数据标签进行采样。默认值: `'label'`。
- **num_samples** (`int`, 可选) - 获取的样本数, 可用于部分获取采样得到的样本。默认值: `None`, 获取采样到的所有样本。

异常:

- **TypeError** - *shuffle* 的类型不是 `bool`。
- **TypeError** - *class_column* 的类型不是 `str`。

- **TypeError** - `num_samples` 的类型不是 `int`。
- **NotImplementedError** - `num_class` 不为 `None`。
- **RuntimeError** - `num_val` 不是正值。
- **ValueError** - `num_samples` 为负值。

样例:

```
>>> import mindspore.dataset as ds
>>> # creates a PKSampler that will get 3 samples from every class.
>>> sampler = ds.PKSampler(3)
>>> dataset = ds.ImageFolderDataset(image_folder_dataset_dir,
...                                num_parallel_workers=8,
...                                sampler=sampler)
```

add_child(sampler)

为给定采样器添加子采样器。父采样器接收子采样器输出数据作为输入，并应用其采样逻辑返回新的采样结果。

说明：被添加的子 `sampler` 如果有 `shuffle` 属性，其值不能是 `Shuffle.PARTIAL`，且父 `sampler` 的 `shuffle` 属性值必须是 `Shuffle.GLOBAL`。

参数:

- **sampler** (`Sampler`) - 用于从数据集中选择样本的对象。仅支持内置采样器 (`mindspore.dataset.DistributedSampler`、`mindspore.dataset.PKSampler`、`mindspore.dataset.RandomSampler`、`mindspore.dataset.SequentialSampler`、`mindspore.dataset.SubsetRandomSampler`、`mindspore.dataset.WeightedRandomSampler`)。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=4))
>>> dataset = ds.Cifar10Dataset(cifar10_dataset_dir, sampler=sampler)
```

get_child()

获取给定采样器的子采样器。

返回:

`Sampler`，给定采样器的子采样器。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=2))
>>> child_sampler = sampler.get_child()
```

get_num_samples()

获取当前采样器实例的 `num_samples` 参数值。此参数在定义 `Sampler` 时，可以选择性传入（默认为 `None`）。此方法将返回 `num_samples` 的值。如果当前采样器有子采样器，会继续访问子采样器，并根据一定的规则处理获取值。

下表显示了各种可能的组合，以及最终返回的结果。

子采样器	num_samples	child_samples	结果
T	x	y	min(x, y)
T	x	None	x
T	None	y	y
T	None	None	None
None	x	n/a	x
None	None	n/a	None

返回：
int，样本数，可为 None。

样例：

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> num_samples = sampler.get_num_samples()
```

12.7.3 mindspore.dataset.RandomSampler

class mindspore.dataset.**RandomSampler** (replacement=False, num_samples=None, shuffle=Shuffle.GLOBAL)
随机采样器。

说明：不同数据集的混洗模式支持如下：

表 471: 混洗模式支持

混洗模式	MindDataset	TFRecordDataset	其他数据集
Shuffle.ADAPTIVE	支持	不支持	不支持
Shuffle.GLOBAL	支持	支持	支持
Shuffle.PARTIAL	支持	不支持	不支持
Shuffle.FILES	支持	支持	不支持
Shuffle.INFILE	支持	不支持	不支持

参数：

- **replacement** (bool, 可选) - 是否将样本 ID 放回下一次采样。默认值：False，无放回采样。
- **num_samples** (int, 可选) - 获取的样本数，可用于部分获取采样得到的样本。默认值：None，获取采样到的所有样本。
- **shuffle** (Shuffle, 可选) - 采用何种混洗逻辑。默认值：Shuffle.GLOBAL，全局混洗。通过传入枚举变量设置数据混洗的模式，枚举变量参考链接[mindspore.dataset.Shuffle](#)：
 - Shuffle.ADAPTIVE：当数据集样本小于等于 1 亿时，采用 Shuffle.GLOBAL，当大于 1 亿时，采用局部 Shuffle.PARTIAL，每 100 万样本混洗一次。
 - Shuffle.GLOBAL：执行全局混洗，一次性混洗数据集中所有样本。占用内存大。
 - Shuffle.PARTIAL：执行局部混洗，每 100 万个样本混洗一次。占用内存小于 Shuffle.GLOBAL。
 - Shuffle.FILES：仅混洗文件序列，不混洗文件中的数据。
 - Shuffle.INFILE：保持读入文件的序列，仅混洗每个文件中的数据。

异常:

- **TypeError** - *replacement* 不是 bool 值。
- **TypeError** - *num_samples* 不是整数值。
- **ValueError** - *num_samples* 为负值。
- **TypeError** - *shuffle* 的类型不是 `Shuffle`。

样例:

```
>>> import mindspore.dataset as ds
>>> # creates a RandomSampler
>>> sampler = ds.RandomSampler()
>>> dataset = ds.ImageFolderDataset(image_folder_dataset_dir,
...                                 num_parallel_workers=8,
...                                 sampler=sampler)
```

`add_child(sampler)`

为给定采样器添加子采样器。父采样器接收子采样器输出数据作为输入，并应用其采样逻辑返回新的采样结果。

说明: 被添加的子 `sampler` 如果有 *shuffle* 属性, 其值不能是 `Shuffle.PARTIAL`, 且父 `sampler` 的 *shuffle* 属性值必须是 `Shuffle.GLOBAL`。

参数:

- **sampler** (`Sampler`) - 用于从数据集中选择样本的对象。仅支持内置采样器 (`mindspore.dataset.DistributedSampler`、`mindspore.dataset.PKSampler`、`mindspore.dataset.RandomSampler`、`mindspore.dataset.SequentialSampler`、`mindspore.dataset.SubsetRandomSampler`、`mindspore.dataset.WeightedRandomSampler`)。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=4))
>>> dataset = ds.Cifar10Dataset(cifar10_dataset_dir, sampler=sampler)
```

`get_child()`

获取给定采样器的子采样器。

返回:

`Sampler`, 给定采样器的子采样器。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=2))
>>> child_sampler = sampler.get_child()
```

`get_num_samples()`

获取当前采样器实例的 *num_samples* 参数值。此参数在定义 `Sampler` 时, 可以选择性传入 (默认为 `None`)。此方法将返回 *num_samples* 的值。如果当前采样器有子采样器, 会继续访问子采样器, 并根据一定的规则处理获取值。

下表显示了各种可能的组合，以及最终返回的结果。

子采样器	num_samples	child_samples	结果
T	x	y	min(x, y)
T	x	None	x
T	None	y	y
T	None	None	None
None	x	n/a	x
None	None	n/a	None

返回：

int，样本数，可为 None。

样例：

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> num_samplers = sampler.get_num_samples()
```

12.7.4 mindspore.dataset.SequentialSampler

class mindspore.dataset.**SequentialSampler** (*start_index=None, num_samples=None*)

按数据集的读取顺序采样数据集样本，相当于不使用采样器。

参数：

- **start_index** (int, 可选) - 采样的起始样本 ID。默认值：None，从数据集第一个样本开始采样。
- **num_samples** (int, 可选) - 获取的样本数，可用于获取部分采样得到的样本。默认值：None，获取采样到的所有样本。

异常：

- **TypeError** - *start_index* 的类型不是 int。
- **TypeError** - *num_samples* 的类型不是 int。
- **RuntimeError** - *start_index* 为负值。
- **ValueError** - *num_samples* 为负值。

样例：

```
>>> import mindspore.dataset as ds
>>> # creates a SequentialSampler
>>> sampler = ds.SequentialSampler()
>>> dataset = ds.ImageFolderDataset(image_folder_dataset_dir,
...                                num_parallel_workers=8,
...                                sampler=sampler)
```

add_child (*sampler*)

为给定采样器添加子采样器。父采样器接收子采样器输出数据作为输入，并应用其采样逻辑返回新的采样结果。

说明：被添加的子 *sampler* 如果有 *shuffle* 属性，其值不能是 *Shuffle.PARTIAL*，且父 *sampler* 的 *shuffle* 属性值必须是 *Shuffle.GLOBAL*。

参数:

- **sampler** (Sampler) - 用于从数据集中选择样本的对象。仅支持内置采样器 (*mindspore.dataset.DistributedSampler*、*mindspore.dataset.PKSampler*、*mindspore.dataset.RandomSampler*、*mindspore.dataset.SequentialSampler*、*mindspore.dataset.SubsetRandomSampler*、*mindspore.dataset.WeightedRandomSampler*)。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=4))
>>> dataset = ds.Cifar10Dataset(cifar10_dataset_dir, sampler=sampler)
```

get_child()

获取给定采样器的子采样器。

返回:

Sampler, 给定采样器的子采样器。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=2))
>>> child_sampler = sampler.get_child()
```

get_num_samples()

获取当前采样器实例的 *num_samples* 参数值。此参数在定义 Sampler 时, 可以选择性传入 (默认为 None)。此方法将返回 *num_samples* 的值。如果当前采样器有子采样器, 会继续访问子采样器, 并根据一定的规则处理获取值。

下表显示了各种可能的组合, 以及最终返回的结果。

子采样器	num_samples	child_samples	结果
T	x	y	min(x, y)
T	x	None	x
T	None	y	y
T	None	None	None
None	x	n/a	x
None	None	n/a	None

返回:

int, 样本数, 可为 None。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> num_samples = sampler.get_num_samples()
```

12.7.5 mindspore.dataset.SubsetRandomSampler

class mindspore.dataset.SubsetRandomSampler (indices, num_samples=None)

给定样本的索引序列，从序列中随机获取索引对数据集进行采样。

参数：

- **indices** (Iterable) - 样本索引的序列（除了 string 类型外的任意 Python 可迭代对象类型）。
- **num_samples** (int, 可选) - 获取的样本数，可用于部分获取采样得到的样本。默认值：None，获取采样到的所有样本。

异常：

- **TypeError** - indices 的类型不是 int。
- **TypeError** - num_samples 的类型不是 int。
- **ValueError** - num_samples 为负值。

样例：

```
>>> import mindspore.dataset as ds
>>> indices = [0, 1, 2, 3, 7, 88, 119]
>>>
>>> # create a SubsetRandomSampler, will sample from the provided indices
>>> sampler = ds.SubsetRandomSampler(indices)
>>> data = ds.ImageFolderDataset(image_folder_dataset_dir, num_parallel_workers=8,
↳ sampler=sampler)
```

add_child (sampler)

为给定采样器添加子采样器。父采样器接收子采样器输出数据作为输入，并应用其采样逻辑返回新的采样结果。

说明：被添加的子 sampler 如果有 *shuffle* 属性，其值不能是 `Shuffle.PARTIAL`，且父 sampler 的 *shuffle* 属性值必须是 `Shuffle.GLOBAL`。

参数：

- **sampler** (Sampler) - 用于从数据集中选择样本的对象。仅支持内置采样器 (*mindspore.dataset.DistributedSampler*、*mindspore.dataset.PKSampler*、*mindspore.dataset.RandomSampler*、*mindspore.dataset.SequentialSampler*、*mindspore.dataset.SubsetRandomSampler*、*mindspore.dataset.WeightedRandomSampler*)。

样例：

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=4))
>>> dataset = ds.Cifar10Dataset(cifar10_dataset_dir, sampler=sampler)
```

get_child ()

获取给定采样器的子采样器。

返回：

Sampler，给定采样器的子采样器。

样例：

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=2))
>>> child_sampler = sampler.get_child()
```

12.7.6 mindspore.dataset.SubsetSampler

class mindspore.dataset.**SubsetSampler** (*indices, num_samples=None*)

给定样本的索引序列，对数据集采样指定索引的样本。

参数：

- **indices** (Iterable) - 索引的序列（包括除了 `string` 类型的任意 Python 可迭代对象类型）。
- **num_samples** (int, 可选) - 获取的样本数，可用于获取部分采样得到的样本。默认值：None，获取采样到的所有样本。

异常：

- **TypeError** - *indices* 的类型不是 int。
- **TypeError** - *num_samples* 的类型不是 int。
- **ValueError** - *num_samples* 为负值。

样例：

```
>>> import mindspore.dataset as ds
>>> indices = [0, 1, 2, 3, 4, 5]
>>>
>>> # creates a SubsetSampler, will sample from the provided indices
>>> sampler = ds.SubsetSampler(indices)
>>> dataset = ds.ImageFolderDataset(image_folder_dataset_dir,
...                                num_parallel_workers=8,
...                                sampler=sampler)
```

add_child (*sampler*)

为给定采样器添加子采样器。父采样器接收子采样器输出数据作为输入，并应用其采样逻辑返回新的采样结果。

说明：被添加的子 `sampler` 如果有 `shuffle` 属性，其值不能是 `Shuffle.PARTIAL`，且父 `sampler` 的 `shuffle` 属性值必须是 `Shuffle.GLOBAL`。

参数：

- **sampler** (Sampler) - 用于从数据集中选择样本的对象。仅支持内置采样器 (`mindspore.dataset.DistributedSampler`、`mindspore.dataset.PKSampler`、`mindspore.dataset.RandomSampler`、`mindspore.dataset.SequentialSampler`、`mindspore.dataset.SubsetRandomSampler`、`mindspore.dataset.WeightedRandomSampler`)。

样例：

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=4))
>>> dataset = ds.Cifar10Dataset(cifar10_dataset_dir, sampler=sampler)
```

get_child()

获取给定采样器的子采样器。

返回：

Sampler，给定采样器的子采样器。

样例：

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=2))
>>> child_sampler = sampler.get_child()
```

12.7.7 mindspore.dataset.WeightedRandomSampler

class mindspore.dataset.**WeightedRandomSampler**(weights, num_samples=None, replacement=True)

给定样本的权重列表，根据权重决定样本的采样概率，随机采样 [0, len(weights) - 1] 中的样本。

参数：

- **weights** (list[float, int]) - 权重序列，总和不一定为 1。
- **num_samples** (int, 可选) - 获取的样本数，可用于部分获取采样得到的样本。默认值：None，获取采样到的所有样本。
- **replacement** (bool, 可选) - 是否将样本 ID 放回供下一次采样。默认值：True，放回样本 ID。

异常：

- **TypeError** - weights 元素的类型不是数值类型。
- **TypeError** - num_samples 的类型不是 int。
- **TypeError** - replacement 的类型不是 bool。
- **RuntimeError** - weights 为空或全为零。
- **ValueError** - num_samples 为负值。

样例：

```
>>> import mindspore.dataset as ds
>>> weights = [0.9, 0.01, 0.4, 0.8, 0.1, 0.1, 0.3]
>>>
>>> # creates a WeightedRandomSampler that will sample 4 elements without replacement
>>> sampler = ds.WeightedRandomSampler(weights, 4)
>>> dataset = ds.ImageFolderDataset(image_folder_dataset_dir,
...                               num_parallel_workers=8,
...                               sampler=sampler)
```

add_child(sampler)

为给定采样器添加子采样器。父采样器接收子采样器输出数据作为输入，并应用其采样逻辑返回新的采样结果。

说明：被添加的子 sampler 如果有 *shuffle* 属性，其值不能是 `Shuffle.PARTIAL`，且父 sampler 的 *shuffle* 属性值必须是 `Shuffle.GLOBAL`。

参数：

- **sampler** (Sampler) - 用于从数据集中选择样本的对象。仅支持内置采样器 (*mindspore.dataset.DistributedSampler*、*mindspore.dataset.PKSampler*、*mindspore.dataset.RandomSampler*、*mindspore.dataset.SequentialSampler*、*mindspore.dataset.SubsetRandomSampler*、*mindspore.dataset.WeightedRandomSampler*)。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=4))
>>> dataset = ds.Cifar10Dataset(cifar10_dataset_dir, sampler=sampler)
```

get_child()

获取给定采样器的子采样器。

返回:

Sampler, 给定采样器的子采样器。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> sampler.add_child(ds.RandomSampler(num_samples=2))
>>> child_sampler = sampler.get_child()
```

get_num_samples()

获取当前采样器实例的 *num_samples* 参数值。此参数在定义 **Sampler** 时, 可以选择性传入 (默认为 *None*)。此方法将返回 *num_samples* 的值。如果当前采样器有子采样器, 会继续访问子采样器, 并根据一定的规则处理获取值。

下表显示了各种可能的组合, 以及最终返回的结果。

子采样器	num_samples	child_samples	结果
T	x	y	min(x, y)
T	x	None	x
T	None	y	y
T	None	None	None
None	x	n/a	x
None	None	n/a	None

返回:

int, 样本数, 可为 *None*。

样例:

```
>>> import mindspore.dataset as ds
>>> sampler = ds.SequentialSampler(start_index=0, num_samples=3)
>>> num_samplers = sampler.get_num_samples()
```

12.8 配置

config 模块能够设置或获取数据处理管道的全局配置参数。

<code>mindspore.dataset.config.set_sending_batches</code>	设置 Host 向 Device 发送数据的批数上限。
<code>mindspore.dataset.config.load</code>	根据文件内容加载项目配置文件。
<code>mindspore.dataset.config.set_seed</code>	设置数据处理中随机数生成器的种子。
<code>mindspore.dataset.config.get_seed</code>	获取随机数的种子。
<code>mindspore.dataset.config.set_prefetch_size</code>	设置流水线中各个数据处理操作的缓冲队列大小。
<code>mindspore.dataset.config.get_prefetch_size</code>	获取数据处理管道的输出缓存队列长度。
<code>mindspore.dataset.config.set_num_parallel_workers</code>	为并行工作线程数量设置新的全局配置默认值。
<code>mindspore.dataset.config.get_num_parallel_workers</code>	获取并行工作线程数量的全局配置，将作用于每个操作的并行工作线程数量。
<code>mindspore.dataset.config.set_numa_enable</code>	设置 NUMA 的默认状态为启动状态。
<code>mindspore.dataset.config.get_numa_enable</code>	获取 NUMA 的启动/禁用状态。
<code>mindspore.dataset.config.set_monitor_sampling_interval</code>	设置监测采样的默认间隔时间（毫秒）。
<code>mindspore.dataset.config.get_monitor_sampling_interval</code>	获取性能监控采样时间间隔的全局配置。
<code>mindspore.dataset.config.set_callback_timeout</code>	为 <code>mindspore.dataset.WaitedDSCallback</code> 设置的默认超时时间（秒）。
<code>mindspore.dataset.config.get_callback_timeout</code>	获取 <code>mindspore.dataset.WaitedDSCallback</code> 的默认超时时间。
<code>mindspore.dataset.config.set_auto_num_workers</code>	自动为每个数据集操作设置并行线程数量（默认情况下，此功能关闭）。
<code>mindspore.dataset.config.get_auto_num_workers</code>	获取当前是否开启自动线程调整。
<code>mindspore.dataset.config.set_enable_shared_mem</code>	设置是否在开启数据处理多进程时使用共享内存进行进程间通信。
<code>mindspore.dataset.config.get_enable_shared_mem</code>	获取当前是否开启共享内存。
<code>mindspore.dataset.config.set_enable_autotune</code>	设置是否开启数据处理参数自动调优。
<code>mindspore.dataset.config.get_enable_autotune</code>	获取当前是否开启自动数据加速。
<code>mindspore.dataset.config.set_autotune_interval</code>	设置自动数据加速的配置调整 step 间隔。
<code>mindspore.dataset.config.get_autotune_interval</code>	获取当前自动数据加速的配置调整 step 间隔。
<code>mindspore.dataset.config.set_auto_offload</code>	设置是否开启数据异构加速。
<code>mindspore.dataset.config.get_auto_offload</code>	获取当前是否开启数据异构加速。
<code>mindspore.dataset.config.set_enable_watchdog</code>	设置 watchdog Python 线程是否启用。
<code>mindspore.dataset.config.get_enable_watchdog</code>	获取当前是否开启 watchdog Python 线程。

下页继续

表 472 – 续上页

<code>mindspore.dataset.config.set_fast_recovery</code>	在数据集管道故障恢复时，是否开启快速恢复模式（快速恢复模式下，无法保证随机性的数据增强操作得到与故障之前相同的结果）。
<code>mindspore.dataset.config.get_fast_recovery</code>	获取当前数据管道是否开启快速恢复模式。
<code>mindspore.dataset.config.set_multiprocessing_timeout_interval</code>	设置在多进程/多线程下，主进程/主线程获取数据超时时，告警日志打印的默认时间间隔（秒）。
<code>mindspore.dataset.config.get_multiprocessing_timeout_interval</code>	获取在多进程/多线程下，主进程/主线程获取数据超时时，告警日志打印的时间间隔的全局配置。
<code>mindspore.dataset.config.set_error_samples_mode</code>	设置在数据管道中处理错误样本的策略。
<code>mindspore.dataset.config.get_error_samples_mode</code>	获取当前数据管道中处理错误样本的策略配置。
<code>mindspore.dataset.config.ErrorSamplesMode</code>	指定数据管道中处理错误样本的策略。
<code>mindspore.dataset.config.set_debug_mode</code>	设置是否启动数据集管道的调试模式。
<code>mindspore.dataset.config.get_debug_mode</code>	获取当前是否开启数据处理流水线的调试模式。
<code>mindspore.dataset.config.set_multiprocessing_start_method</code>	设置启动数据预处理子进程方式的全局配置。
<code>mindspore.dataset.config.get_multiprocessing_start_method</code>	获取数据预处理子进程启动方式。

12.8.1 mindspore.dataset.config.set_sending_batches

`mindspore.dataset.config.set_sending_batches (batch_num)`

设置 Host 向 Device 发送数据的批数上限。

可用于实现自定义的数据发送控制逻辑，以解决 Device 内存不足的问题。在每个 epoch 中，当实际向 Device 发送的批数达到该值时，Host 将停止继续发送，直到用户再次通过该接口提高这个上限。

当前仅支持在 Ascend 后端的下沉模式训练时使用，下沉模式可通过 `mindspore.train.Model.train` 接口开启。

参数：

- **batch_num** (int) - Host 向 Device 发送数据的批数上限。0 表示没有发送上限。

异常：

- **TypeError** - `batch_num` 不是 int 类型。

样例：

```
>>> # Set a new global configuration value for the sending batches
>>> import mindspore.dataset as ds
>>> ds.config.set_sending_batches(10)
```


12.8.2 mindspore.dataset.config.load

`mindspore.dataset.config.load(file)`

根据文件内容加载项目配置文件。

参数：

- **file** (str) - 表示待加载的配置文件的路径。

异常：

- **RuntimeError** - 文件无效，解析失败。

样例：

```
>>> # Set new default configuration according to values in the configuration file.
>>> # example config file:
>>> # {
>>> #     "logFilePath": "/tmp",
>>> #     "numParallelWorkers": 4,
>>> #     "seed": 5489,
>>> #     "monitorSamplingInterval": 30
>>> # }
>>> import mindspore.dataset as ds
>>> config_file = "/path/to/config/file"
>>> ds.config.load(config_file)
```

12.8.3 mindspore.dataset.config.set_seed

`mindspore.dataset.config.set_seed(seed)`

设置数据处理中随机数生成器的种子。

设置种子能够控制随机生成器的初始状态，起到固定随机数生成结果的作用。

说明：该接口将同时设置 *random*、*numpy.random* 和 *mindspore.dataset* 模块的随机种子为指定值。

参数：

- **seed** (int) - 期望设置的种子值。需为非负数。

异常：

- **TypeError** - 当 *seed* 不为 int 类型。
- **ValueError** - 当 *seed* 为负数。

样例：

```
>>> # Set a new global configuration value for the seed value.
>>> # Operations with randomness will use the seed value to generate random values.
>>> import mindspore.dataset as ds
>>> ds.config.set_seed(1000)
```

12.8.4 mindspore.dataset.config.get_seed

`mindspore.dataset.config.get_seed()`

获取随机数的种子。如果随机数的种子已设置，则返回设置的值，否则将返回 `std::mt19937::default_seed` 这个默认种子值。

返回：

`int`，表示种子的随机数量。

样例：

```
>>> # Get the global configuration of seed.
>>> # If set_seed() is never called before, the default value(std::mt19937::default_seed)
    ↳ will be returned.
>>> import mindspore.dataset as ds
>>> seed = ds.config.get_seed()
```

12.8.5 mindspore.dataset.config.set_prefetch_size

`mindspore.dataset.config.set_prefetch_size(size)`

设置流水线中各个数据处理操作的缓冲队列大小。

缓冲队列的存在使得当前操作在下一操作取走数据前就能开始处理后续数据，各操作异步并发地执行。

更大的缓冲队列大小能够减少相邻操作吞吐速率不平衡时的整体处理时延，但也会消耗更大的系统内存。

参数：

- **size (int)** - 缓冲队列的大小，取值必须大于 0。

异常：

- **TypeError** - 当 `size` 的类型不为 `int`。
- **ValueError** - 当 `size` 不为正数。

说明：缓冲队列消耗的总内存与工作线程数成正比，为了避免内存占用过大，当工作线程数大于 4 时，实际使用的缓冲队列大小将调整为 $(size * 4 / \text{工作线程数})$ 与 1 中的较大值。

样例：

```
>>> # Set a new global configuration value for the prefetch size.
>>> import mindspore.dataset as ds
>>> ds.config.set_prefetch_size(1000)
```

12.8.6 mindspore.dataset.config.get_prefetch_size

`mindspore.dataset.config.get_prefetch_size()`

获取数据处理管道的输出缓存队列长度。如果 `set_prefetch_size` 方法未被调用，那么将会返回默认值 16。

返回：

`int`，表示预取的总行数。

样例：

```
>>> # Get the global configuration of prefetch size.
>>> # If set_prefetch_size() is never called before, the default value(16) will be returned.
>>> import mindspore.dataset as ds
>>> prefetch_size = ds.config.get_prefetch_size()
```

12.8.7 mindspore.dataset.config.set_num_parallel_workers

`mindspore.dataset.config.set_num_parallel_workers(num)`

为并行工作线程数量设置新的全局配置默认值。此设置会影响所有数据集操作的并行性。

参数：

- **num** (int) - 表示并行工作线程的数量，用作为每个数据集操作的默认值。

异常：

- **TypeError** - *num* 不是 int 类型。
- **ValueError** - *num* 小于等于 0 或 *num* 大于 `INT32_MAX(2147483647)` 时，并行工作线程数量设置无效。

样例：

```
>>> # Set a new global configuration value for the number of parallel workers.
>>> # Now parallel dataset operations will run with 8 workers.
>>> import mindspore.dataset as ds
>>> ds.config.set_num_parallel_workers(8)
```

12.8.8 mindspore.dataset.config.get_num_parallel_workers

`mindspore.dataset.config.get_num_parallel_workers()`

获取并行工作线程数量的全局配置，将作用于每个操作的并行工作线程数量。

返回：

int，表示每个操作中默认的并行工作线程的数量。

样例：

```
>>> # Get the global configuration of parallel workers.
>>> # If set_num_parallel_workers() is never called before, the default value(8) will be
    ↪ returned.
>>> import mindspore.dataset as ds
>>> num_parallel_workers = ds.config.get_num_parallel_workers()
```

12.8.9 mindspore.dataset.config.set_numa_enable

`mindspore.dataset.config.set_numa_enable(numa_enable)`

设置 NUMA 的默认状态为启动状态。如果 *numa_enable* 为 True，则需要确保安装了 NUMA 库。

参数：

- **numa_enable** (bool) - 表示是否使用 NUMA 绑定功能。

异常：

- **TypeError** - *numa_enable* 不是 bool 类型。

样例:

```
>>> # Set a new global configuration value for the state of numa enabled.
>>> # Now parallel dataset operations will run with numa bind function
>>> import mindspore.dataset as ds
>>> ds.config.set_numa_enable(True)
```

12.8.10 mindspore.dataset.config.get_numa_enable

`mindspore.dataset.config.get_numa_enable()`

获取 NUMA 的启动/禁用状态。该状态将用于所有进程。

返回:

bool, 表示 NUMA 的启动状态。

样例:

```
>>> # Get the global configuration of numa.
>>> import mindspore.dataset as ds
>>> numa_state = ds.config.get_numa_enable()
```

12.8.11 mindspore.dataset.config.set_monitor_sampling_interval

`mindspore.dataset.config.set_monitor_sampling_interval(interval)`

设置监测采样的默认间隔时间 (毫秒)。

参数:

- **interval** (int) - 表示用于性能监测采样的间隔时间 (毫秒)。

异常:

- **TypeError** - *interval* 不是 int 类型。
- **ValueError** - *interval* 小于等于 0 或 *interval* 大于 *INT32_MAX*(2147483647) 时, *interval* 无效。

样例:

```
>>> # Set a new global configuration value for the monitor sampling interval.
>>> import mindspore.dataset as ds
>>> ds.config.set_monitor_sampling_interval(100)
```

12.8.12 mindspore.dataset.config.get_monitor_sampling_interval

`mindspore.dataset.config.get_monitor_sampling_interval()`

获取性能监控采样时间间隔的全局配置。如果 *set_monitor_sampling_interval* 方法未被调用, 那么将会返回默认值 1000。

返回:

int, 表示性能监控采样间隔时间 (毫秒)。

样例:

```
>>> # Get the global configuration of monitor sampling interval.
>>> # If set_monitor_sampling_interval() is never called before, the default value(1000)
    ↳ will be returned.
>>> import mindspore.dataset as ds
>>> sampling_interval = ds.config.get_monitor_sampling_interval()
```

12.8.13 mindspore.dataset.config.set_callback_timeout

`mindspore.dataset.config.set_callback_timeout(timeout)`

为`mindspore.dataset.WaitedDSCallback`设置的默认超时时间（秒）。

参数：

- **timeout** (int) - 表示在出现死锁情况下，用于结束`mindspore.dataset.WaitedDSCallback`中等待的超时时间（秒）。`timeout`取值必须大于0。

异常：

- **TypeError** - `timeout`不是int类型。
- **ValueError** - `timeout`不为正数。

样例：

```
>>> # Set a new global configuration value for the timeout value.
>>> import mindspore.dataset as ds
>>> ds.config.set_callback_timeout(100)
```

12.8.14 mindspore.dataset.config.get_callback_timeout

`mindspore.dataset.config.get_callback_timeout()`

获取`mindspore.dataset.WaitedDSCallback`的默认超时时间。如果出现死锁，等待的函数将在超时时间结束后退出。

返回：

int，表示在出现死锁情况下，用于结束`mindspore.dataset.WaitedDSCallback`中的等待函数的超时时间（秒）。

样例：

```
>>> # Get the global configuration of callback timeout.
>>> # If set_callback_timeout() is never called before, the default value(60) will be
    ↳ returned.
>>> import mindspore.dataset as ds
>>> callback_timeout = ds.config.get_callback_timeout()
```

12.8.15 mindspore.dataset.config.set_auto_num_workers

`mindspore.dataset.config.set_auto_num_workers(enable)`

自动为每个数据集操作设置并行线程数量（默认情况下，此功能关闭）。

如果启用该功能，将自动调整每个数据集操作中的并行线程数量，这可能会覆盖用户通过脚本定义的并行线程数量或通过`mindspore.dataset.config.set_num_parallel_workers()`设置的默认值（如果用户未传递任何内容）。

目前，此函数仅针对具有 `per_batch_map`（batch 中的运行映射）的 YOLOv3 数据集进行了优化。此功能旨在为每个操作的优化线程数量分配一个基础值。并行线程数有所调整的数据集操作将会被记录。

参数：

- **enable** (bool) - 表示是否启用自动设置线程数量的特性。

异常：

- **TypeError** - `enable` 不是 bool 类型。

样例：

```
>>> # Enable auto_num_worker feature, this might override the num_parallel_workers passed in_
    ↳by user
>>> import mindspore.dataset as ds
>>> ds.config.set_auto_num_workers(True)
```

12.8.16 mindspore.dataset.config.get_auto_num_workers

`mindspore.dataset.config.get_auto_num_workers()`

获取当前是否开启自动线程调整。

返回：

bool，表示是否开启自动线程调整。

样例：

```
>>> # Get the global configuration of auto number worker feature.
>>> import mindspore.dataset as ds
>>> flag = ds.config.get_auto_num_workers()
```

12.8.17 mindspore.dataset.config.set_enable_shared_mem

`mindspore.dataset.config.set_enable_shared_mem(enable)`

设置是否在开启数据处理多进程时使用共享内存进行进程间通信。

使用共享内存可以加速进程间的数据传递效率。

该功能默认开启。

说明：暂不支持 Windows 和 MacOS 系统。

参数：

- **enable** (bool) - 是否使用共享内存进行进程间通信。

异常:

- **TypeError** - 当 *enable* 不为 bool 类型。

样例:

```
>>> # Enable shared memory feature to improve the performance of Python multiprocessing.
>>> import mindspore.dataset as ds
>>> ds.config.set_enable_shared_mem(True)
```

12.8.18 mindspore.dataset.config.get_enable_shared_mem

`mindspore.dataset.config.get_enable_shared_mem()`

获取当前是否开启共享内存。

说明: Windows 和 MacOS 平台尚不支持 *get_enable_shared_mem* 。

返回:

bool, 表示是否开启共享内存。

样例:

```
>>> # Get the flag of shared memory feature.
>>> import mindspore.dataset as ds
>>> shared_mem_flag = ds.config.get_enable_shared_mem()
```

12.8.19 mindspore.dataset.config.set_enable_autotune

`mindspore.dataset.config.set_enable_autotune(enable, filepath_prefix=None)`

设置是否开启数据处理参数自动调优。

在训练中根据环境资源的负载, 自动调整数据处理流水线中各个操作的参数配置, 如并行度、缓冲队列大小, 从而提高整体处理速度。

该功能默认不开启。

参数:

- **enable** (bool) - 是否开启自动调优。
- **filepath_prefix** (str, 可选) - 优化后的参数配置的保存路径。仅当 *enable* 为 *True* 时生效。各个 Device 上的参数配置文件将分别保存, 最终保存的文件名将为 *filepath_prefix* + *RANK_ID* + ".json", 其中 *RANK_ID* 为该文件对应的 Device 编号。默认值: *None*, 表示不保存配置文件。

异常:

- **TypeError** - 当 *enable* 的类型不为 bool。
- **TypeError** - 当 *filepath_prefix* 的类型不为 str。
- **RuntimeError** - 当 *filepath_prefix* 字符串长度为 0。
- **RuntimeError** - 当 *filepath_prefix* 为目录。
- **RuntimeError** - 当 *filepath_prefix* 路径不存在。

- **RuntimeError** - 当 `filepath_prefix` 没有写入权限。

说明:

- 保存的参数配置文件可通过 `mindspore.dataset.deserialize()` 接口加载, 返回最优参数的数据处理流水线对象。
- 可通过开启 INFO 级别日志, 查看参数调优过程。

生成的配置文件内容示例如下, "remark" 字段描述是否进行了数据处理参数调优, "summary" 字段简要展示了数据处理流水线中各个操作及其对应的最优配置, 而 "tree" 字段则为完整的数据处理流水线结构信息。

```
{
  "remark": "The following file has been auto-generated by the Dataset AutoTune.",
  "summary": [
    "CifarOp(ID:5)          (num_parallel_workers: 2, prefetch_size:64)",
    "MapOp(ID:4)           (num_parallel_workers: 2, prefetch_size:64)",
    "MapOp(ID:3)           (num_parallel_workers: 2, prefetch_size:64)",
    "BatchOp(ID:2)         (num_parallel_workers: 8, prefetch_size:64)"
  ],
  "tree": {
    ...
  }
}
```

样例:

```
>>> import mindspore.dataset as ds
>>>
>>> # enable AutoTune and save optimized data pipeline configuration
>>> ds.config.set_enable_autotune(True, "/path/to/autotune_out.json")
>>>
>>> # enable AutoTune
>>> ds.config.set_enable_autotune(True)
```

12.8.20 mindspore.dataset.config.get_enable_autotune

`mindspore.dataset.config.get_enable_autotune()`

获取当前是否开启自动数据加速。

返回:

bool, 表示是否开启自动数据加速。

样例:

```
>>> # get the state of AutoTune
>>> import mindspore.dataset as ds
>>> autotune_flag = ds.config.get_enable_autotune()
```


12.8.21 mindspore.dataset.config.set_autotune_interval

`mindspore.dataset.config.set_autotune_interval(interval)`

设置自动数据加速的配置调整 step 间隔。

默认设置为 0，将在每个 epoch 结束后调整配置；否则，将每隔 *interval* 个 step 调整一次配置。

参数：

- **interval** (int) - 配置调整的 step 间隔。

异常：

- **TypeError** - 当 *interval* 类型不为 int。
- **ValueError** - 当 *interval* 的值小于零。

样例：

```
>>> # set a new interval for AutoTune
>>> import mindspore.dataset as ds
>>> ds.config.set_autotune_interval(30)
```

12.8.22 mindspore.dataset.config.get_autotune_interval

`mindspore.dataset.config.get_autotune_interval()`

获取当前自动数据加速的配置调整 step 间隔。

返回：

int，自动数据加速的配置调整 step 间隔。

样例：

```
>>> # get the global configuration of the autotuning interval
>>> import mindspore.dataset as ds
>>> autotune_interval = ds.config.get_autotune_interval()
```

12.8.23 mindspore.dataset.config.set_auto_offload

`mindspore.dataset.config.set_auto_offload(offload)`

设置是否开启数据异构加速。

数据异构加速可以自动将数据处理的部分运算分配到不同的异构硬件（GPU 或 Ascend）上，以提高数据处理的速度。

参数：

- **offload** (bool) - 是否开启数据异构加速。

异常：

- **TypeError** - 当 *offload* 的类型不为 bool。

样例：

```
>>> # Enable automatic offload feature
>>> import mindspore.dataset as ds
>>> ds.config.set_auto_offload(True)
```

12.8.24 mindspore.dataset.config.get_auto_offload

`mindspore.dataset.config.get_auto_offload()`

获取当前是否开启数据异构加速。

返回:

bool, 表示是否开启数据异构加速。

样例:

```
>>> # Get the global configuration of the automatic offload feature.
>>> import mindspore.dataset as ds
>>> auto_offload = ds.config.get_auto_offload()
```

12.8.25 mindspore.dataset.config.set_enable_watchdog

`mindspore.dataset.config.set_enable_watchdog(enable)`

设置 watchdog Python 线程是否启用。watchdog Python 线程负责清理卡死或假死的子进程，默认启用。

参数:

- **enable** (bool) - 是否开启 watchdog Python 线程。

异常:

- **TypeError** - *enable* 不是 bool 类型。

样例:

```
>>> # Set a new global configuration value for the state of watchdog Python thread as_
↳enabled.
>>> import mindspore.dataset as ds
>>> ds.config.set_enable_watchdog(True)
```

12.8.26 mindspore.dataset.config.get_enable_watchdog

`mindspore.dataset.config.get_enable_watchdog()`

获取当前是否开启 watchdog Python 线程。默认初始状态是开启。

返回:

bool, 表示是否开启 watchdog Python 线程。

样例:

```
>>> # Get the global configuration of watchdog Python thread.
>>> import mindspore.dataset as ds
>>> watchdog_state = ds.config.get_enable_watchdog()
```

12.8.27 mindspore.dataset.config.set_fast_recovery

`mindspore.dataset.config.set_fast_recovery(fast_recovery)`

在数据集管道故障恢复时，是否开启快速恢复模式（快速恢复模式下，无法保证随机性的数据增强操作得到与故障之前相同的结果）。

参数：

- **fast_recovery** (bool) - 是否开启快速恢复模式。

异常：

- **TypeError** - *fast_recovery* 不是 bool 类型。

样例：

```
>>> import mindspore.dataset as ds
>>> ds.config.set_fast_recovery(False)
```

12.8.28 mindspore.dataset.config.get_fast_recovery

`mindspore.dataset.config.get_fast_recovery()`

获取当前数据管道是否开启快速恢复模式。

返回：

bool，当前数据管道是否开启快速恢复模式。

样例：

```
>>> import mindspore.dataset as ds
>>> is_fast_recovery = ds.config.get_fast_recovery()
```

12.8.29 mindspore.dataset.config.set_multiprocessing_timeout_interval

`mindspore.dataset.config.set_multiprocessing_timeout_interval(interval)`

设置在多进程/多线程下，主进程/主线程获取数据超时，告警日志打印的默认时间间隔（秒）。

参数：

- **interval** (int) - 表示多进程/多线程下，主进程/主线程获取数据超时，告警日志打印的时间间隔（秒）。

异常：

- **TypeError** - *interval* 不是 int 类型。
- **ValueError** - *interval* 小于等于 0 或 *interval* 大于 *INT32_MAX*(2147483647) 时，*interval* 无效。

样例：

```
>>> # Set a new global configuration value for multiprocessing/multithreading timeout when
    ↪ getting data.
>>> import mindspore.dataset as ds
>>> ds.config.set_multiprocessing_timeout_interval(300)
```

12.8.30 mindspore.dataset.config.get_multiprocessing_timeout_interval

`mindspore.dataset.config.get_multiprocessing_timeout_interval()`

获取在多线程/多进程下，主进程/主线程获取数据超时时，告警日志打印的时间间隔的全局配置。

返回：

int，表示多线程/多进程下，主进程/主线程获取数据超时时，告警日志打印的时间间隔（默认 300 秒）。

样例：

```
>>> # Get the global configuration of multiprocessing/multithreading timeout when main_
    ↳process/thread gets data
>>> # from subprocesses/child threads. If set_multiprocessing_timeout_interval() is never_
    ↳called before, the
>>> # default value(300) will be returned.
>>> import mindspore.dataset as ds
>>> multiprocessing_timeout_interval = ds.config.get_multiprocessing_timeout_interval()
```

12.8.31 mindspore.dataset.config.set_error_samples_mode

`mindspore.dataset.config.set_error_samples_mode(error_samples_mode)`

设置在数据管道中处理错误样本的策略。

说明：

- 此错误样本特性仅适用于数据集管道中的 Map 操作。
- 对于 `ErrorSamplesMode.REPLACE` 模式，将使用内部缓存中的其他样本。
- 如果在多机设置中使用 `ErrorSamplesMode.SKIP` 模式，请手动确保每个分片的有效样本数是相同的（否则可能会导致挂起）。一种解决方法是通过 `Concat` 操作拼接一个样本全有效的数据集，然后采用 `Take` 操作填补跳过的错误样本数。

参数：

- **error_samples_mode** (`ErrorSamplesMode`) - 处理错误样本的方法。
 - `ErrorSamplesMode.RETURN`：表示处理过程中遇到错误样本时将报错并抛出异常。
 - `ErrorSamplesMode.REPLACE`：表示处理过程中遇到错误样本时将使用正确的样本替换处理。
 - `ErrorSamplesMode.SKIP`：表示处理过程中遇到错误样本时将直接跳过此样本。

异常：

- **TypeError** - `error_samples_mode` 不是 `mindspore.dataset.config.ErrorSamplesMode` 类型。

样例：

```
>>> import mindspore.dataset as ds
>>> ds.config.set_error_samples_mode(ds.config.ErrorSamplesMode.SKIP)
```

12.8.32 mindspore.dataset.config.get_error_samples_mode

`mindspore.dataset.config.get_error_samples_mode()`

获取当前数据管道中处理错误样本的策略配置。

返回:

`mindspore.dataset.config.ErrorSamplesMode` , 当前数据管道处理错误样本的方法。

- `ErrorSamplesMode.RETURN`: 表示错误样本会导致产生错误并返回。
- `ErrorSamplesMode.REPLACE`: 表示错误样本会被内部确定的样本替换。
- `ErrorSamplesMode.SKIP`: 表示错误样本会被跳过。

样例:

```
>>> import mindspore.dataset as ds
>>> error_samples_mode = ds.config.get_error_samples_mode()
```

12.8.33 mindspore.dataset.config.ErrorSamplesMode

`class mindspore.dataset.config.ErrorSamplesMode`

指定数据管道中处理错误样本的策略。

- `ErrorSamplesMode.RETURN` - 表示处理过程中遇到错误样本时将报错并抛出异常。
- `ErrorSamplesMode.REPLACE` - 表示处理过程中遇到错误样本时将使用正确的样本替换处理。
- `ErrorSamplesMode.SKIP` - 表示处理过程中遇到错误样本时将直接跳过此样本。

12.8.34 mindspore.dataset.config.set_debug_mode

`mindspore.dataset.config.set_debug_mode(debug_mode_flag: bool, debug_hook_list: list = None)`

设置是否启动数据集管道的调试模式。当启用该模式时, 数据集管道会使用单线程工作, 所有数据集操作均以同步方式执行。

说明: 当启用调试模式时,

- 如果随机种子没有被设置, 则会将随机种子设置为 1, 以便在调试模式下执行数据集管道, 从而获得确定性的结果。
- 以下设置将不会生效:
 - `auto_offload` (强制设置为 `False`)
 - `enable_autotune` (强制设置为 `False`)
 - `error_samples_mode` (强制设置为 `ErrorSamplesMode.RETURN`)
 - `num_parallel_workers` (强制设置为 1)
- 数据集管道中 `map` 操作的 `offload` 参数将被忽略。
- 数据集管道中 `GeneratorDataset`、`map` 操作和 `batch` 操作的 `python_multiprocessing` 参数将被忽略。
- 数据集加载 API 中的 `cache` 参数将会被忽略。

参数:

- **debug_mode_flag** (bool) - 是否开启数据集管道调试模式。该模式会强制数据集管道以单线程同步的方式运行。
- **debug_hook_list** (list[*DebugHook*]) - 调试钩子列表，用于启用调试模式时插入到 *map* 操作中各个变换操作的前后。默认值：None，仅插入基础的信息打印钩子，用于打印各个变换操作输入/输出数据的形状/大小/类型信息。

异常：

- **TypeError** - *debug_mode_flag* 不是 bool 类型。
- **TypeError** - *debug_hook_list* 不是 list 类型。
- **TypeError** - *debug_hook_list* 中的元素不是 *DebugHook* 类型。

样例：

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> import mindspore.dataset.debug as debug
>>>
>>> # 1. Enable dataset pipeline debug mode and use default debug hook.
>>> # Print shape and type of input/output data of each transform op in map operator.
>>> ds.config.set_debug_mode(True)
>>>
>>> # 2. Enable dataset pipeline debug mode and use pre-defined debug hook provided by
↳MindData.
>>> ds.config.set_debug_mode(True, debug_hook_list=[debug.PrintDataHook()])
>>>
>>> # 3. Enable dataset pipeline debug mode and use user-defined debug hook. It must define a
>>> # class inherited from DebugHook.
>>> class CustomizedHook(debug.DebugHook):
...     def __init__(self):
...         super().__init__()
...
...     def compute(self, *args):
...         # Add your debugging code here.
...         return args
>>>
>>> ds.config.set_debug_mode(True, debug_hook_list=[CustomizedHook()])
>>>
>>> # 4. Enable dataset pipeline debug mode and use user-defined debug hook and insert by
↳users manually.
>>> ds.config.set_debug_mode(True)
>>> dataset = ds.ImageFolderDataset(dataset_dir="/path/to/image_folder_dataset_directory")
>>>
>>> # The debug hook is added after Decode operation.
>>> dataset = dataset.map([vision.Decode(), CustomizedHook(), vision.CenterCrop(100)])
```

12.8.35 mindspore.dataset.config.get_debug_mode

`mindspore.dataset.config.get_debug_mode()`

获取当前是否开启数据处理流水线的调试模式。

返回:

bool, 表示是否开启数据处理流水线的调试模式。

样例:

```
>>> import mindspore.dataset as ds
>>> debug_mode = ds.config.get_debug_mode()
```

12.8.36 mindspore.dataset.config.set_multiprocessing_start_method

`mindspore.dataset.config.set_multiprocessing_start_method(start_method='fork')`

设置启动数据预处理子进程方式的全局配置。

此设置将影响: `GeneratorDataset` 自定义数据集、`map` 操作和 `batch` 操作子进程的启动方式。

参数:

- **start_method** (str, 可选) - 数据预处理阶段子进程的启动方式。默认值: 'fork'。可选值: ['fork', 'spawn']。

异常:

- **TypeError** - `start_method` 不是 str 类型。
- **ValueError** - `start_method` 不是 'fork' 或者 'spawn'。

样例:

```
>>> # Set a new global configuration value for multiprocessing start method.
>>> import mindspore.dataset as ds
>>> ds.config.set_multiprocessing_start_method("fork")
```

12.8.37 mindspore.dataset.config.get_multiprocessing_start_method

`mindspore.dataset.config.get_multiprocessing_start_method()`

获取数据预处理子进程启动方式。

如果 `set_multiprocessing_start_method` 没有被调用过, 默认返回 `fork`。

返回:

str, 表示数据预处理阶段子进程启动方式。

样例:

```
>>> # Get the global configuration of multiprocessing start method when main process gets_
↳data
>>> # from subprocesses.
>>> # default ``fork`` will be returned.
>>> import mindspore.dataset as ds
>>> multiprocessing_start_method = ds.config.get_multiprocessing_start_method()
```

12.9 工具

<code>mindspore.dataset.BatchInfo</code>	当 <i>batch</i> 操作中, 参数 <i>batch_size</i> 或 <i>per_batch_map</i> 的传入对象是回调函数时, 可以通过此类提供的方法获取数据集信息。
<code>mindspore.dataset.DatasetCache</code>	创建数据缓存客户端实例。
<code>mindspore.dataset.DSCallback</code>	数据处理回调类的抽象基类, 用户可以基于此类实现自己的回调操作。
<code>mindspore.dataset.Schema</code>	用于解析和存储数据列属性的类。
<code>mindspore.dataset.Shuffle</code>	指定混洗模式的枚举类。
<code>mindspore.dataset.WaitedDSCallback</code>	阻塞式数据处理回调类的抽象基类, 用于与训练回调类 <code>mindspore.train.Callback</code> 的同步。
<code>mindspore.dataset.compare</code>	比较两个数据处理管道是否相同。
<code>mindspore.dataset.debug.DebugHook</code>	数据集管道 Python 调试器钩子的基类。
<code>mindspore.dataset.deserialize</code>	数据处理管道反序列化, 支持输入 Python 字典或使用 <code>mindspore.dataset.serialize()</code> 接口生成的 JSON 文件。
<code>mindspore.dataset.serialize</code>	将数据处理管道序列化成 JSON 文件。
<code>mindspore.dataset.show</code>	将数据处理管道图写入 <code>logger.info</code> 文件。
<code>mindspore.dataset.sync_wait_for_dataset</code>	等待所有的卡需要的数据集文件下载完成。
<code>mindspore.dataset.utils.imshow_det_bbox</code>	使用给定的边界框和类别置信度绘制图像。
<code>mindspore.dataset.utils.LineReader</code>	基于行的文件读取器。

12.9.1 mindspore.dataset.BatchInfo

class mindspore.dataset.BatchInfo

当 *batch* 操作中, 参数 *batch_size* 或 *per_batch_map* 的传入对象是回调函数时, 可以通过此类提供的方法获取数据集信息。

`get_batch_num()`
返回当前 epoch 已经处理的 batch 数, 数值从 0 开始。
样例:

```
>>> # Create a dataset where its batch size is dynamic
>>> # Define a callable batch size function and let batch size increase 1 each time.
>>> import mindspore.dataset as ds
>>> from mindspore.dataset import BatchInfo
>>>
>>> dataset = ds.GeneratorDataset([i for i in range(3)], "column1", shuffle=False)
>>> def add_one(BatchInfo):
...     return BatchInfo.get_batch_num() + 1
>>> dataset = dataset.batch(batch_size=add_one)
>>> print(list(dataset))
[[Tensor(shape=[1], dtype=Int64, value= [0])], [Tensor(shape=[2], dtype=Int64, value= [1,
↪ 2])]]
```

`get_epoch_num()`
返回当前的 epoch 数, 数值从 0 开始。
样例:


```

>>> # Create a dataset where its batch size is dynamic
>>> # Define a callable batch size function and let batch size increase 1 each epoch.
>>> import mindspore.dataset as ds
>>> from mindspore.dataset import BatchInfo
>>>
>>> dataset = ds.GeneratorDataset([i for i in range(4)], "column1", shuffle=False)
>>> def add_one_by_epoch(BatchInfo):
...     return BatchInfo.get_epoch_num() + 1
>>> dataset = dataset.batch(batch_size=add_one_by_epoch)
>>>
>>> result = []
>>> epoch = 2
>>> iterator = dataset.create_tuple_iterator(num_epochs=epoch)
>>> for i in range(epoch):
...     result.extend(list(iterator))
>>> # result:
>>> # [[Tensor(shape=[1], dtype=Int64, value= [0])], [Tensor(shape=[1], dtype=Int64,
↪value= [1])],
>>> # [Tensor(shape=[1], dtype=Int64, value= [2])], [Tensor(shape=[1], dtype=Int64,
↪value= [3])],
>>> # [Tensor(shape=[2], dtype=Int64, value= [0, 1])], [Tensor(shape=[2], dtype=Int64,
↪value= [2, 3])]]

```

12.9.2 mindspore.dataset.DatasetCache

class mindspore.dataset.**DatasetCache** (session_id, size=0, spilling=False, hostname=None, port=None, num_connections=None, prefetch_size=None)

创建数据缓存客户端实例。

关于单节点数据缓存的使用，请参阅 [单节点数据缓存教程](#)。

参数：

- **session_id** (int) - 当前数据缓存客户端的会话 ID，用户在命令行开启缓存服务端后，可通过 `dataset-cache -g` 获取。
- **size** (int, 可选) - 设置数据缓存服务可用的内存大小。默认值：0，表示内存使用没有限制。
- **spilling** (bool, 可选) - 如果共享内存不足，是否将溢出部分缓存到磁盘。默认值：False。
- **hostname** (str, 可选) - 数据缓存服务客户端的主机 IP。默认值：None，表示使用默认主机 IP 127.0.0.1。
- **port** (int, 可选) - 指定连接到数据缓存服务端的端口号。默认值：None，表示端口为 50052。
- **num_connections** (int, 可选) - TCP/IP 连接数量。默认值：None，表示连接数量为 12。
- **prefetch_size** (int, 可选) - 指定缓存队列大小，使用缓存功能时，将直接从缓存队列中获取数据。默认值：None，表示缓存队列大小为 20。

样例：

```

>>> import subprocess
>>> import mindspore.dataset as ds
>>>
>>> # Create a cache instance with command line `dataset-cache --start`
>>> # Create a session with `dataset-cache -g`
>>> # After creating cache with a valid session, get session id with command `dataset-cache -
↪-list_sessions`

```

(下页继续)

(续上页)

```
>>> command = "dataset-cache --list_sessions | tail -1 | awk -F ' ' '{{print $1;}}'"
>>> session_id = subprocess.getoutput(command).split('\n')[-1]
>>> some_cache = ds.DatasetCache(session_id=int(session_id), size=0)
>>>
>>> dataset_dir = "/path/to/image_folder_dataset_directory"
>>> dataset = ds.ImageFolderDataset(dataset_dir, cache=some_cache)
```

get_stat()

获取缓存实例的统计信息。在数据管道结束后，可获取三类统计信息，包括平均缓存命中数（avg_cache_sz），内存中的缓存数（num_mem_cached）和磁盘中的缓存数（num_disk_cached）。

样例：

```
>>> import os
>>> import subprocess
>>> import mindspore.dataset as ds
>>>
>>> # In example above, we created cache with a valid session id
>>> command = "dataset-cache --list_sessions | tail -1 | awk -F ' ' '{{print $1;}}'"
>>> id = subprocess.getoutput(command).split('\n')[-1]
>>> some_cache = ds.DatasetCache(session_id=int(id), size=0)
>>>
>>> # run the dataset pipeline to trigger cache
>>> dataset = ds.ImageFolderDataset("/path/to/image_folder_dataset_directory",
↪ cache=some_cache)
>>> data = list(dataset)
>>>
>>> # get status of cache
>>> stat = some_cache.get_stat()
>>> # Average cache size
>>> cache_sz = stat.avg_cache_sz
>>> # Number of rows cached in memory
>>> num_mem_cached = stat.num_mem_cached
>>> # Number of rows spilled to disk
>>> num_disk_cached = stat.num_disk_cached
```

12.9.3 mindspore.dataset.DSCallback

class mindspore.dataset.DSCallback (step_size=1)

数据处理回调类的抽象基类，用户可以基于此类实现自己的回调操作。

用户可通过 `ds_run_context` 获取数据处理管道相关信息，包括 `cur_epoch_num` (当前 epoch 数)、`cur_step_num_in_epoch` (当前 epoch 的 step 数)、`cur_step_num` (当前 step 数)。

参数：

- **step_size** (int, 可选) - 定义相邻的 `ds_step_begin/ds_step_end` 调用之间相隔的 step 数。默认值：1，表示每个 step 都会调用。

样例：

```
>>> import mindspore.dataset as ds
>>> from mindspore.dataset import DSCallback
>>>
>>> class PrintInfo(DSCallback):
...     def ds_begin(self, ds_run_context):
```

(下页继续)

(续上页)

```

...     print("callback: start dataset pipeline", flush=True)
...
...     def ds_epoch_begin(self, ds_run_context):
...         print("callback: epoch begin, we are in epoch", ds_run_context.cur_epoch_num,
↪flush=True)
...
...     def ds_epoch_end(self, ds_run_context):
...         print("callback: epoch end, we are in epoch", ds_run_context.cur_epoch_num,
↪flush=True)
...
...     def ds_step_begin(self, ds_run_context):
...         print("callback: step begin, step", ds_run_context.cur_step_num_in_epoch,
↪flush=True)
...
...     def ds_step_end(self, ds_run_context):
...         print("callback: step end, step", ds_run_context.cur_step_num_in_epoch,
↪flush=True)
>>>
>>> dataset = ds.GeneratorDataset([1, 2], "col1", shuffle=False, num_parallel_workers=1)
>>> dataset = dataset.map(operations=lambda x: x, callbacks=PrintInfo())
>>>
>>> # Start dataset pipeline
>>> iterator = dataset.create_tuple_iterator(num_epochs=2)
>>> for i in range(2):
...     for d in iterator:
...         pass
callback: start dataset pipeline
callback: epoch begin, we are in epoch 1
callback: step begin, step 1
callback: step begin, step 2
callback: step end, step 1
callback: step end, step 2
callback: epoch end, we are in epoch 1
callback: epoch begin, we are in epoch 2
callback: step begin, step 1
callback: step begin, step 2
callback: step end, step 1
callback: step end, step 2
callback: epoch end, we are in epoch 2

```

ds_begin(ds_run_context)

用于定义在数据处理管道启动前执行的回调方法。

参数:

- **ds_run_context** (RunContext) - 数据处理管道运行信息。

ds_epoch_begin(ds_run_context)

用于定义在每个数据 epoch 开始前执行的回调方法。

参数:

- **ds_run_context** (RunContext) - 数据处理管道运行信息。

ds_epoch_end(ds_run_context)

用于定义在每个数据 epoch 结束后执行的回调方法。

参数:

- **ds_run_context** (RunContext) - 数据处理管道运行信息。

ds_step_begin (*ds_run_context*)

用于定义在指定数据 step 开始前执行的回调方法。

参数:

- **ds_run_context** (RunContext) - 数据处理管道运行信息。

ds_step_end (*ds_run_context*)

用于定义在指定数据 step 结束后执行的回调方法。

参数:

- **ds_run_context** (RunContext) - 数据处理管道运行信息。

12.9.4 mindspore.dataset.Schema

class mindspore.dataset.**Schema** (*schema_file=None*)

用于解析和存储数据列属性的类。

参数:

- **schema_file** (str, 可选) - schema 文件的路径。默认值: None。

异常:

- **RuntimeError** - schema 文件加载失败。

样例:

```
>>> import mindspore.dataset as ds
>>> from mindspore import dtype as mstype
>>>
>>> # Create schema; specify column name, mindspore.dtype and shape of the column
>>> schema = ds.Schema()
>>> schema.add_column(name='col1', de_type=mstype.int64, shape=[2])
```

add_column (*name, de_type, shape=None*)

向 schema 中添加新列。

参数:

- **name** (str) - 新列的名称。
- **de_type** (str) - 列的数据类型。
- **shape** (list[int], 可选) - 列 shape。默认值: None, -1 表示该维度的 shape 是未知的。

异常:

- **ValueError** - 列类型未知。

样例:

```
>>> import mindspore.dataset as ds
>>> from mindspore import dtype as mstype
>>>
>>> schema = ds.Schema()
>>> schema.add_column('col_1d', de_type=mstype.int64, shape=[2])
```

from_json (*json_obj*)

从 JSON 对象获取 schema 文件。

参数:

- **json_obj** (dictionary) - 解析的 JSON 对象。

异常:

- **RuntimeError** - 对象中存在未知的项。
- **RuntimeError** - 对象中缺少数据集类型。
- **RuntimeError** - 对象中缺少列。

样例:

```
>>> import json
>>> from mindspore.dataset import Schema
>>>
>>> with open("/path/to/schema_file", "r") as file:
...     json_obj = json.load(file)
...     schema = Schema()
...     schema.from_json(json_obj)
```

parse_columns (*columns*)

解析传入的数据列的属性，并将其添加到自身的 schema 中。

参数:

- **columns** (Union[dict, list[dict], tuple[dict]]) - 数据集属性信息，从 schema 文件解码。
 - **list** [dict]: *name* 和 *type* 必须为 key 值，*shape* 可选。
 - **dict**: *columns.keys()* 作为名称，*columns.values()* 是 dict，其中包含可选的 *type*，*shape*。

异常:

- **RuntimeError** - 解析列失败。
- **RuntimeError** - 列 *name* 字段缺失。
- **RuntimeError** - 列 *type* 字段缺失。

样例:

```
>>> from mindspore.dataset import Schema
>>> schema = Schema()
>>> columns1 = [{'name': 'image', 'type': 'int8', 'shape': [3, 3]},
...             {'name': 'label', 'type': 'int8', 'shape': [1]}]
>>> schema.parse_columns(columns1)
>>> columns2 = {'image': {'shape': [3, 3], 'type': 'int8'}, 'label': {'shape': [1], 'type': 'int8'}}
>>> schema.parse_columns(columns2)
```

to_json ()

获取 schema 的 JSON 字符串。

返回:

str, schema 模式的 JSON 字符串。

样例:

```
>>> from mindspore.dataset import Schema
>>> from mindspore import dtype as mstype
>>>
>>> schema = Schema()
```

(下页继续)

(续上页)

```
>>> schema.add_column('col_1d', de_type=mstype.int64, shape=[2])
>>> json = schema.to_json()
```

12.9.5 mindspore.dataset.Shuffle

class mindspore.dataset.Shuffle

指定混洗模式的枚举类。

- **Shuffle.FALSE** - 关闭数据集混洗。
- **Shuffle.ADAPTIVE** - 当数据集样本小于等于 1 亿时, 采用 **Shuffle.GLOBAL**, 当大于 1 亿时, 采用局部 **Shuffle.PARTIAL**, 每 100 万样本混洗一次。
- **Shuffle.GLOBAL** - 执行全局混洗, 一次性混洗数据集中所有样本。
- **Shuffle.PARTIAL** - 执行局部混洗, 每 100 万个样本混洗一次。
- **Shuffle.FILES** - 仅混洗文件, 不混洗文件中的数据。
- **Shuffle.INFILE** - 保持读入文件的序列, 仅混洗每个文件中的数据。

12.9.6 mindspore.dataset.WaitedDSCallback

class mindspore.dataset.WaitedDSCallback (*step_size=1*)

阻塞式数据处理回调类的抽象基类, 用于与训练回调类 [mindspore.train.Callback](#) 的同步。

可用于在 **step** 或 **epoch** 开始前执行自定义的回调方法, 例如在自动数据增强中根据上一个 **epoch** 的 **loss** 值来更新增强操作参数配置。

用户可通过 *train_run_context* 获取网络训练相关信息, 如 *network*、*train_network*、*epoch_num*、*batch_num*、*loss_fn*、*optimizer*、*parallel_mode*、*device_number*、*list_callback*、*cur_epoch_num*、*cur_step_num*、*dataset_sink_mode*、*net_outputs* 等, 详见 [mindspore.train.Callback](#)。

用户可通过 *ds_run_context* 获取数据处理管道相关信息, 包括 *cur_epoch_num* (当前 epoch 数)、*cur_step_num_in_epoch* (当前 epoch 的 step 数)、*cur_step_num* (当前 step 数)。

说明: 注意, 第 2 个 **step** 或 **epoch** 开始时才会触发该调用。

参数:

- **step_size** (int, 可选) - 每个 **step** 包含的数据行数。通常 **step_size** 与 **batch_size** 一致。默认值: 1。

样例:

```
>>> import mindspore as ms
>>> import mindspore.dataset as ds
>>> import mindspore.nn as nn
>>> from mindspore.dataset import WaitedDSCallback
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> ms.set_device(device_target="CPU")
>>>
>>> # custom callback class for data synchronization in data pipeline
>>> class MyWaitedCallback(WaitedDSCallback):
```

(下页继续)

(续上页)

```

...     def __init__(self, events, step_size=1):
...         super().__init__(step_size)
...         self.events = events
...
...     # callback method to be executed by data pipeline before the epoch starts
...     def sync_epoch_begin(self, train_run_context, ds_run_context):
...         event = f"ds_epoch_begin_{ds_run_context.cur_epoch_num}_{ds_run_context.cur_step_
↪num}"
...         self.events.append(event)
...
...     # callback method to be executed by data pipeline before the step starts
...     def sync_step_begin(self, train_run_context, ds_run_context):
...         event = f"ds_step_begin_{ds_run_context.cur_epoch_num}_{ds_run_context.cur_step_
↪num}"
...         self.events.append(event)
>>>
>>> # custom callback class for data synchronization in network training
>>> class MyMSCallback(ms.Callback):
...     def __init__(self, events):
...         self.events = events
...
...     # callback method to be executed by network training after the epoch ends
...     def epoch_end(self, run_context):
...         cb_params = run_context.original_args()
...         event = f"ms_epoch_end_{cb_params.cur_epoch_num}_{cb_params.cur_step_num}"
...         self.events.append(event)
...
...     # callback method to be executed by network training after the step ends
...     def step_end(self, run_context):
...         cb_params = run_context.original_args()
...         event = f"ms_step_end_{cb_params.cur_epoch_num}_{cb_params.cur_step_num}"
...         self.events.append(event)
>>>
>>> # custom network
>>> class Net(nn.Cell):
...     def construct(self, x, y):
...         return x
>>>
>>> # define a parameter that needs to be synchronized between data pipeline and network_
↪training
>>> events = []
>>>
>>> # define callback classes of data pipeline and network training
>>> my_cb1 = MyWaitedCallback(events, 1)
>>> my_cb2 = MyMSCallback(events)
>>> arr = [1, 2, 3, 4]
>>>
>>> # construct data pipeline
>>> data = ds.NumpySlicesDataset((arr, arr), column_names=["c1", "c2"], shuffle=False)
>>> # map the data callback object into the pipeline
>>> data = data.map(operations=(lambda x: x), callbacks=my_cb1)
>>>
>>> net = Net()
>>> model = ms.train.Model(net)
>>>
>>> # add the data and network callback objects to the model training callback list
>>> model.train(2, data, dataset_sink_mode=False, callbacks=[my_cb2, my_cb1])

```

sync_epoch_begin (*train_run_context, ds_run_context*)

用于定义在数据 epoch 开始前, 训练 epoch 结束后执行的回调方法。

参数:

- **train_run_context** - 包含前一个 epoch 反馈信息的网络训练运行信息。
- **ds_run_context** - 数据处理管道运行信息。

sync_step_begin (*train_run_context, ds_run_context*)

用于定义在数据 step 开始前, 训练 step 结束后执行的回调方法。

参数:

- **train_run_context** - 包含前一个 step 反馈信息的网络训练运行信息。
- **ds_run_context** - 数据处理管道运行信息。

12.9.7 mindspore.dataset.compare

mindspore.dataset.compare (*pipeline1, pipeline2*)

比较两个数据处理管道是否相同。

参数:

- **pipeline1** (Dataset) - 数据处理管道。
- **pipeline2** (Dataset) - 数据处理管道。

返回:

bool, 两个数据处理管道是否相同。

样例:

```
>>> import mindspore.dataset as ds
>>>
>>> pipeline1 = ds.MnistDataset("/path/to/mnist_dataset_directory", num_samples=100)
>>> pipeline2 = ds.Cifar10Dataset("/path/to/cifar10_dataset_directory", num_samples=100)
>>> res = ds.compare(pipeline1, pipeline2)
```

12.9.8 mindspore.dataset.debug.DebugHook

class mindspore.dataset.debug.DebugHook (*prev_op_name=None*)

数据集管道 Python 调试器钩子的基类。所有用户定义的钩子行为都必须继承这个基类。

为了调试数据集管道中 *map* 操作的输入和输出数据, 用户可以在该类的 *compute* 函数中添加断点, 可以打印日志查看数据的类型和形状等。

参数:

- **prev_op_name** (str, 可选) - 上一个调试点的变换名称, 默认值: None, 一般不需指定。

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.debug as debug
>>>
>>> class CustomizedHook(debug.DebugHook):
...     def __init__(self):
```

(下页继续)

(续上页)

```

...     super().__init__()
...
...     def compute(self, *args):
...         import pdb
...         pdb.set_trace()
...         print("Data after decode", *args)
...         return args
>>>
>>> # Enable debug mode
>>> ds.config.set_debug_mode(True, debug_hook_list=[CustomizedHook()])
>>>
>>> # Define dataset pipeline
>>> dataset = ds.ImageFolderDataset(dataset_dir="/path/to/image_folder_dataset_directory")
>>> # Insert debug hook after `Decode` operation.
>>> dataset = dataset.map([vision.Decode(), CustomizedHook(), vision.CenterCrop(100)])

```

abstract compute(*args)

定义该调试钩子的行为，此方法需要被子类重写。参考上述样例如何自定义一个调试钩子。

参数：

- **args** (Any) - 被调试的变换的输入/输出数据，可以直接打印查看。

12.9.9 mindspore.dataset.deserialize

mindspore.dataset.deserialize (input_dict=None, json_filepath=None)

数据处理管道反序列化，支持输入 Python 字典或使用 `mindspore.dataset.serialize()` 接口生成的 JSON 文件。

参数：

- **input_dict** (dict) - 以 Python 字典存储的数据处理管道。默认值：None。
- **json_filepath** (str) - 数据处理管道 JSON 文件的路径，该文件以通用 JSON 格式存储数据处理管道信息，用户可通过 `mindspore.dataset.serialize()` 接口生成。默认值：None。

返回：

当反序列化成功时，将返回 Dataset 对象；当无法被反序列化时，`deserialize` 将会失败，且返回 None。

异常：

- **OSError** - 无法打开 `json_filepath` 指定的文件。

样例：

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> mnist_dataset_dir = "/path/to/mnist_dataset_directory"
>>> dataset = ds.MnistDataset(mnist_dataset_dir, num_samples=100)
>>> one_hot_encode = transforms.OneHot(10) # num_classes is input argument
>>> dataset = dataset.map(operations=one_hot_encode, input_columns="label")
>>> dataset = dataset.batch(batch_size=10, drop_remainder=True)
>>>
>>> # Case 1: to/from JSON file
>>> serialized_data = ds.serialize(dataset, json_filepath="/path/to/mnist_dataset_pipeline.
↪ json")

```

(下页继续)

(续上页)

```
>>> deserialized_dataset = ds.deserialize(json_filepath="/path/to/mnist_dataset_pipeline.json
↪")
>>>
>>> # Case 2: to/from Python dictionary
>>> serialized_data = ds.serialize(dataset)
>>> deserialized_dataset = ds.deserialize(input_dict=serialized_data)
```

12.9.10 mindspore.dataset.serialize

`mindspore.dataset.serialize(dataset, json_filepath="")`

将数据处理管道序列化成 JSON 文件。

说明：目前不支持对 Python 对象进行完整的序列化。不支持的场景包括数据管道中使用了 *GeneratorDataset* 或包含自定义 Python 函数的 *map* 或 *batch* 操作。对于 Python 对象，序列化操作不会得到完整的对象内容，这意味着对序列化得到的 JSON 文件进行反序列化时，可能会导致错误。例如，对自定义 Python 函数（Python user-defined functions）的数据管道进行序列化时，会出现相关警告提示，并且得到的 JSON 文件不能被反序列化为可用的数据管道。

参数：

- **dataset** (Dataset) - 数据处理管道对象。
- **json_filepath** (str) - 生成序列化 JSON 文件的路径。默认值：''，不指定 JSON 路径。

返回：

Dict，包含序列化数据集图的字典。

异常：

- **OSError** - 无法打开文件。

样例：

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> mnist_dataset_dir = "/path/to/mnist_dataset_directory"
>>> dataset = ds.MnistDataset(mnist_dataset_dir, num_samples=100)
>>> one_hot_encode = transforms.OneHot(10) # num_classes is input argument
>>> dataset = dataset.map(operations=one_hot_encode, input_columns="label")
>>> dataset = dataset.batch(batch_size=10, drop_remainder=True)
>>> # serialize it to JSON file
>>> serialized_data = ds.serialize(dataset, json_filepath="/path/to/mnist_dataset_pipeline.
↪json")
```

12.9.11 mindspore.dataset.show

`mindspore.dataset.show(dataset, indentation=2)`

将数据处理管道图写入 `logger.info` 文件。

参数：

- **dataset** (Dataset) - 数据处理管道对象。
- **indentation** (int, 可选) - 设置 JSON 字符串打印时的缩进字符数，None 表示不缩进。默认值：2，缩进 2 个字符。

样例：

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> mnist_dataset_dir = "/path/to/mnist_dataset_directory"
>>> dataset = ds.MnistDataset(mnist_dataset_dir, num_samples=100)
>>> one_hot_encode = transforms.OneHot(10)
>>> dataset = dataset.map(operations=one_hot_encode, input_columns="label")
>>> dataset = dataset.batch(batch_size=10, drop_remainder=True)
>>> ds.show(dataset)
```

12.9.12 mindspore.dataset.sync_wait_for_dataset

`mindspore.dataset.sync_wait_for_dataset(rank_id, rank_size, current_epoch)`

等待所有的卡需要的数据集文件下载完成。

说明：需要配合 `mindspore.dataset.OBSMindDataset` 使用，建议在每次 epoch 开始前调用。

参数：

- **rank_id** (int) - 当前卡的逻辑序号。
- **rank_size** (int) - 卡的数量。
- **current_epoch** (int) - 训练时当前的 epoch 数。

样例：

```
>>> # Create a synchronization callback
>>> import mindspore as ms
>>> from mindspore.dataset import sync_wait_for_dataset
>>>
>>> class SyncForDataset(ms.Callback):
...     def __init__(self):
...         super(SyncForDataset, self).__init__()
...     def epoch_begin(self, run_context):
...         cb_params = run_context.original_args()
...         epoch_num = cb_params.cur_epoch_num
...         sync_wait_for_dataset(rank_id, rank_size, epoch_num)
```

12.9.13 mindspore.dataset.utils.imshow_det_bbox

`mindspore.dataset.utils.imshow_det_bbox (image, bboxes, labels, segm=None, class_names=None, score_threshold=0, bbox_color=(0, 255, 0), text_color=(203, 192, 255), mask_color=(128, 0, 128), thickness=2, font_size=0.8, show=True, win_name='win', wait_time=2000, out_file=None)`

使用给定的边界框和类别置信度绘制图像。

参数：

- **image** (numpy.ndarray) - 待绘制的图像，shape 为 $\langle C, H, W \rangle$ 或 $\langle H, W, C \rangle$ ，通道顺序为 RGB。
- **bboxes** (numpy.ndarray) - 边界框 (包含类别置信度)，shape 为 $\langle N, 4 \rangle$ 或 $\langle N, 5 \rangle$ ，格式为 $\langle N, X, Y, W, H \rangle$ 。
- **labels** (numpy.ndarray) - 边界框的类别，shape 为 $\langle N, 1 \rangle$ 。
- **segm** (numpy.ndarray) - 图像分割掩码，shape 为 $\langle M, H, W \rangle$ ，M 表示类别总数。默认值：None，不绘制掩码。
- **class_names** (list[str], tuple[str], dict) - 类别索引到类别名的映射表。默认值：None，仅显示类别索引。
- **score_threshold** (float) - 绘制边界框的类别置信度阈值。默认值：0，绘制所有边界框。
- **bbox_color** (tuple(int)) - 指定绘制边界框时线条的颜色，顺序为 BGR。默认值：(0, 255, 0)，表示绿色。
- **text_color** (tuple(int)) - 指定类别文本的显示颜色，顺序为 BGR。默认值：(203, 192, 255)，表示粉色。
- **mask_color** (tuple(int)) - 指定掩码的显示颜色，顺序为 BGR。默认值：(128, 0, 128)，表示紫色。
- **thickness** (int) - 指定边界框和类别文本的线条粗细。默认值：2。
- **font_size** (int, float) - 指定类别文本字体大小。默认值：0.8。
- **show** (bool) - 是否显示图像。默认值：True。
- **win_name** (str) - 指定窗口名称。默认值："win"。
- **wait_time** (int) - 等待按键输入的时延，单位为 ms。在图像显示过程中，若无按键输入，则等待该时延后跳转到下一图像；若输入 ESC，则立刻结束显示；若输入其他按键，则停止等待直接跳转下一图像。默认值：2000，等待 2000ms 跳转下一图像。
- **out_file** (str, 可选) - 输出图像的文件路径，用于在绘制后将结果存储到本地。默认值：None，不保存。

返回：

numpy.ndarray，带边界框和类别置信度的图像。

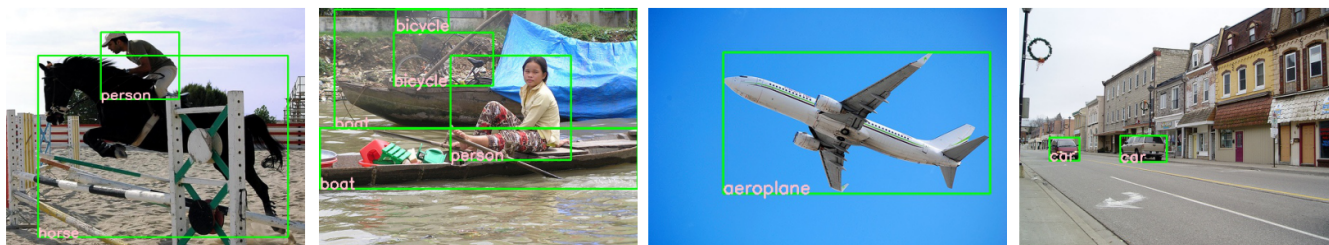
说明：该接口依赖 *opencv-python* 库。

异常：

- **ImportError** - 当未安装 *opencv-python* 库。
- **AssertionError** - 当 *image* 的格式不为 $\langle H, W, C \rangle$ 或 $\langle C, H, W \rangle$ 。
- **AssertionError** - 当 *bboxes* 的格式不为 $\langle N, 4 \rangle$ 或 $\langle N, 5 \rangle$ 。
- **AssertionError** - 当 *labels* 的格式不为 $\langle N, 1 \rangle$ 。
- **AssertionError** - 当 *segm* 的格式不为 $\langle M, H, W \rangle$ 。
- **AssertionError** - 当 *class_names* 的类型不为 list、tuple 或 dict。
- **AssertionError** - 当 *bbox_color* 不为 (B, G, R) 格式的 tuple。

- **AssertionError** - 当 `text_color` 不为 (B, G, R) 格式的 tuple。
- **AssertionError** - 当 `mask_color` 不为 (B, G, R) 格式的 tuple。

`imshow_det_bbox` 在 VOC2012 数据集的使用图示：



样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> from mindspore.dataset.utils import imshow_det_bbox
>>>
>>> # Read Detection dataset, such as VOC2012.
>>> voc_dataset_dir = "/path/to/voc_dataset_directory"
>>> dataset = ds.VOCDataset(voc_dataset_dir, task="Detection", shuffle=False, decode=True,
↪ num_samples=5)
>>> dataset_iter = dataset.create_dict_iterator(output_numpy=True, num_epochs=1)
>>>
>>> # draw dataset
>>> for index, data in enumerate(dataset_iter):
...     image = data["image"]
...     bbox = data["bbox"]
...     label = data["label"]
...     # draw image with bboxes
...     imshow_det_bbox(image, bbox, label,
...                       class_names=['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus',
↪ 'car', 'cat',
...                                   'chair', 'cow', 'diningtable', 'dog', 'horse',
↪ 'motorbike', 'person',
...                                   'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor'],
...     win_name="my_window",
...     wait_time=5000,
...     show=True,
...     out_file="voc_dataset_{}.jpg".format(str(index)))
```

12.9.14 mindspore.dataset.utils.LineReader

class mindspore.dataset.utils.**LineReader** (*filename*)

基于行的文件读取器。

通过提前缓存文件基于行的元信息，实现对文件各行的随机访问读取。

参数：

- **filename** (str) - 待读取的文件名。

异常：

- **TypeError** - *filename* 无效。

- **RuntimeError** - *filename* 不存在或者不是常规文件。

样例:

```
>>> from mindspore.dataset import LineReader
>>>
>>> reader = LineReader("/path/to/txt/or/csv/file")
>>> # Read the first line of csv file
>>> reader.readline(1)
>>> # Return the row size in csv file
>>> reader.len()
>>> # Close the handle
>>> reader.close()
```

close()

关闭文件句柄。

len()

获取当前文件的总行数。

readline(*line*)

读取指定行的内容。

参数:

- **line** (int) - 待读取的行号, 起始行号为 1。

返回:

str, 对应行的内容, 不包含换行符。

异常:

- **TypeError** - 如果 *line* 不是 int 类型。
- **ValueError** - 如果 *line* 取值大于文件总行数。

13.1 通用

此模块用于通用数据增强，其中一部分增强操作是用 C++ 实现的，具有较好的高性能，另一部分是基于 Python 实现，使用了 NumPy 模块作为支持。

在 API 示例中，常用的模块导入方法如下：

```
import mindspore.dataset as ds
import mindspore.dataset.transforms as transforms
```

常用数据处理术语说明如下：

- TensorOperation，所有 C++ 实现的数据处理操作的基类。
- PyTensorOperation，所有 Python 实现的数据处理操作的基类。

13.1.1 变换

<i>mindspore.dataset.transforms.Compose</i>	将多个数据增强操作组合使用。
<i>mindspore.dataset.transforms.Concatenate</i>	在输入数据的某一个轴上进行数组拼接，目前仅支持拼接形状为 1D 的数组。
<i>mindspore.dataset.transforms.Duplicate</i>	将输入的数据列复制得到新的数据列，每次仅可以输入 1 个数据列进行复制。
<i>mindspore.dataset.transforms.Fill</i>	将 Tensor 的所有元素都赋值为指定的值。
<i>mindspore.dataset.transforms.Mask</i>	用给条件判断输入 Tensor 的内容，并返回一个掩码 Tensor。
<i>mindspore.dataset.transforms.OneHot</i>	对输入标签进行 OneHot 编码。
<i>mindspore.dataset.transforms.PadEnd</i>	对输入 Tensor 进行填充，要求 <i>pad_shape</i> 与输入 Tensor 的维度保持一致。

下页继续

表 1 - 续上页

<code>mindspore.dataset.transforms.RandomApply</code>	指定一组数据增强处理及其被应用的概率，在运算时按概率随机应用其中的增强处理。
<code>mindspore.dataset.transforms.RandomChoice</code>	从一组数据增强变换中随机选择一个进行应用。
<code>mindspore.dataset.transforms.RandomOrder</code>	给一个数据增强的列表，随机打乱数据增强处理的顺序。
<code>mindspore.dataset.transforms.Slice</code>	对输入进行切片。
<code>mindspore.dataset.transforms.TypeCast</code>	将输入的 Tensor 转换为指定的数据类型。
<code>mindspore.dataset.transforms.Unique</code>	对输入张量进行唯一运算，每次只支持对一个数据列进行变换。

mindspore.dataset.transforms.Compose

class `mindspore.dataset.transforms.Compose` (*transforms*)
将多个数据增强操作组合使用。

说明: `Compose` 可以将 `mindspore.dataset.transforms` / `mindspore.dataset.vision` 等模块中的数据增强操作，以及用户自定义的 Python 可调用对象合并成单个数据增强。对于用户定义的 Python 可调用对象，要求其返回值是 `numpy.ndarray` 类型。

参数:

- **transforms** (list) - 一个数据增强的列表。

异常:

- **TypeError** - 参数 *transforms* 类型不为 list。
- **ValueError** - 参数 *transforms* 是空的 list。
- **TypeError** - 参数 *transforms* 的元素不是 Python 的可调用对象或 `audio/text/transforms/vision` 模块中的数据增强方法。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Relational
>>>
>>> # Use the transform in dataset pipeline mode
>>> # create a dataset that reads all files in dataset_dir with 8 threads
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>>
>>> # create a list of transformations to be applied to the image data
>>> transform = transforms.Compose([
...     vision.RandomHorizontalFlip(0.5),
...     vision.ToTensor(),
...     vision.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262), is_hwc=False),
...     vision.RandomErasing()])
```

(下页继续)

(续上页)

```

>>> # apply the transform to the dataset through dataset.map function
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transform, input_columns=[
    ↪ "image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Compose is also be invoked implicitly, by just passing in a list of ops
>>> # the above example then becomes:
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomHorizontalFlip(0.5),
...                     vision.ToTensor(),
...                     vision.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262), is_
    ↪ hwc=False),
...                     vision.RandomErasing() ]
>>>
>>> # apply the transform to the dataset through dataset.map()
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↪ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Certain C++ and Python ops can be combined, but not all of them
>>> # An example of combined operations
>>> arr = [0, 1]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(arr, column_names=["cols"], shuffle=False)
>>> transformed_list = [transforms.OneHot(2),
...                      transforms.Mask(transforms.Relational.EQ, 1)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transformed_list, input_
    ↪ columns=["cols"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["cols"].shape, item["cols"].dtype)
...     break
(2,) bool
>>>
>>> # Here is an example of mixing vision ops
>>> op_list=[vision.Resize((224, 244)),
...          vision.ToPIL(),
...          np.array, # need to convert PIL image to a NumPy array to pass it to C++
    ↪ operation
...          vision.Resize((24, 24))]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=op_list, input_columns=[
    ↪ "image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(24, 24, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([1, 2, 3])
>>> output = transforms.Compose([transforms.Fill(10), transforms.Mask(Relational.EQ,
    ↪ 100)]) (data)

```

(下页继续)

(续上页)

```
>>> print(output.shape, output.dtype)
(3,) bool
```

mindspore.dataset.transforms.Concatenate

class mindspore.dataset.transforms.**Concatenate** (*axis=0, prepend=None, append=None*)

在输入数据的某一个轴上进行数组拼接，目前仅支持拼接形状为 1D 的数组。

参数：

- **axis** (int, 可选) - 指定一个轴用于拼接数组。默认值：0。
- **prepend** (numpy.ndarray, 可选) - 想要拼接到输入数据首部的数组。默认值：None，不在首部拼接数组。
- **append** (numpy.ndarray, 可选) - 想要拼接到输入数据尾部的数组。默认值：None，不在尾部拼接数组。

异常：

- **TypeError** - 参数 *axis* 的类型不为 int。
- **TypeError** - 参数 *prepend* 的类型不为 *numpy.ndarray*。
- **TypeError** - 参数 *append* 的类型不为 *numpy.ndarray*。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> # concatenate string
>>> prepend_tensor = np.array(["dw", "df"])
>>> append_tensor = np.array(["dwsdf", "df"])
>>> concatenate_op = transforms.Concatenate(0, prepend_tensor, append_tensor)
>>> data = [{"This", "is", "a", "string"}]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=concatenate_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["column_0"].shape, item["column_0"].dtype)
(8,) <U6
>>>
>>> # Use the transform in eager mode
>>> data = np.array([1, 2, 3])
>>> prepend_tensor = np.array([10, 20])
>>> append_tensor = np.array([100])
>>> output = transforms.Concatenate(0, prepend_tensor, append_tensor)(data)
>>> print(output.shape, output.dtype)
(6,) int64
```

`mindspore.dataset.transforms.Duplicate`

`class mindspore.dataset.transforms.Duplicate`

将输入的数据列复制得到新的数据列，每次仅可以输入 1 个数据列进行复制。

异常：

- **RuntimeError** - 输入数据列数量大于 1。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> # Data before
>>> # | x      |
>>> # +-----+
>>> # | [1,2,3] |
>>> # +-----+
>>> data = [[1,2,3]]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["x"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms.Duplicate(),
...                                              input_columns=["x"],
...                                              output_columns=["x", "y"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["x"].shape, item["x"].dtype)
...     print(item["y"].shape, item["y"].dtype)
(3,) int64
(3,) int64
>>> # Data after
>>> # | x      | y      |
>>> # +-----+-----+
>>> # | [1,2,3] | [1,2,3] |
>>> # +-----+-----+
>>>
>>> # Use the transform in eager mode
>>> data = [1, 2, 3]
>>> output = transforms.Duplicate()(data)
>>> print(np.array(output).shape, np.array(output).dtype)
(2, 3) int64
```

`mindspore.dataset.transforms.Fill`

`class mindspore.dataset.transforms.Fill (fill_value)`

将 Tensor 的所有元素都赋值为指定的值。输出 Tensor 将与输入 Tensor 具有相同的 shape 和数据类型。

参数：

- **fill_value** (Union[str, bytes, int, float, bool]) - 用于填充 Tensor 的值。

异常：

- **TypeError** - 参数 `fill_value` 类型不为 str、float、bool、int 或 bytes。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> # generate a 1D integer numpy array from 0 to 4
>>> def generator_1d():
...     for i in range(5):
...         yield (np.array([i]),)
>>> generator_dataset = ds.GeneratorDataset(generator_1d, column_names="col1")
>>> # [[0], [1], [2], [3], [4]]
>>> fill_op = transforms.Fill(3)
>>> generator_dataset = generator_dataset.map(operations=fill_op)
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["col1"].shape, item["col1"].dtype)
...     break
(1,) int64
>>>
>>> # Use the transform in eager mode
>>> data = np.array([1, 2, 3])
>>> output = transforms.Fill(100)(data)
>>> print(output.shape, output.dtype)
(3,) int64
```

mindspore.dataset.transforms.Mask

class mindspore.dataset.transforms.**Mask** (*operator, constant, dtype=mstype.bool_*)

用给条件判断输入 Tensor 的内容, 并返回一个掩码 Tensor。Tensor 中任何符合条件的元素都将被标记为 True, 否则为 False。

参数:

- **operator** (*Relational*)- 关系操作符, 可以取值为 `Relational.EQ`、`Relational.NE`、`Relational.LT`、`Relational.GT`、`Relational.LE`、`Relational.GE`。以 `Relational.EQ` 为例, 将找出 Tensor 中与 *constant* 相等的元素。
- **constant** (`Union[str, int, float, bool]`) - 与输入 Tensor 进行比较的基准值。
- **dtype** (*mindspore.dtype*, 可选) - 生成的掩码 Tensor 的数据类型。默认值: `mstype.bool_`。

异常:

- **TypeError** - 参数 *operator* 类型不为 `mindspore.dataset.transforms.Relational`。
- **TypeError** - 参数 *constant* 类型不为 `str`、`int`、`float` 或 `bool`。
- **TypeError** - 参数 *dtype* 类型不为 `mindspore.dtype`。

支持平台:

CPU

样例:

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>> from mindspore.dataset.transforms import Relational
>>>
>>> # Use the transform in dataset pipeline mode
>>> # Data before
>>> # | col |
>>> # +-----+
>>> # | [1,2,3] |
>>> # +-----+
>>> data = [[1, 2, 3]]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["col"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms.Mask(Relational.EQ,
↪ 2))
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["col"].shape, item["col"].dtype)
(3,) bool
>>> # Data after
>>> # | col |
>>> # +-----+
>>> # | [False,True,False] |
>>> # +-----+
>>>
>>> # Use the transform in eager mode
>>> data = [1, 2, 3]
>>> output = transforms.Mask(Relational.EQ, 2)(data)
>>> print(output.shape, output.dtype)
(3,) bool

```

mindspore.dataset.transforms.OneHot

class mindspore.dataset.transforms.**OneHot** (*num_classes, smoothing_rate=0.0*)

对输入标签进行 OneHot 编码。

对于 shape 为 (*) 的 1 维输入，将返回 shape 为 (*, num_classes) 的输出，其中输入值对应的索引位置处的元素值为 1，其余位置值为 0。若指定了标签平滑系数，还将进一步平滑各元素值，增强泛化能力。

参数：

- **num_classes** (int) - 标签类别总数。需大于输入标签值的最大值。
- **smoothing_rate** (float, 可选) - 标签平滑系数。取值需在 [0.0, 1.0] 之间。默认值：0.0，不进行标签平滑。

异常：

- **TypeError** - 当 *num_classes* 不为 int 类型。
- **TypeError** - 当 *smoothing_rate* 不为 float 类型。
- **ValueError** - 当 *smoothing_rate* 的取值不在 [0.0, 1.0] 范围中。
- **RuntimeError** - 输入标签不为 int 类型。
- **RuntimeError** - 输入标签的维数不为 1。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = [1, 2, 3, 4, 5, 6, 7, 8]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["label"])
>>>
>>> # Assume that dataset has 10 classes, thus the label ranges from 0 to 9
>>> onehot_op = transforms.OneHot(num_classes=10)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=onehot_op, input_columns=[
    ↪ "label"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["label"].shape, item["label"].dtype)
...     break
(10,) int64
>>>
>>> # Use the transform in eager mode
>>> data = np.array([1, 2, 3])
>>> output = transforms.OneHot(num_classes=5, smoothing_rate=0)(data)
>>> print(output.shape, output.dtype)
(3, 5) int64

```

mindspore.dataset.transforms.PadEnd

class mindspore.dataset.transforms.**PadEnd** (*pad_shape, pad_value=None*)

对输入 Tensor 进行填充, 要求 *pad_shape* 与输入 Tensor 的维度保持一致。

参数:

- **pad_shape** (list(int)) - 指定填充的 shape。维度设置为 None 时将不会被填充, 设置为较小的维数时该维度的元素将被截断。
- **pad_value** (Union[str, bytes, int, float, bool], 可选) - 用于填充的值。默认值: None, 表示不指定填充值。当指定为默认值, 输入 Tensor 为数值型时默认填充 0, 输入 Tensor 为字符型时填充空字符串。

异常:

- **TypeError** - 参数 *pad_shape* 的类型不为 list。
- **TypeError** - 参数 *pad_value* 的类型不为 str、float、bool、int 或 bytes。
- **TypeError** - 参数 *pad_shape* 的元素类型不为 int。
- **ValueError** - 参数 *pad_shape* 的元素不为正数。

支持平台:

CPU

样例:

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> # Data before
>>> # |   col   |
>>> # +-----+

```

(下页继续)

(续上页)

```

>>> # | [1,2,3] |
>>> # +-----/
>>> data = [[1, 2, 3]]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["col"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms.PadEnd(pad_
->shape=[4], pad_value=10))
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["col"].shape, item["col"].dtype)
...     break
(4,) int64
>>> # Data after
>>> # |   col   |
>>> # +-----+
>>> # | [1,2,3,10] |
>>> # +-----/
>>>
>>> # Use the transform in eager mode
>>> data = [1, 2, 3]
>>> output = transforms.PadEnd(pad_shape=[4], pad_value=10)(data)
>>> print(output.shape, output.dtype)
(4,) int64

```

mindspore.dataset.transforms.RandomApply

class mindspore.dataset.transforms.**RandomApply** (*transforms, prob=0.5*)

指定一组数据增强处理及其被应用的概率，在运算时按概率随机应用其中的增强处理。

参数：

- **transforms** (list) - 一个数据增强的列表。
- **prob** (float, 可选) - 随机应用某个数据增强的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 参数 *transforms* 类型不为 list。
- **ValueError** - 参数 *transforms* 的长度为空。
- **TypeError** - 参数 *transforms* 的元素不是 Python 可调用对象或 audio/text/transforms/vision 模块中的数据处理操作。
- **TypeError** - 参数 *prob* 的类型不为 float。
- **ValueError** - 参数 *prob* 的取值范围不为 [0.0, 1.0]。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode

```

(下页继续)

(续上页)

```

>>> seed = ds.config.get_seed()
>>> ds.config.set_seed(12345)
>>> transforms_list = [vision.RandomHorizontalFlip(0.5),
...                     vision.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262)),
...                     vision.RandomErasing() ]
>>> composed_transform = Compose([transforms.RandomApply(transforms_list, prob=0.6),
...                               vision.ToTensor()])
>>>
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=composed_transform, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> transform = [vision.HsvToRgb(is_hwc=True), vision.Crop((0, 0), 10), vision.ToTensor()]
>>> output = transforms.RandomApply(transform, prob=1.0)(data)
>>> print(output.shape, output.dtype)
(3, 10, 10) float32
>>> ds.config.set_seed(seed)

```

mindspore.dataset.transforms.RandomChoice

class mindspore.dataset.transforms.RandomChoice (transforms)

从一组数据增强变换中随机选择一个进行应用。

参数:

- **transforms** (list) - 可供选择的数据增强变换列表。

异常:

- **TypeError** - 参数 *transforms* 类型不为 list。
- **ValueError** - 参数 *transforms* 为空。
- **TypeError** - 参数 *transforms* 的元素不是 Python 可调对象或 audio/text/transforms/vision 模块中的数据处理操作。

支持平台:

CPU

样例:

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> seed = ds.config.get_seed()
>>> ds.config.set_seed(12345)

```

(下页继续)

(续上页)

```

>>> transforms_list = [vision.RandomHorizontalFlip(0.5),
...                     vision.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262)),
...                     vision.RandomErasing()]
>>> composed_transform = Compose([transforms.RandomChoice(transforms_list),
...                               vision.ToTensor()])
>>>
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=composed_transform, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.array([1, 2, 3])
>>> output = transforms.RandomChoice([transforms.Fill(100)])(data)
>>> print(output.shape, output.dtype)
(3,) int64
>>> ds.config.set_seed(seed)

```

mindspore.dataset.transforms.RandomOrder

class mindspore.dataset.transforms.**RandomOrder** (*transforms*)

给一个数据增强的列表，随机打乱数据增强处理的顺序。

参数：

- **transforms** (list) - 一个数据增强的列表。

异常：

- **TypeError** - 参数 *transforms* 类型不为 list。
- **TypeError** - 参数 *transforms* 的元素不是 Python 可调用对象或 audio/text/transforms/vision 模块中的数据处理操作。
- **ValueError** - 参数 *transforms* 是空的 list。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose, Relational
>>>
>>> # Use the transform in dataset pipeline mode
>>> seed = ds.config.get_seed()
>>> ds.config.set_seed(12345)
>>> transforms_list = [vision.RandomHorizontalFlip(0.5),
...                     vision.Normalize((0.491, 0.482, 0.447), (0.247, 0.243, 0.262)),
...                     vision.RandomErasing()]

```

(下页继续)

(续上页)

```

>>> composed_transform = Compose([transforms.RandomOrder(transforms_list),
...                               vision.ToTensor()])
>>>
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=composed_transform, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.array([1, 2, 3])
>>> output = transforms.RandomOrder([transforms.Mask(Relational.EQ, 100)])(data)
>>> print(output.shape, output.dtype)
(3,) bool
>>> ds.config.set_seed(seed)

```

mindspore.dataset.transforms.Slice

class mindspore.dataset.transforms.Slice(*slices)

对输入进行切片。

当前仅支持 1-D 输入。

参数：

- **slices** (Union[int, list[int], slice, Ellipsis]) - 想要切片的片段。
 - 若输入类型为 **int**，将切片指定索引值的元素，支持负索引。
 - 若输入类型为 **list[int]**，将切片所有指定索引值的元素，支持负索引。
 - 若输入类型为 **slice**，将根据其指定的起始位置、结束位置和步长进行切片。
 - 若输入类型为 **Ellipsis**，即省略符，将切片所有元素。
 - 若输入为 **None**，将切片所有元素。

异常：

- **TypeError** - 当 *slices* 不为 Union[int, list[int], slice, Ellipsis] 类型。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> # Data before
>>> # |   col   |
>>> # +-----+
>>> # | [1,2,3] |

```

(下页继续)

(续上页)

```

>>> # +-----/
>>> data = [[1, 2, 3]]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["col"])
>>> # slice indices 1 and 2 only
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms.Slice(slice(1,3)))
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["col"].shape, item["col"].dtype)
...     break
(2,) int64
>>> # Data after
>>> # |   col   |
>>> # +-----+
>>> # | [2,3] |
>>> # +-----/
>>>
>>> # Use the transform in eager mode
>>> data = np.array([1, 2, 3])
>>> output = transforms.Slice(slice(1, 3))(data)
>>> print(output.shape, output.dtype)
(2,) int64

```

mindspore.dataset.transforms.TypeCast

class mindspore.dataset.transforms.**TypeCast** (*data_type*)

将输入的 Tensor 转换为指定的数据类型。

说明：此操作默认通过 CPU 执行，也支持异构加速到 GPU 或 Ascend 上执行。

参数：

- **data_type** (Union[mindspore.dtype, numpy.dtype]) - 指定要转换的数据类型。

异常：

- **TypeError** - 当 *data_type* 的类型不为 *mindspore.dtype* 或 *numpy.dtype* 。

支持平台：

CPU GPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>> from mindspore import dtype as mstype
>>>
>>> # Use the transform in dataset pipeline mode
>>> # Generate 1d int numpy array from 0 - 63
>>> def generator_1d():
...     for i in range(64):
...         yield (np.array([i]),)
>>>
>>> generator_dataset = ds.GeneratorDataset(generator_1d, column_names='col')

```

(下页继续)

(续上页)

```

>>> type_cast_op = transforms.TypeCast(mstype.int32)
>>> generator_dataset = generator_dataset.map(operations=type_cast_op)
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["col"].shape, item["col"].dtype)
...     break
(1,) int32
>>>
>>> # Use the transform in eager mode
>>> data = np.array([2.71606445312564e-03, 6.3476562564e-03]).astype(np.float64)
>>> output = transforms.TypeCast(np.float16)(data)
>>> print(output.shape, output.dtype)
(2,) float16

```

mindspore.dataset.transforms.Unique

class mindspore.dataset.transforms.Unique

对输入张量进行唯一运算，每次只支持对一个数据列进行变换。

Unique 将返回 3 个 Tensor: 运算结果 Tensor, 索引 Tensor, 计数 Tensor。

- 运算结果 Tensor 包含输入张量的所有唯一元素，且和输入张量的顺序是一样的。
- 索引 Tensor 包含输入 Tensor 的每个元素在运算结果中的索引。
- 计数 Tensor 包含运算结果 Tensor 的每个元素的计数。

说明：需要在 *batch* 操作之后调用该运算。

异常：

- **RuntimeError** - 当输入的 Tensor 具有两列。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> # Data before
>>> # | x |
>>> # +-----+
>>> # | [[0,1,2], [1,2,3]] |
>>> # +-----+
>>> data = [[[0,1,2], [1,2,3]]]
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["x"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms.Unique(),
...                                                input_columns=["x"],
...                                                output_columns=["x", "y", "z"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["x"].shape, item["y"].shape, item["z"].shape)

```

(下页继续)

(续上页)

```

...     print(item["x"].dtype, item["y"].dtype, item["z"].dtype)
(4,) (6,) (4,)
int64 int32 int32
>>> # Data after
>>> # | x          | y          | z          |
>>> # +-----+-----+-----+
>>> # | [0,1,2,3] | [0,1,2,1,2,3] | [1,2,2,1] |
>>> # +-----+-----+-----+
>>>
>>> # Use the transform in eager mode
>>> data = [[0, -1, -2, -1, 2], [2, -0, 2, 1, -3]]
>>> output = transforms.Unique()(data)
>>> print(output[0].shape, output[1].shape, output[2].shape)
(6,) (10,) (6,)
>>> print(output[0].dtype, output[1].dtype, output[2].dtype)
int64 int32 int32

```

13.1.2 工具

`mindspore.dataset.transforms.Relational`

关系运算符。

`mindspore.dataset.transforms.Relational`

class `mindspore.dataset.transforms.Relational`

关系运算符。

可选值如下:

- `Relational.EQ`: 相等。
- `Relational.NE`: 不相等。
- `Relational.GT`: 大于。
- `Relational.GE`: 大于等于。
- `Relational.LT`: 小于。
- `Relational.LE`: 小于等于。

13.2 视觉

此模块用于图像数据增强，其中有一部分增强是基于 C++ OpenCV 实现的，具有较好的性能，而另一部分是基于 Python Pillow 实现的。

API 样例中常用的导入模块如下:

```

import mindspore.dataset as ds
import mindspore.dataset.vision as vision
import mindspore.dataset.vision.utils as utils

```

常用数据处理术语说明如下:

- `TensorOperation`, 所有 C++ 实现的数据处理操作的基类。
- `ImageTensorOperation`, 所有图像类数据处理操作的基类, 继承自 `TensorOperation`。
- `PyTensorOperation`, 所有 Python 实现的数据处理操作的基类。

数据增强操作可以放入数据处理 Pipeline 中执行, 也可以 Eager 模式执行:

- Pipeline 模式用于流式处理大型数据集, 示例可参考 [数据处理 Pipeline 介绍](#)。
- Eager 模式用于函数调用方式处理样本, 示例可参考 [轻量化数据处理](#)。

13.2.1 样例库

快速上手使用视觉类变换的 API, 跳转参考 [视觉变换样例库](#)。此指南中展示了多个变换 API 的用法, 以及输入输出结果。

13.2.2 变换

<code>mindspore.dataset.vision.AdjustBrightness</code>	调整输入图像的亮度。
<code>mindspore.dataset.vision.AdjustContrast</code>	调整输入图像的对比度。
<code>mindspore.dataset.vision.AdjustGamma</code>	对输入图像应用伽马校正。
<code>mindspore.dataset.vision.AdjustHue</code>	调整输入图像的色调。
<code>mindspore.dataset.vision.AdjustSaturation</code>	调整输入图像的饱和度。
<code>mindspore.dataset.vision.AdjustSharpness</code>	调整输入图像的锐度。
<code>mindspore.dataset.vision.Affine</code>	对输入图像进行仿射变换, 保持图像中心不动。
<code>mindspore.dataset.vision.AutoAugment</code>	应用 <code>AutoAugment</code> 数据增强方法, 基于论文 AutoAugment: Learning Augmentation Strategies from Data 。
<code>mindspore.dataset.vision.AutoContrast</code>	在输入图像上自动调整对比度。
<code>mindspore.dataset.vision.BoundingBoxAugment</code>	对图像的随机标注边界框区域, 应用给定的图像变换处理。
<code>mindspore.dataset.vision.CenterCrop</code>	对输入图像应用中心区域裁剪。
<code>mindspore.dataset.vision.ConvertColor</code>	更改图像的色彩空间。
<code>mindspore.dataset.vision.Crop</code>	在输入图像上裁剪出指定区域。
<code>mindspore.dataset.vision.CutMixBatch</code>	对输入批次的图像和标注应用剪切混合转换。
<code>mindspore.dataset.vision.CutOut</code>	从输入图像数组中, 随机裁剪出给定数量的正方形区域。
<code>mindspore.dataset.vision.Decode</code>	将输入的压缩图像解码为 RGB 格式。
<code>mindspore.dataset.vision.DecodeVideo</code>	将输入的视频原始字节解码为视频、音频。
<code>mindspore.dataset.vision.Equalize</code>	对输入图像进行直方图均衡化。
<code>mindspore.dataset.vision.Erase</code>	擦除输入图像中指定的区域, 并填充指定像素值 <code>value</code> 。
<code>mindspore.dataset.vision.FiveCrop</code>	在输入 PIL 图像的中心与四个角处, 分别裁剪指定尺寸大小的子图。
<code>mindspore.dataset.vision.GaussianBlur</code>	使用指定的高斯核对输入图像进行模糊处理。
<code>mindspore.dataset.vision.Grayscale</code>	将输入 PIL 图像转换为灰度图。
<code>mindspore.dataset.vision.HorizontalFlip</code>	水平翻转输入图像。
<code>mindspore.dataset.vision.HsvToRgb</code>	将输入的 HSV 格式 <code>numpy.ndarray</code> 图像转换为 RGB 格式。
<code>mindspore.dataset.vision.HWC2CHW</code>	将输入图像的 shape 从 <code><H, W, C></code> 转换为 <code><C, H, W></code> 。
<code>mindspore.dataset.vision.Invert</code>	对输入的 RGB 图像进行色彩反转。
<code>mindspore.dataset.vision.LinearTransformation</code>	使用指定的变换方阵和均值向量, 对输入 <code>numpy.ndarray</code> 图像进行线性变换。
<code>mindspore.dataset.vision.MixUp</code>	随机混合一批输入的 <code>numpy.ndarray</code> 图像及其标签。
<code>mindspore.dataset.vision.MixUpBatch</code>	对输入批次的图像和标注应用混合转换。

下页继续

表 3 - 续上页

<code>mindspore.dataset.vision.Normalize</code>	根据均值和标准差, 对输入图像进行归一化。
<code>mindspore.dataset.vision.NormalizePad</code>	根据均值和标准差对输入图像进行归一化, 然后填充一个全零的额外通道。
<code>mindspore.dataset.vision.Pad</code>	填充图像。
<code>mindspore.dataset.vision.PadToSize</code>	将图像填充到固定大小。
<code>mindspore.dataset.vision.Perspective</code>	对输入图像进行透视变换。
<code>mindspore.dataset.vision.Posterize</code>	减少图像的颜色通道的比特位数, 使图像变得高对比度和颜色鲜艳, 类似于海报或印刷品的效果。
<code>mindspore.dataset.vision.RandAugment</code>	对输入图像应用 RandAugment 数据增强方法。
<code>mindspore.dataset.vision. RandomAdjustSharpness</code>	以给定的概率, 随机调整输入图像的锐度。
<code>mindspore.dataset.vision.RandomAffine</code>	对输入图像应用随机仿射变换。
<code>mindspore.dataset.vision. RandomAutoContrast</code>	以给定的概率, 自动调整图像的对比度。
<code>mindspore.dataset.vision.RandomColor</code>	随机调整输入图像的颜色。
<code>mindspore.dataset.vision. RandomColorAdjust</code>	随机调整输入图像的亮度、对比度、饱和度和色调。
<code>mindspore.dataset.vision.RandomCrop</code>	对输入图像进行随机区域的裁剪。
<code>mindspore.dataset.vision. RandomCropDecodeResize</code>	”裁剪”、”解码”和”调整尺寸大小”的组合处理。
<code>mindspore.dataset.vision. RandomCropWithBBox</code>	在输入图像的随机位置进行裁剪, 并相应地调整边界框。
<code>mindspore.dataset.vision.RandomEqualize</code>	以给定的概率, 随机对输入图像进行直方图均衡化。
<code>mindspore.dataset.vision.RandomErasing</code>	以给定的概率, 擦除输入 <code>numpy.ndarray</code> 图像上随机矩形区域内的像素。
<code>mindspore.dataset.vision.RandomGrayscale</code>	以给定的概率, 将输入 PIL 图像转换为灰度图。
<code>mindspore.dataset.vision. RandomHorizontalFlip</code>	以给定的概率, 对输入图像进行水平随机翻转。
<code>mindspore.dataset.vision. RandomHorizontalFlipWithBBox</code>	以给定的概率, 对输入图像及其边界框进行随机水平翻转。
<code>mindspore.dataset.vision.RandomInvert</code>	以给定的概率, 随机反转图像的颜色。
<code>mindspore.dataset.vision.RandomLighting</code>	将 AlexNet PCA 的噪声添加到图像中。
<code>mindspore.dataset.vision. RandomPerspective</code>	以给定的概率, 对输入 PIL 图像进行透视变换。
<code>mindspore.dataset.vision.RandomPosterize</code>	随机减少图像的颜色通道的比特位数, 使图像变得高对比度和颜色鲜艳。
<code>mindspore.dataset.vision. RandomResizedCrop</code>	对输入图像进行随机裁剪, 并使用指定的 <code>mindspore.dataset.vision.Inter</code> 插值方式去调整为指定的尺寸大小。
<code>mindspore.dataset.vision. RandomResizedCropWithBBox</code>	对输入图像进行随机裁剪且随机调整纵横比, 并将处理后的图像调整为指定的尺寸大小, 并相应地调整边界框。
<code>mindspore.dataset.vision.RandomResize</code>	使用随机选择的 <code>mindspore.dataset.vision.Inter</code> 插值方式, 调整输入图像的尺寸大小。
<code>mindspore.dataset.vision. RandomResizeWithBBox</code>	对输入图像使用随机选择的 <code>mindspore.dataset.vision.Inter</code> 插值方式去调整它的尺寸大小, 并相应地调整边界框的尺寸大小。
<code>mindspore.dataset.vision.RandomRotation</code>	在指定的角度范围内, 随机旋转输入图像。
<code>mindspore.dataset.vision. RandomSelectSubpolicy</code>	从策略列表中, 随机选择一个子策略以应用于输入图像。
<code>mindspore.dataset.vision.RandomSharpness</code>	在固定或随机的范围, 调整输入图像的锐度。

下页继续

表 3 - 续上页

<code>mindspore.dataset.vision.RandomSolarize</code>	从给定阈值范围内随机选择一个子范围，对位于给定子范围内的像素，将其像素值设置为 (255 - 原本像素值)。
<code>mindspore.dataset.vision.RandomVerticalFlip</code>	以给定的概率，对输入图像在垂直方向进行随机翻转。
<code>mindspore.dataset.vision.RandomVerticalFlipWithBBox</code>	以给定的概率，对输入图像和边界框在垂直方向进行随机翻转。
<code>mindspore.dataset.vision.Rescale</code>	基于给定的缩放和平移因子，调整图像的像素大小。
<code>mindspore.dataset.vision.Resize</code>	使用给定的 <code>mindspore.dataset.vision.Inter</code> 插值方式，将输入图像调整至给定的尺寸大小。
<code>mindspore.dataset.vision.ResizedCrop</code>	裁剪输入图像的指定区域，并缩放至指定尺寸大小。
<code>mindspore.dataset.vision.ResizeWithBBox</code>	将输入图像调整为给定的尺寸大小，并相应地调整边界框的大小。
<code>mindspore.dataset.vision.RgbToHsv</code>	将输入的 RGB 格式 <code>numpy.ndarray</code> 图像转换为 HSV 格式。
<code>mindspore.dataset.vision.Rotate</code>	将输入图像旋转指定的度数。
<code>mindspore.dataset.vision.SlicePatches</code>	在水平和垂直方向上将 <code>Tensor</code> 切片为多个块。
<code>mindspore.dataset.vision.Solarize</code>	通过反转阈值内的所有像素值，对输入图像进行曝光。
<code>mindspore.dataset.vision.TenCrop</code>	在输入 PIL 图像的中心与四个角处，分别裁剪指定尺寸大小的子图，并将其翻转图一并返回。
<code>mindspore.dataset.vision.ToNumpy</code>	将输入 PIL 图像转换为 <code>numpy.ndarray</code> 图像。
<code>mindspore.dataset.vision.ToPIL</code>	将 <code>numpy.ndarray</code> 格式的解码图像转换为 <code>PIL.Image.Image</code> 格式的图像。
<code>mindspore.dataset.vision.ToTensor</code>	将输入 PIL 图像或 <code>numpy.ndarray</code> 图像转换为指定类型的 <code>numpy.ndarray</code> 图像，图像的像素值范围将从 [0, 255] 放缩为 [0.0, 1.0]，shape 将从 <H, W, C> 调整为 <C, H, W>。
<code>mindspore.dataset.vision.ToType</code>	将输入转换为指定的 MindSpore 数据类型或 NumPy 数据类型。
<code>mindspore.dataset.vision.TrivialAugmentWide</code>	对输入图像应用 <code>TrivialAugmentWide</code> 数据增强方法。
<code>mindspore.dataset.vision.UniformAugment</code>	从指定序列中均匀采样一批数据处理操作，并按顺序随机执行，即采样出的操作也可能不被执行。
<code>mindspore.dataset.vision.VerticalFlip</code>	对输入图像进行垂直翻转。

mindspore.dataset.vision.AdjustBrightness

class `mindspore.dataset.vision.AdjustBrightness` (*brightness_factor*)

调整输入图像的亮度。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **brightness_factor** (float) - 亮度调节因子，需为非负数。输入 0.0 值将得到全黑图像，1.0 值将得到原始图像，2.0 值将调整图像亮度为原来的 2 倍。

异常：

- **TypeError** - 如果 *brightness_factor* 不是 float 类型。
- **ValueError** - 如果 *brightness_factor* 小于 0.0。
- **RuntimeError** - 如果输入图像的形状不是 <H, W, C>。

支持平台：

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustBrightness(brightness_factor=2.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↪ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 256, (20, 20, 3)) / 255.0
>>> data = data.astype(np.float32)
>>> output = vision.AdjustBrightness(2.666)(data)
>>> print(output.shape, output.dtype)
(20, 20, 3) float32
```

教程样例:

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时, 输入数据的维度限制在 [4, 6] 和 [8192, 4096] 范围内。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustBrightness(2.0).device("Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↪ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↪ numpy=True):
```

(下页继续)

(续上页)

```

...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 256, (20, 20, 3)) / 255.0
>>> data = data.astype(np.float32)
>>> output = vision.AdjustBrightness(2.666).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(20, 20, 3) float32

```

mindspore.dataset.vision.AdjustContrast

class mindspore.dataset.vision.AdjustContrast (contrast_factor)

调整输入图像的对比度。

支持 Ascend 硬件加速, 需要通过 .device("Ascend") 方式开启。

参数:

- **contrast_factor** (float) - 对比度调节因子, 需为非负数。输入 0.0 值将得到灰度图像, 1.0 值将得到原始图像, 2.0 值将调整图像对比度为原来的 2 倍。

异常:

- **TypeError** - 如果 contrast_factor 不是 float 类型。
- **ValueError** - 如果 contrast_factor 小于 0.0 。
- **RuntimeError** - 如果输入图像的形状不是 <H, W, C>。

支持平台:

CPU Ascend

样例:

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustContrast(contrast_factor=2.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.AdjustContrast(2.0)(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8

```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据的维度限制在 [4, 6] 和 [8192, 4096] 范围内。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustContrast(0).device("Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
->numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.AdjustContrast(2.0).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

mindspore.dataset.vision.AdjustGamma

class mindspore.dataset.vision.**AdjustGamma** (*gamma*, *gain*=1)

对输入图像应用伽马校正。输入图片 shape 应该为 <..., H, W, C> 或 <H, W>。

$$I_{\text{out}} = 255 \times \text{gain} \times \left(\frac{I_{\text{in}}}{255} \right)^{\gamma}$$

更多详细信息，请参见 [Gamma 矫正](#)。

参数：

- **gamma** (float) - 非负实数。输出图像像素值与输入图像像素值呈指数相关。*gamma* 大于 1.0 使阴影更暗, 而 *gamma* 小于 1.0 使黑暗区域更亮。
- **gain** (float, 可选) - 常数乘数。默认值: 1.0。

异常:

- **TypeError** - 如果 *gain* 不是浮点类型。
- **TypeError** - 如果 *gamma* 不是浮点类型。
- **ValueError** - 如果 *gamma* 小于 0.0。
- **RuntimeError** - 如果给定的张量形状不是 <H, W> 或 <..., H, W, C>。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustGamma(gamma=10.0, gain=1.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.AdjustGamma(gamma=0.1, gain=1.0)(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8
```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.AdjustHue

class mindspore.dataset.vision.**AdjustHue** (*hue_factor*)

调整输入图像的色调。

支持 Ascend 硬件加速, 需要通过 *.device("Ascend")* 方式开启。

参数:

- **hue_factor** (float) - 色调通道调节值, 值必须在 [-0.5, 0.5] 范围内。

异常:

- **TypeError** - 如果 *hue_factor* 不是 float 类型。

- **ValueError** - 如果 *hue_factor* 的值不在 $[-0.5, 0.5]$ 的范围内。
- **RuntimeError** - 如果输入图像的形状不是 $\langle H, W, C \rangle$ 。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustHue(hue_factor=0.2)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.AdjustHue(hue_factor=0.2)(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据的维度限制在 $[4, 6]$ 和 $[8192, 4096]$ 范围内。

参数：

- **device_target** (str, 可选) - 算子将在该参数指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 $["CPU", "Ascend"]$ 。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
```

(下页继续)

(续上页)

```

>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustHue(0.5).device("Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.AdjustHue(hue_factor=0.2).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

`mindspore.dataset.vision.AdjustSaturation`

class `mindspore.dataset.vision.AdjustSaturation` (*saturation_factor*)

调整输入图像的饱和度。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **saturation_factor** (float) - 饱和度调节因子，需为非负数。输入 0.0 将得到全黑图像，1.0 将得到原始图像，2.0 将调整图像饱和度为原来的 2 倍。

异常：

- **TypeError** - 如果 *saturation_factor* 不是 float 类型。
- **ValueError** - 如果 *saturation_factor* 小于 0.0。
- **RuntimeError** - 如果输入图像的形状不是 <H, W, C>。
- **RuntimeError** - 如果输入图像的通道数不是 3。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustSaturation(saturation_factor=2.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)

```

(下页继续)

(续上页)

```

...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2, 6,
->3))
>>> output = vision.AdjustSaturation(saturation_factor=2.0)(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8

```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据的维度限制在 [4, 6] 和 [8192, 4096] 范围内。

参数：

- **device_target** (str, 可选) - 算子将在该参数指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustSaturation(2.0).device("Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
->numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.AdjustSaturation(saturation_factor=2.0).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

`mindspore.dataset.vision.AdjustSharpness`

class `mindspore.dataset.vision.AdjustSharpness` (*sharpness_factor*)

调整输入图像的锐度。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **sharpness_factor** (float) - 锐度调节因子，需为非负数。输入 0.0 值将得到模糊图像，1.0 值将得到原始图像，2.0 值将调整图像锐度为原来的 2 倍。

异常：

- **TypeError** - 如果 *sharpness_factor* 不是 float 类型。
- **ValueError** - 如果 *sharpness_factor* 小于 0.0。
- **RuntimeError** - 如果输入图像的形状不是 <H, W, C> 或 <H, W>。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> # create a dataset that reads all files in dataset_dir with 8 threads
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustSharpness(sharpness_factor=2.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((3, 4))
>>> output = vision.AdjustSharpness(sharpness_factor=0)(data)
>>> print(output.shape, output.dtype)
(3, 4) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据支持 `uint8` 或者 `float32` 类型，输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192]，宽度限制范围为 [6, 4096]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常:

- **TypeError** - 当 `device_target` 的类型不为 `str`。
- **ValueError** - 当 `device_target` 的取值不为 `["CPU", "Ascend"]`。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> # create a dataset that reads all files in dataset_dir with 8 threads
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AdjustSharpness(sharpness_factor=2.0).device("Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↵columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↵numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.AdjustSharpness(sharpness_factor=0).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

`mindspore.dataset.vision.Affine`

class `mindspore.dataset.vision.Affine` (*degrees, translate, scale, shear, resample=Inter.NEAREST, fill_value=0*)

对输入图像进行仿射变换, 保持图像中心不动。

支持 Ascend 硬件加速, 需要通过 `.device("Ascend")` 方式开启。

参数:

- **degrees** (float) - 顺时针的旋转角度, 取值需为 `-180.0` 到 `180.0` 之间。
- **translate** (`Sequence[float, float]`) - 水平和垂直方向上的平移长度, 需为 2 元素序列, 取值在 `-1.0` 和 `1.0` 之间。
- **scale** (float) - 放缩因子, 需为正数。
- **shear** (`Union[float, Sequence[float, float]]`) - 裁切度数, 取值需为 `-180.0` 到 `180.0` 之间。
 - 若输入单个数值, 表示平行于 X 轴的裁切角度, 不进行 Y 轴上的裁切。
 - 若输入序列 `[float, float]`, 分别表示平行于 X 轴和 Y 轴的裁切角度。
- **resample** (`Inter`, 可选) - 图像插值方法。默认值: `Inter.NEAREST`。
- **fill_value** (`Union[int, tuple[int, int, int]]`, 可选) - 用于填充输出图像中变换之外的区域。元组中必须有三个值, 每个值的取值范围是 `[0, 255]`。默认值: `0`。

异常:

- **TypeError** - 如果 *degrees* 不是 float 类型。
- **TypeError** - 如果 *translate* 不是 Sequence[float, float] 类型。
- **TypeError** - 如果 *scale* 不是 float 类型。
- **ValueError** - 如果 *scale* 非正数。
- **TypeError** - 如果 *shear* 不是 float 或 Sequence[float, float] 类型。
- **TypeError** - 如果 *resample* 不是 `mindspore.dataset.vision.Inter` 的类型。
- **TypeError** - 如果 *fill_value* 不是 int 或 tuple[int, int, int] 类型。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> affine_op = vision.Affine(degrees=15, translate=[0.2, 0.2], scale=1.1, shear=[1.0, 1.0],
...                           resample=Inter.BILINEAR)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[affine_op], input_columns=[
↪ "image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2, 6,
↪ 3))
>>> output = vision.Affine(degrees=15, translate=[0.2, 0.2], scale=1.1,
...                         shear=[1.0, 1.0], resample=Inter.BILINEAR)(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据的维度限制在 [4, 6] 和 [32768, 32768] 范围内。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。

- **ValueError** - 当 `device_target` 的取值不为 ["CPU" , "Ascend"]。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> affine_op = vision.Affine(degrees=15, translate=[0.2, 0.2], scale=1.1,
...                           shear=[1.0, 1.0], resample=Inter.BILINEAR).device("Ascend")
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[affine_op], input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳ numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Affine(degrees=15, translate=[0.2, 0.2], scale=1.1,
...                        shear=[1.0, 1.0], resample=Inter.BILINEAR).device("Ascend
↳ ") (data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

mindspore.dataset.vision.AutoAugment

class mindspore.dataset.vision.**AutoAugment** (*policy=AutoAugmentPolicy.IMAGENET, interpolation=Inter.NEAREST, fill_value=0*)

应用 AutoAugment 数据增强方法，基于论文 [AutoAugment: Learning Augmentation Strategies from Data](#)。此操作仅适用于 3 通道 RGB 图像。

参数:

- **policy** (*AutoAugmentPolicy*, 可选) - 在不同数据集上学习的 AutoAugment 策略。默认值: `AutoAugmentPolicy.IMAGENET`。可以是 `AutoAugmentPolicy.IMAGENET`、`AutoAugmentPolicy.CIFAR10`、`AutoAugmentPolicy.SVHN`。
 - **AutoAugmentPolicy.IMAGENET**: 表示应用在 ImageNet 数据集上学习的 AutoAugment。
 - **AutoAugmentPolicy.CIFAR10**: 表示应用在 Cifar10 数据集上学习的 AutoAugment。
 - **AutoAugmentPolicy.SVHN**: 表示应用在 SVHN 数据集上学习的 AutoAugment。
- **interpolation** (*Inter*, 可选) - 图像插值方法。可选值详见 `mindspore.dataset.vision.Inter`。默认值: `Inter.NEAREST`。
- **fill_value** (*Union[int, tuple[int]]*, 可选) - 填充的像素值。默认值: 0。
 - 如果是 3 元素元组，则分别用于填充 R、G、B 通道。
 - 如果是整数，则用于所有 RGB 通道。`fill_value` 值必须在 [0, 255] 范围内。

异常:

- **TypeError** - 如果 *policy* 不是 `mindspore.dataset.vision.AutoAugmentPolicy` 类型。
- **TypeError** - 如果 *interpolation* 不是 `mindspore.dataset.vision.Inter` 类型。
- **TypeError** - 如果 *fill_value* 不是整数或长度为 3 的元组。
- **RuntimeError** - 如果给定的张量 *shape* 不是 $\langle H, W, C \rangle$ 。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import AutoAugmentPolicy, Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> transforms_list = [vision.AutoAugment(policy=AutoAugmentPolicy.IMAGENET,
...                                     interpolation=Inter.NEAREST,
...                                     fill_value=0)]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.AutoAugment()(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.AutoContrast

class mindspore.dataset.vision.**AutoContrast** (*cutoff=0.0, ignore=None*)

在输入图像上自动调整对比度。首先计算图像的直方图，将直方图中最亮像素的值映射为 255，将直方图中最暗像素的值映射为 0。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数:

- **cutoff** (float, 可选) - 输入图像直方图中，需要剔除的最亮和最暗像素的百分比。该值必须在 [0.0, 50.0) 范围内。默认值: 0.0。
- **ignore** (Union[int, sequence], 可选) - 要忽略的背景像素值，忽略值必须在 [0, 255] 范围内。默认值: None。

异常:

- **TypeError** - 如果 *cutoff* 不是 float 类型。

- **TypeError** - 如果 *ignore* 不是 int 或 sequence 类型。
- **ValueError** - 如果 *cutoff* 不在 [0.0, 50.0) 范围内。
- **ValueError** - 如果 *ignore* 不在 [0, 255] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AutoContrast(cutoff=10.0, ignore=[10, 20])]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.AutoContrast(cutoff=10.0, ignore=[10, 20])(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据支持 *uint8* 或者 *float32* 类型，输入数据的通道仅支持 1 和 3。如果数据类型是 *float32*，期望输入的值的范围为 [0, 1]。输入数据的高度限制范围为 [4, 8192]，宽度限制范围为 [6, 4096]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU" , "Ascend"]。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.AutoContrast(cutoff=10.0, ignore=[10, 20]).device("Ascend
↪")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↪columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↪numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.AutoContrast(cutoff=10.0, ignore=[10, 20]).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

`mindspore.dataset.vision.BoundingBoxAugment`

class `mindspore.dataset.vision.BoundingBoxAugment` (*transform*, *ratio=0.3*)

对图像的随机标注边界框区域，应用给定的图像变换处理。

参数：

- **transform** (`TensorOperation`) - 对图像的随机标注边界框区域应用的变换处理。
- **ratio** (`float`, 可选) - 要应用变换的边界框的比例。范围：[0.0, 1.0]。默认值：0.3。

异常：

- **TypeError** - 如果 *transform* 不是 `mindspore.dataset.vision` 模块中的图像变换处理。
- **TypeError** - 如果 *ratio* 不是 `float` 类型。
- **ValueError** - 如果 *ratio* 不在 [0.0, 1.0] 范围内。
- **RuntimeError** - 如果给定的边界框无效。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(np.
↪float32))

```

(下页继续)

(续上页)

```

>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[func],
...                                                    input_columns=["image"],
...                                                    output_columns=["image", "bbox"])
>>> # set bounding box operation with ratio of 1 to apply rotation on all bounding boxes
>>> bbox_aug_op = vision.BoundingBoxAugment(vision.RandomRotation(90), 1)
>>> # map to apply ops
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[bbox_aug_op],
...                                                    input_columns=["image", "bbox"],
...                                                    output_columns=["image", "bbox"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["bbox"].shape, item["bbox"].dtype)
...     break
(100, 100, 3) float32
(1, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((3, 4))
>>> data = data.astype(np.float32)
>>> func = lambda img, bboxes: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).
    ↳astype(bboxes.dtype))
>>> func_data, func_bboxes = func(data, data)
>>> output = vision.BoundingBoxAugment(transforms.Fill(100), 1.0)(func_data, func_bboxes)
>>> print(output[0].shape, output[0].dtype)
(3, 4) float32
>>> print(output[1].shape, output[1].dtype)
(1, 4) float32

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.CenterCrop

class mindspore.dataset.vision.CenterCrop(size)

对输入图像应用中心区域裁剪。如果输入图像尺寸小于输出尺寸，则在裁剪前对输入图像边界填充 0 像素。

参数：

- **size** (Union[int, sequence]) - 裁剪区域尺寸大小。如果 size 是整数，则返回一个裁剪尺寸大小为 (size, size) 的正方形。如果 size 是一个长度为 2 的序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。值必须大于 0。

异常：

- **TypeError** - 如果 size 不是 int 或 sequence 类型。
- **ValueError** - 如果 size 小于或等于 0。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <..., H, W, C>。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>>
>>> # crop image to a square
>>> transforms_list1 = [vision.CenterCrop(50)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list1, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(50, 50, 3) uint8
>>>
>>> # crop image to portrait style
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list2 = [vision.CenterCrop((60, 40))]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list2, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(60, 40, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.CenterCrop(1)(data)
>>> print(output.shape, output.dtype)
(1, 1, 3) uint8

```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.ConvertColor

class mindspore.dataset.vision.**ConvertColor** (*convert_mode*)

更改图像的色彩空间。

支持 Ascend 硬件加速, 需要通过 `.device("Ascend")` 方式开启。

参数:

- **convert_mode** (*ConvertMode*) - 图像色彩空间转换的模式。
 - **ConvertMode.COLOR_BGR2BGRA**: 将 BGR 图像转换为 BGRA 图像。
 - **ConvertMode.COLOR_RGB2RGBA**: 将 RGB 图像转换为 RGBA 图像。
 - **ConvertMode.COLOR_BGRA2BGR**: 将 BGRA 图像转换为 BGR 图像。
 - **ConvertMode.COLOR_RGBA2RGB**: 将 RGBA 图像转换为 RGB 图像。
 - **ConvertMode.COLOR_BGR2RGBA**: 将 BGR 图像转换为 RGBA 图像。
 - **ConvertMode.COLOR_RGB2BGRA**: 将 RGB 图像转换为 BGRA 图像。

- `ConvertMode.COLOR_RGBA2BGR`: 将 RGBA 图像转换为 BGR 图像。
- `ConvertMode.COLOR_BGRA2RGB`: 将 BGRA 图像转换为 RGB 图像。
- `ConvertMode.COLOR_BGR2RGB`: 将 BGR 图像转换为 RGB 图像。
- `ConvertMode.COLOR_RGB2BGR`: 将 RGB 图像转换为 BGR 图像。
- `ConvertMode.COLOR_BGRA2RGBA`: 将 BGRA 图像转换为 RGBA 图像。
- `ConvertMode.COLOR_RGBA2BGRA`: 将 RGBA 图像转换为 BGRA 图像。
- `ConvertMode.COLOR_BGR2GRAY`: 将 BGR 图像转换为 GRAY 图像。
- `ConvertMode.COLOR_RGB2GRAY`: 将 RGB 图像转换为 GRAY 图像。
- `ConvertMode.COLOR_GRAY2BGR`: 将 GRAY 图像转换为 BGR 图像。
- `ConvertMode.COLOR_GRAY2RGB`: 将 GRAY 图像转换为 RGB 图像。
- `ConvertMode.COLOR_GRAY2BGRA`: 将 GRAY 图像转换为 BGRA 图像。
- `ConvertMode.COLOR_GRAY2RGBA`: 将 GRAY 图像转换为 RGBA 图像。
- `ConvertMode.COLOR_BGRA2GRAY`: 将 BGRA 图像转换为 GRAY 图像。
- `ConvertMode.COLOR_RGBA2GRAY`: 将 RGBA 图像转换为 GRAY 图像。

异常:

- **TypeError** - 如果 `convert_mode` 不是 `mindspore.dataset.vision.ConvertMode` 类型。
- **RuntimeError** - 如果输入图像的 shape 不是 `<H, W>` 或 `<H, W, C>`。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>>
>>> # Convert RGB images to GRAY images
>>> convert_op = vision.ConvertColor(vision.ConvertMode.COLOR_RGB2GRAY)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=convert_op, input_columns=[
↪ "image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100) uint8
>>> # Convert RGB images to BGR images
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> convert_op = vision.ConvertColor(vision.ConvertMode.COLOR_RGB2BGR)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=convert_op, input_columns=[
↪ "image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
```

(下页继续)

(续上页)

```
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.ConvertColor(vision.ConvertMode.COLOR_RGB2GRAY)(data)
>>> print(output.shape, output.dtype)
(100, 100) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据支持 *uint8* 或者 *float32* 类型，数据格式支持 NHWC，Channels: [1, 3, 4]，N 只支持 1。输入数据的高度限制范围为 [4, 8192]，宽度限制范围为 [6, 4096]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU" , "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.ConvertColor(vision.ConvertMode.COLOR_RGB2BGR).device(
    ↪ "Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↪ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
    ↪ numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.ConvertColor(vision.ConvertMode.COLOR_RGB2BGR).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

`mindspore.dataset.vision.Crop`

class `mindspore.dataset.vision.Crop` (*coordinates, size*)

在输入图像上裁剪出指定区域。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **coordinates** (sequence) - 裁剪区域的起始左上角坐标。必须是两个值的序列，形式为 (上, 左)。
- **size** (Union[int, sequence]) - 裁剪区域的尺寸大小。
 - 如果 *size* 是整数，则返回一个裁剪尺寸大小为 (*size*, *size*) 的正方形。
 - 如果 *size* 是一个长度为 2 的序列，则返回一个裁剪尺寸大小为 (高度, 宽度) 的长方形。

值必须大于 0。

异常：

- **TypeError** - 如果 *coordinates* 不是 sequence 类型。
- **TypeError** - 如果 *size* 不是 int 或 sequence 类型。
- **ValueError** - 如果 *coordinates* 小于 0。
- **ValueError** - 如果 *size* 小于或等于 0。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> crop_op = vision.Crop((0, 0), 32)
>>> transforms_list = [crop_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(32, 32, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.Crop((0, 0), 1)(data)
>>> print(output.shape, output.dtype)
(1, 1, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时, 输入/输出数据的维度限制在 [4, 6] 和 [32768, 32768] 范围内。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> crop_op = vision.Crop((0, 0), (100, 75)).device("Ascend")
>>> transforms_list = [crop_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
->numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 75, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Crop((0, 0), 64).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(64, 64, 3) uint8
```

mindspore.dataset.vision.CutMixBatch

class mindspore.dataset.vision.**CutMixBatch** (*image_batch_format*, *alpha*=1.0, *prob*=1.0)

对输入批次的图像和标注应用剪切混合转换。请注意, 在调用此操作符之前, 您需要将标注制作为 one-hot 格式并进行批处理。

参数:

- **image_batch_format** (*ImageBatchFormat*) - 图像批处理输出格式。可以是 *ImageBatchFormat.NHWC* 或 *ImageBatchFormat.NCHW*。
- **alpha** (float, 可选) - β 分布的超参数, 必须大于 0。默认值: 1.0。
- **prob** (float, 可选) - 对每个图像应用剪切混合处理的概率, 取值范围: [0.0, 1.0]。默认值: 1.0。

异常:

- **TypeError** - 如果 `image_batch_format` 不是 `mindspore.dataset.vision.ImageBatchFormat` 的类型。
- **TypeError** - 如果 `alpha` 不是 float 类型。
- **TypeError** - 如果 `prob` 不是 float 类型。
- **ValueError** - 如果 `alpha` 小于或等于 0。
- **ValueError** - 如果 `prob` 不在 [0.0, 1.0] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.transforms as transforms
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import ImageBatchFormat
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(28, 28, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=lambda img: (data, np.random.randint(0, 5, (3, 1))),
...     input_columns=["image"],
...     output_columns=["image", "label"])
>>> onehot_op = transforms.OneHot(num_classes=10)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=onehot_op, input_columns=[
↪ "label"])
>>> cutmix_batch_op = vision.CutMixBatch(ImageBatchFormat.NHWC, 1.0, 0.5)
>>> numpy_slices_dataset = numpy_slices_dataset.batch(5)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=cutmix_batch_op,
...     input_columns=["image", "label"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["label"].shape, item["label"].dtype)
...     break
(5, 28, 28, 3) uint8
(5, 3, 10) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, (3, 3, 10, 10)).astype(np.uint8)
>>> label = np.array([[0, 1], [1, 0], [1, 0]])
>>> output = vision.CutMixBatch(vision.ImageBatchFormat.NCHW, 1.0, 1.0)(data, label)
>>> print(output[0].shape, output[0].dtype)
(3, 3, 10, 10) uint8
>>> print(output[1].shape, output[1].dtype)
(3, 2) float32
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.CutOut

class mindspore.dataset.vision.CutOut (length, num_patches=1, is_hwc=True)

从输入图像数组中，随机裁剪出给定数量的正方形区域。

参数：

- **length** (int) - 每个正方形区域的边长，必须大于 0。
- **num_patches** (int, 可选) - 要从图像中切出的正方形区域数，必须大于 0。默认值：1。
- **is_hwc** (bool, 可选) - 表示输入图像是否为 HWC 格式，True 为 HWC 格式，False 为 CHW 格式。默认值：True。

异常：

- **TypeError** - 如果 *length* 不是 int 类型。
- **TypeError** - 如果 *num_patches* 不是 int 类型。
- **TypeError** - 如果 *is_hwc* 不是 bool 类型。
- **ValueError** - 如果 *length* 小于或等于 0。
- **ValueError** - 如果 *num_patches* 小于或等于 0。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.CutOut(80, num_patches=10)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.CutOut(20)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.Decode`

class `mindspore.dataset.vision.Decode` (*to_pil=False*)

将输入的压缩图像解码为 RGB 格式。当前支持的图片类型：JPEG、BMP、PNG、TIFF、GIF（需要指定 *to_pil=True*）、WEBP（需要指定 *to_pil=True*）。

支持 Ascend 硬件加速，需要通过 *.device("Ascend")* 方式开启。

参数：

- **to_pil** (bool, 可选) - 是否将图像解码为 PIL 数据类型。若为 `True`，图像将被解码为 PIL 数据类型，否则解码为 NumPy 数据类型。默认值：`False`。

异常：

- **RuntimeError** - 如果输入图像不是一维序列。
- **RuntimeError** - 如果输入数据不是合法的图像字节数据。
- **RuntimeError** - 如果输入数据已经是解码的图像数据。

支持平台：

CPU Ascend

样例：

```
>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
...     def __init__(self):
...         self.data = []
...         img = Image.new("RGB", (300, 300), (255, 255, 255))
...         draw = ImageDraw.Draw(img)
...         draw.ellipse((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
...width=5)
...         img.save("./1.jpg")
...         data = np.fromfile("./1.jpg", np.uint8)
...         self.data.append(data)
...
...     def __getitem__(self, index):
...         return self.data[0]
...
...     def __len__(self):
...         return 5
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> transforms_list = [vision.Decode(), vision.RandomHorizontalFlip()]
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=[
... "image"])
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(300, 300, 3) uint8
>>> os.remove("./1.jpg")
```

(下页继续)

(续上页)

```

>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255, 0))
>>> img.save("./2.jpg")
>>> data = np.fromfile("./2.jpg", np.uint8)
>>> output = vision.Decode()(data)
>>> print(output.shape, output.dtype)
(300, 300, 3) uint8
>>> os.remove("./2.jpg")

```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据仅支持 *uint8* 类型。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```

>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
...     def __init__(self):
...         self.data = []
...         img = Image.new("RGB", (300, 300), (255, 255, 255))
...         draw = ImageDraw.Draw(img)
...         draw.ellipse(((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
...width=5)
...         img.save("./1.jpg")
...         data = np.fromfile("./1.jpg", np.uint8)
...         self.data.append(data)
...
...     def __getitem__(self, index):
...         return self.data[0]

```

(下页继续)

(续上页)

```

...
...     def __len__(self):
...         return 5
...
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> decode_op = vision.Decode().device("Ascend")
>>> resize_op = vision.Resize([100, 75], Inter.BICUBIC)
>>> transforms_list = [decode_op, resize_op]
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=[
↪ "image"])
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 75, 3) uint8
>>> os.remove("./1.jpg")
>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255,
↪ 0))
>>> img.save("./2.jpg")
>>> data = np.fromfile("./2.jpg", np.uint8)
>>> output = vision.Decode().device("Ascend")(data)
>>> print(output.shape, output.dtype)
(300, 300, 3) uint8
>>> os.remove("./2.jpg")

```

mindspore.dataset.vision.DecodeVideo

class mindspore.dataset.vision.DecodeVideo

将输入的视频原始字节解码为视频、音频。

支持的视频格式有 AVI、H264、H265、MOV、MP4 和 WMV。

异常：

- **RuntimeError** - 如果输入不是一维序列。
- **RuntimeError** - 如果输入数据的数据类型不是 uint8。
- **RuntimeError** - 如果输入数据为空。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> # Custom class to generate and read video dataset
>>> class VideoDataset:

```

(下页继续)

(续上页)

```

...     def __init__(self, file_list):
...         self.file_list = file_list
...
...     def __getitem__(self, index):
...         filename = self.file_list[index]
...         return np.fromfile(filename, np.uint8)
...
...     def __len__(self):
...         return len(self.file_list)
>>>
>>> dataset = ds.GeneratorDataset(VideoDataset(["/path/to/video/file"]), ["data"])
>>> decode_video = vision.DecodeVideo()
>>> dataset = dataset.map(operations=[decode_video], input_columns=["data"], output_columns=[
    ↪ "video", "audio"])
>>>
>>> # Use the transform in eager mode
>>> filename = "/path/to/video/file"
>>> raw_ndarray = np.fromfile(filename, np.uint8)
>>> mindspore_output = vision.DecodeVideo()(raw_ndarray)

```

mindspore.dataset.vision.Equalize

class mindspore.dataset.vision.Equalize

对输入图像进行直方图均衡化。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

异常：

- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Equalize()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↪ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
    ↪ 3))
>>> output = vision.Equalize()(data)

```

(下页继续)

(续上页)

```
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据仅支持 *uint8* 类型，输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192]，宽度限制范围为 [6, 4096]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Equalize().device("Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Equalize().device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

mindspore.dataset.vision.Erase

class mindspore.dataset.vision.**Erase** (*top, left, height, width, value=0.0, inplace=False*)

擦除输入图像中指定的区域，并填充指定像素值 *value*。

支持 Ascend 硬件加速，需要通过 *.device("Ascend")* 方式开启。

参数：

- **top** (int) - 擦除区域左上角位置的纵坐标。
- **left** (int) - 擦除区域左上角位置的横坐标。
- **height** (int) - 擦除区域的高度。
- **width** (int) - 擦除区域的宽度。
- **value** (Union[float, Sequence[float, float, float]], 可选) - 擦除区域的像素填充值。默认值：0.0。
 - 若输入 float，将以该值填充 RGB 通道。
 - 若输入 Sequence[float, float, float]，将分别用于填充 R、G、B 通道。
- **inplace** (bool, 可选) - 是否直接在原图上执行擦除。默认值：False。

异常：

- **TypeError** - 如果 *top* 不是 int 类型。
- **ValueError** - 如果 *top* 为负数。
- **TypeError** - 如果 *left* 不是 int 类型。
- **ValueError** - 如果 *left* 为负数。
- **TypeError** - 如果 *height* 不是 int 类型。
- **ValueError** - 如果 *height* 非正数。
- **TypeError** - 如果 *width* 不是 int 类型。
- **ValueError** - 如果 *width* 非正数。
- **TypeError** - 如果 *value* 不是 float 或 Sequence[float, float, float] 类型。
- **ValueError** - 如果 *value* 中元素的值不在 [0, 255] 范围。
- **TypeError** - 如果 *inplace* 不是 bool 类型。
- **RuntimeError** - 如果输入图像的形状不是 <H, W, C>。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Erase(10, 10, 10, 10)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↪ columns=["image"])
```

(下页继续)

(续上页)

```
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Erase(10, 10, 10, 10)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据支持 *uint8* 或者 *float32* 类型，输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192]，宽度限制范围为 [6, 4096]。不支持 *inplace* 参数。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Erase(10, 10, 10, 10, (100, 100, 100)).device("Ascend")]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳ numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Erase(10, 10, 10, 10, (100, 100, 100)).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

mindspore.dataset.vision.FiveCrop**class** mindspore.dataset.vision.**FiveCrop** (*size*)

在输入 PIL 图像的中心与四个角处，分别裁剪指定尺寸大小的子图。

参数：

- **size** (Union[int, Sequence[int, int]]) - 裁剪子图的尺寸大小。若输入 int，则以该值为边长裁剪 (*size*, *size*) 尺寸大小的子图；若输入 Sequence[int, int]，则以 2 个元素分别为高和宽裁剪子图。

异常：

- **TypeError** - 当 *size* 的类型不为 int 或 Sequence[int, int]。
- **ValueError** - 当 *size* 不为正数。

支持平台：

CPU

样例：

```
>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
...     def __init__(self):
...         self.data = []
...         img = Image.new("RGB", (300, 300), (255, 255, 255))
...         draw = ImageDraw.Draw(img)
...         draw.ellipse(((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
... width=5)
...         img.save("./1.jpg")
...         data = np.fromfile("./1.jpg", np.uint8)
...         self.data.append(data)
...
...     def __getitem__(self, index):
...         return self.data[0]
...
...     def __len__(self):
...         return 5
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> transforms_list = Compose([vision.Decode(to_pil=True),
...                             vision.FiveCrop(size=200),
...                             # 4D stack of 5 images
...                             lambda *images: np.stack([vision.ToTensor()(image) for image_
... in images])])
>>> # apply the transform to dataset through map function
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=
... "image")
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
```

(下页继续)

(续上页)

```

(5, 3, 200, 200) float32
>>> os.remove("./1.jpg")
>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255, 0))
>>> img.save("./2.jpg")
>>> data = Image.open("./2.jpg")
>>> output = vision.FiveCrop(size=20)(data)
>>> for cropped_img in output:
...     print(cropped_img.size)
...     break
(20, 20)
>>> os.remove("./2.jpg")

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.GaussianBlur

class mindspore.dataset.vision.**GaussianBlur** (*kernel_size*, *sigma=None*)

使用指定的高斯核对输入图像进行模糊处理。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **kernel_size** (Union[int, Sequence[int, int]]) - 高斯核的大小。需为正奇数。
 - 若输入类型为 int，将同时使用该值作为高斯核的宽、高。
 - 若输入类型为 Sequence[int, int]，将分别使用这两个元素作为高斯核的宽、高。
- **sigma** (Union[float, Sequence[float, float]], 可选) - 高斯核的标准差。需为正数。
 - 若输入类型为 float，将同时使用该值作为高斯核宽、高的标准差。
 - 若输入类型为 Sequence[float, float]，将分别使用这两个元素作为高斯核宽、高的标准差。

默认值：None，将通过公式 $((kernel\_size - 1) * 0.5 - 1) * 0.3 + 0.8$ 计算得到高斯核的标准差。

异常：

- **TypeError** - 如果 *kernel_size* 不是 int 或 Sequence[int] 类型。
- **TypeError** - 如果 *sigma* 不是 float 或 Sequence[float] 类型。
- **ValueError** - 如果 *kernel_size* 不是正数和奇数。
- **ValueError** - 如果 *sigma* 不是正数。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.GaussianBlur(3, 3)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.GaussianBlur(3, 3)(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8

```

教程样例:

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时, 参数 *kernel_size* 仅支持取值 1、3、5。输入数据的维度限制在 [4, 6] 和 [8192, 4096] 范围内。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU" , "Ascend"]。

支持平台:

CPU Ascend

样例:

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> blur_op = vision.GaussianBlur(3, 3).device("Ascend")
>>> transforms_list = [blur_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])

```

(下页继续)

(续上页)

```
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
    ↳numpy=True):
    ...     print(item["image"].shape, item["image"].dtype)
    ...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.GaussianBlur(3, 3).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

mindspore.dataset.vision.Grayscale

class mindspore.dataset.vision.Grayscale(num_output_channels=1)

将输入 PIL 图像转换为灰度图。

参数：

- **num_output_channels** (int) - 输出灰度图的通道数，取值可为 1 或 3。当取值为 3 时，返回图像各通道的像素值将相同。默认值：1。

异常：

- **TypeError** - 当 *num_output_channels* 的类型不为 int。
- **ValueError** - 当 *num_output_channels* 取值不为 1 或 3。

支持平台：

CPU

样例：

```
>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
    ...     def __init__(self):
    ...         self.data = []
    ...         img = Image.new("RGB", (300, 300), (255, 255, 255))
    ...         draw = ImageDraw.Draw(img)
    ...         draw.ellipse(((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
    ↳width=5)
    ...         img.save("./1.jpg")
    ...         data = np.fromfile("./1.jpg", np.uint8)
    ...         self.data.append(data)
    ...
    ...     def __getitem__(self, index):
    ...         return self.data[0]
    ...
    ...     def __len__(self):
```

(下页继续)

(续上页)

```

...         return 5
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> transforms_list = Compose([vision.Decode(to_pil=True),
...                             vision.Grayscale(3),
...                             vision.ToTensor()])
>>> # apply the transform to dataset through map function
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=
↳ "image")
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 300, 300) float32
>>> os.remove("./1.jpg")
>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255, 0))
>>> img.save("./2.jpg")
>>> data = Image.open("./2.jpg")
>>> output = vision.Grayscale(3)(data)
>>> print(np.array(output).shape, np.array(output).dtype)
(300, 300, 3) uint8
>>> os.remove("./2.jpg")

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.HorizontalFlip

class mindspore.dataset.vision.HorizontalFlip

水平翻转输入图像。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

异常：

- **RuntimeError** - 如果输入图像的 shape 不是 `<H, W>` 或 `<..., H, W, C>`。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.HorizontalFlip()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])

```

(下页继续)

(续上页)

```
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2, 6,
↪3))
>>> output = vision.HorizontalFlip()(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据支持 *uint8* 和 *float32* 类型，输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192]，宽度限制范围为 [6, 4096]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> horizontal_flip_op = vision.HorizontalFlip().device("Ascend")
>>> transforms_list = [horizontal_flip_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↪columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↪numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.HorizontalFlip().device("Ascend")(data)
```

(下页继续)

(续上页)

```
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

mindspore.dataset.vision.HsvToRgb

class mindspore.dataset.vision.HsvToRgb(is_hwc=False)

将输入的 HSV 格式 numpy.ndarray 图像转换为 RGB 格式。

参数：

- **is_hwc** (bool) - 若为 True，表示输入图像的 shape 为 <H, W, C> 或 <N, H, W, C>；否则为 <C, H, W> 或 <N, C, H, W>。默认值：False。

异常：

- **TypeError** - 当 is_hwc 的类型不为 bool。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> transforms_list = Compose([vision.CenterCrop(20),
...                             vision.ToTensor(),
...                             vision.HsvToRgb()])
>>> # apply the transform to dataset through map function
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 20, 20) float64
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2,
↳ 3))
>>> output = vision.HsvToRgb(is_hwc=True)(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) float64
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.HWC2CHW`

`class mindspore.dataset.vision.HWC2CHW`

将输入图像的 shape 从 $\langle H, W, C \rangle$ 转换为 $\langle C, H, W \rangle$ 。如果输入图像的 shape 为 $\langle H, W \rangle$ ，图像将保持不变。

说明：此操作默认通过 CPU 执行，也支持异构加速到 GPU 或 Ascend 上执行。

异常：

- **RuntimeError** - 如果输入图像的 shape 不是 $\langle H, W \rangle$ 或 $\langle H, W, C \rangle$ 。

支持平台：

CPU GPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomHorizontalFlip(0.75),
...                    vision.RandomCrop(64),
...                    vision.HWC2CHW()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 64, 64) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2, 6,
↳ 3))
>>> output = vision.HWC2CHW()(data)
>>> print(output.shape, output.dtype)
(3, 2, 2) uint8
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.Invert`

`class mindspore.dataset.vision.Invert`

对输入的 RGB 图像进行色彩反转。

对于图像中的每个像素，若原像素值为 $pixel$ ，则反转后的像素值为 $255 - pixel$ 。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

异常：

- **RuntimeError** - 如果输入图像的 shape 不是 $\langle H, W, C \rangle$ 。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Invert()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5], [0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 2, 6,
↳ 3))
>>> output = vision.Invert()(data)
>>> print(output.shape, output.dtype)
(2, 2, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据仅支持 *uint8* 类型，输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192]，宽度限制范围为 [6, 4096]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
```

(下页继续)

(续上页)

```

>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> invert_op = vision.Invert()
>>> transforms_list = [invert_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Invert().device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

mindspore.dataset.vision.LinearTransformation

class mindspore.dataset.vision.LinearTransformation (transformation_matrix, mean_vector)

使用指定的变换方阵和均值向量，对输入 `numpy.ndarray` 图像进行线性变换。

先将输入图像展平为一维，从中减去均值向量，然后计算其与变换方阵的点积，最后再变形回原始 shape。

参数：

- **transformation_matrix** (`numpy.ndarray`) - shape 为 (D, D) 的变换方阵，其中 $D = C \times H \times W$ 。
- **mean_vector** (`numpy.ndarray`) - shape 为 (D,) 的均值向量，其中 $D = C \times H \times W$ 。

异常：

- **TypeError** - 当 `transformation_matrix` 的类型不为 `numpy.ndarray`。
- **TypeError** - 当 `mean_vector` 的类型不为 `numpy.ndarray`。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> height, width = 32, 32
>>> dim = 3 * height * width
>>> transformation_matrix = np.ones([dim, dim])
>>> mean_vector = np.zeros(dim)
>>> transforms_list = Compose([vision.Resize((height, width)),
...                             vision.ToTensor(),
...                             vision.LinearTransformation(transformation_matrix, mean_
↳vector)])
>>> # apply the transform to dataset through map function

```

(下页继续)

(续上页)

```

>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 32, 32) float64
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randn(10, 10, 3)
>>> transformation_matrix = np.random.randn(300, 300)
>>> mean_vector = np.random.randn(300,)
>>> output = vision.LinearTransformation(transformation_matrix, mean_vector)(data)
>>> print(output.shape, output.dtype)
(10, 10, 3) float64

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.MixUp

class mindspore.dataset.vision.MixUp (batch_size, alpha, is_single=True)

随机混合一批输入的 numpy.ndarray 图像及其标签。

首先将每个图像乘以一个从 Beta 分布随机生成的权重 λ ，然后加上另一个图像与 $1 - \lambda$ 的乘积。使用同样的 λ 值将图像对应的标签进行混合。请确保标签预先进行了 one-hot 编码。

参数：

- **batch_size** (int) - 批处理大小，即图片的数量。
- **alpha** (float) - Beta 分布的 α 参数值， β 参数也将使用该值。
- **is_single** (bool, 可选) - 若为 True，将在批内随机混合图像 [img0, ..., img(n-1), img(n)] 与 [img1, ..., img(n), img0] 及对应标签；否则，将每批图像与前一批图像的处理结果混合。默认值：True。

异常：

- **TypeError** - 当 batch_size 的类型不为 int。
- **TypeError** - 当 alpha 的类型不为 float。
- **TypeError** - 当 is_single 的类型不为 bool。
- **ValueError** - 当 batch_size 不为正数。
- **ValueError** - 当 alpha 不为正数。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> import mindspore.dataset.transforms as transforms

```

(下页继续)

(续上页)

```

>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(64, 64, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=lambda img: (data, np.random.randint(0, 5, (3, 1))),
...     input_columns=["image"],
...     output_columns=["image", "label"])
>>> # on the fly decode the label
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms.OneHot(10), input_
... columns="label")
>>> # batch the samples
>>> numpy_slices_dataset = numpy_slices_dataset.batch(batch_size=4)
>>> # finally mix up the images and labels
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=vision.MixUp(batch_size=1, alpha=0.2),
...     input_columns=["image", "label"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["label"].shape, item["label"].dtype)
...     break
(4, 64, 64, 3) float64
(4, 3, 10) float64
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> label = np.array([[0, 1]])
>>> output = vision.MixUp(batch_size=2, alpha=0.2, is_single=False)(data, label)
>>> print(output[0].shape, output[0].dtype)
(2, 100, 100, 3) float64
>>> print(output[1].shape, output[1].dtype)
(2, 2) float64

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.MixUpBatch

class mindspore.dataset.vision.MixUpBatch (*alpha=1.0*)

对输入批次的图像和标注应用混合转换。从批处理中随机抽取两个图像，其中一个图像乘以随机权重 (*lambda*)，另一个图像乘以 ($1 - \text{lambda}$)，并相加。该处理将会同时应用于 one-hot 标注。

上述的 *lambda* 是根据指定的参数 *alpha* 生成的。计算方式为在 [*alpha*, 1] 范围内随机生成两个系数 *x1*, *x2*，然后 $\text{lambda} = (x1 / (x1 + x2))$ 。

请注意，在调用此处理之前，您需要将标注制作成 one-hot 格式并进行 batch 操作。

参数：

- **alpha** (float, 可选) - β 分布的超参数，该值必须为正。默认值：1.0。

异常：

- **TypeError** - 如果 *alpha* 不是 float 类型。
- **ValueError** - 如果 *alpha* 不是正数。

- **RuntimeError** - 如果输入图像的 shape 不是 <N, H, W, C> 或 <N, C, H, W>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> import mindspore.dataset.transforms as transforms
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(64, 64, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=lambda img: (data, np.random.randint(0, 5, (3, 1))),
...     input_columns=["image"],
...     output_columns=["image", "label"])
>>> onehot_op = transforms.OneHot(num_classes=10)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=onehot_op,
...     input_columns=["label"])
>>> mixup_batch_op = vision.MixUpBatch(alpha=0.9)
>>> numpy_slices_dataset = numpy_slices_dataset.batch(5)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=mixup_batch_op,
...     input_columns=["image", "label"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["label"].shape, item["label"].dtype)
...     break
(5, 64, 64, 3) uint8
(5, 3, 10) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, (2, 10, 10, 3)).astype(np.uint8)
>>> label = np.array([[0, 1], [1, 0]])
>>> output = vision.MixUpBatch(1)(data, label)
>>> print(output[0].shape, output[0].dtype)
(2, 10, 10, 3) uint8
>>> print(output[1].shape, output[1].dtype)
(2, 2) float32
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.Normalize

class mindspore.dataset.vision.**Normalize** (*mean, std, is_hwc=True*)

根据均值和标准差，对输入图像进行归一化。

此处理将使用以下公式对输入图像进行归一化： $\text{output}[\text{channel}] = (\text{input}[\text{channel}] - \text{mean}[\text{channel}]) / \text{std}[\text{channel}]$ ，其中 channel 代表通道索引， $\text{channel} \geq 1$ 。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

说明：此操作默认通过 CPU 执行，也支持异构加速到 GPU 或 Ascend 上执行。

参数：

- **mean** (sequence) - 图像每个通道的均值组成的列表或元组。平均值必须在 [0.0, 255.0] 范围内。
- **std** (sequence) - 图像每个通道的标准差组成的列表或元组。标准差值必须在 (0.0, 255.0] 范围内。
- **is_hwc** (bool, 可选) - 表示输入图像是否为 HWC 格式，True 为 HWC 格式，False 为 CHW 格式。默认值：True。

异常：

- **TypeError** - 如果 *mean* 不是 sequence 类型。
- **TypeError** - 如果 *std* 不是 sequence 类型。
- **TypeError** - 如果 *is_hwc* 不是 bool 类型。
- **ValueError** - 如果 *mean* 不在 [0.0, 255.0] 范围内。
- **ValueError** - 如果 *std* 不在 (0.0, 255.0] 范围内。
- **RuntimeError** - 如果给定的 tensor format 不是 <H, W> 或 <..., H, W, C>。

支持平台：

CPU GPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> normalize_op = vision.Normalize(mean=[121.0, 115.0, 100.0], std=[70.0, 68.0, 71.0], is_
    ↪ hwc=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[normalize_op],
    ...                                             input_columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
    ...     print(item["image"].shape, item["image"].dtype)
    ...     break
(100, 100, 3) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Normalize(mean=[121.0, 115.0, 100.0], std=[70.0, 68.0, 71.0])(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) float32
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

- 当执行设备是 CPU 时, 输入数据支持 `uint8`、`float32` 或者 `float64` 类型, 输入数据的通道支持 1、2 和 3。
- 当执行设备是 Ascend 时, 输入数据支持 `uint8` 或者 `float32` 类型, 输入数据的通道仅支持 1 和 3。输入数据的维度限制在 [4, 6] 和 [8192, 4096] 范围内。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 `device_target` 的类型不为 str。
- **ValueError** - 当 `device_target` 的取值不为 ["CPU", "Ascend"]。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> resize_op = vision.Resize([100, 75], Inter.BICUBIC)
>>> transforms_list = [resize_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> normalize_op = vision.Normalize(mean=[121.0, 115.0, 100.0], std=[70.0, 68.0, 71.0]).
↳ device("Ascend")
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=normalize_op, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳ numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 75, 3) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Normalize(mean=[121.0, 115.0, 100.0], std=[70.0, 68.0, 71.0]).device(
↳ "Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) float32
```

mindspore.dataset.vision.NormalizePad

class mindspore.dataset.vision.NormalizePad(mean, std, dtype='float32', is_hwc=True)

根据均值和标准差对输入图像进行归一化，然后填充一个全零的额外通道。

参数：

- **mean** (sequence) - 图像每个通道的均值组成的列表或元组。平均值必须在 (0.0, 255.0] 范围内。
- **std** (sequence) - 图像每个通道的标准差组成的列表或元组。标准差值必须在 (0.0, 255.0] 范围内。
- **dtype** (str, 可选) - 输出图像的数据类型。默认值: "float32"。
- **is_hwc** (bool, 可选) - 指定输入图像的格式，若为 True，表示输入为 HWC 格式，否则为 CHW 格式。默认值: True。

异常：

- **TypeError** - 如果 *mean* 不是 sequence 类型。
- **TypeError** - 如果 *std* 不是 sequence 类型。
- **TypeError** - 如果 *dtype* 不是 str 类型。
- **TypeError** - 如果 *is_hwc* 不是 bool 类型。
- **ValueError** - 如果 *mean* 不在 [0.0, 255.0] 范围内。
- **ValueError** - 如果 *std* 不在范围内 (0.0, 255.0]。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W>, <H, W, C> 或 <C, H, W>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> normalize_pad_op = vision.NormalizePad(mean=[121.0, 115.0, 100.0],
...                                     std=[70.0, 68.0, 71.0],
...                                     dtype="float32")
>>> transforms_list = [normalize_pad_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.NormalizePad(mean=[121.0, 115.0, 100.0], std=[70.0, 68.0, 71.0], dtype=
↳ "float32")(data)
>>> print(output.shape, output.dtype)
(100, 100, 4) float32
```

教程样例：

- [视觉变换样例库](#)

mindspore.dataset.vision.Pad

class mindspore.dataset.vision.**Pad** (*padding*, *fill_value=0*, *padding_mode=Border.CONSTANT*)
填充图像。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **padding** (Union[int, Sequence[int, int], Sequence[int, int, int, int]]) - 图像各边填充的像素数。
 - 如果 *padding* 是一个整数，代表为图像的所有方向填充该值大小的像素。
 - 如果 *padding* 是一个包含 2 个值的元组或列表，第一个值会用于填充图像的左侧和右侧，第二个值会用于填充图像的上侧和下侧。
 - 如果 *padding* 是一个包含 4 个值的元组或列表，则分别用于填充图像的左侧、上侧、右侧和下侧。

填充值必须为非负值。

- **fill_value** (Union[int, tuple[int]], 可选) - 填充的像素值，仅在 *padding_mode* 取值为 `Border.CONSTANT` 时有效。
 - 如果是 3 元素元组，则分别用于填充 R、G、B 通道。
 - 如果是整数，则用于所有 RGB 通道。

fill_value 值必须在 [0, 255] 范围内。默认值：0。

- **padding_mode** (*Border*, 可选) - 边界填充方式。可以是 `Border.CONSTANT`、`Border.EDGE`、`Border.REFLECT`、`Border.SYMMETRIC`。默认值：`Border.CONSTANT`。
 - **Border.CONSTANT** - 使用常量值进行填充。
 - **Border.EDGE** - 使用各边的边界像素值进行填充。
 - **Border.REFLECT** - 以各边的边界为轴进行镜像填充，忽略边界像素值。
 - **Border.SYMMETRIC** - 以各边的边界为轴进行对称填充，包括边界像素值。

异常：

- **TypeError** - 如果 *padding* 不是 int、Sequence[int, int] 或 Sequence[int, int, int, int] 类型。
- **TypeError** - 如果 *fill_value* 不是 int 或 tuple[int] 类型。
- **TypeError** - 如果 *padding_mode* 不是 `mindspore.dataset.vision.Border` 的类型。
- **ValueError** - 如果 *padding* 为负数。
- **ValueError** - 如果 *fill_value* 不在 [0, 255] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Pad([100, 100, 100, 100])]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(300, 300, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Pad([100, 100, 100, 100])(data)
>>> print(output.shape, output.dtype)
(300, 300, 3) uint8

```

教程样例:

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时, 输入/输出数据的维度限制在 [4, 6] 和 [32768, 32768] 范围内。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台:

CPU Ascend

样例:

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> pad_op = vision.Pad([100, 100, 100, 100]).device("Ascend")
>>> transforms_list = [pad_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳ numpy=True):

```

(下页继续)

```

...     print(item["image"].shape, item["image"].dtype)
...     break
(300, 300, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Pad([100, 100, 100, 100]).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(300, 300, 3) uint8

```

mindspore.dataset.vision.PadToSize

class mindspore.dataset.vision.**PadToSize** (*size, offset=None, fill_value=0, padding_mode=Border.CONSTANT*)

将图像填充到固定大小。

参数：

- **size** (Union[int, Sequence[int, int]]) - 要填充的目标大小。若输入整型，则将图像填充为 (size, size) 大小；如果提供了序列 [int, int]，则将图像填充为 (高度, 宽度) 大小。
- **offset** (Union[int, Sequence[int, int]], 可选) - 顶部和左侧要填充的长度。如果输入整型，使用此值填充图像上侧和左侧。如果提供了序列 [int, int]，则应按 [top, left] 的顺序排列，填充图像上侧和左侧。默认值：None，表示对称填充，保持原始图像处于中心位置。
- **fill_value** (Union[int, tuple[int, int, int]], 可选) - 填充的像素值，仅在 *padding_mode* 取值为 `Border.CONSTANT` 时有效。如果是 3 元素元组，则分别用于填充 R、G、B 通道。如果是整数，则用于所有 RGB 通道。*fill_value* 值必须在 [0, 255] 范围内。默认值：0。
- **padding_mode** (*Border*, 可选) - 边界填充方式。可以是 `Border.CONSTANT`、`Border.EDGE`、`Border.REFLECT`、`Border.SYMMETRIC`。默认值：`Border.CONSTANT`。
 - **Border.CONSTANT** - 使用常量值进行填充。
 - **Border.EDGE** - 使用各边的边界像素值进行填充。
 - **Border.REFLECT** - 以各边的边界为轴进行镜像填充，忽略边界像素值。
 - **Border.SYMMETRIC** - 以各边的边界为轴进行对称填充，包括边界像素值。

异常：

- **TypeError** - 如果 *size* 不是 int 或 tuple[int, int] 类型。
- **TypeError** - 如果 *offset* 不是 int 或 tuple[int, int] 类型。
- **TypeError** - 如果 *fill_value* 不是 int 或 tuple[int] 类型。
- **TypeError** - 如果 *padding_mode* 不是 `mindspore.dataset.vision.Border` 的类型。
- **ValueError** - 如果 *size* 不是正数。
- **ValueError** - 如果 *offset* 为负数。
- **ValueError** - 如果 *fill_value* 不在 [0, 255] 的范围内。
- **RuntimeError** - 如果输入图像的形状不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：


```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.PadToSize([256, 256])]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(256, 256, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.PadToSize([256, 256])(data)
>>> print(output.shape, output.dtype)
(256, 256, 3) uint8

```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.Perspective`

class `mindspore.dataset.vision.Perspective` (*start_points, end_points, interpolation=Inter.BILINEAR*)

对输入图像进行透视变换。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **start_points** (`Sequence[Sequence[int, int]]`) - 起始点坐标序列，包含四个两元素子序列，分别对应原图中四边形的 [左上、右上、右下、左下]。
- **end_points** (`Sequence[Sequence[int, int]]`) - 目标点坐标序列，包含四个两元素子序列，分别对应目标图中四边形的 [左上、右上、右下、左下]。
- **interpolation** (`Inter`, 可选) - 图像插值方法。可选值详见 `mindspore.dataset.vision.Inter`。默认值：`Inter.BILINEAR`。
 - **Ascend 模式**：通过 `.device("Ascend")` 设定执行设备为 Ascend 时，只支持 `Inter.LINEAR`、`Inter.NEAREST` 差值方法。

异常：

- **TypeError** - 如果 `start_points` 不是 `Sequence[Sequence[int, int]]` 类型。
- **TypeError** - 如果 `end_points` 不是 `Sequence[Sequence[int, int]]` 类型。
- **TypeError** - 当 `interpolation` 的类型不为 `mindspore.dataset.vision.Inter`。
- **RuntimeError** - 如果输入图像的形状不是 `<H, W>` 或 `<H, W, C>`。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> start_points = [[0, 63], [63, 63], [63, 0], [0, 0]]
>>> end_points = [[0, 32], [32, 32], [32, 0], [0, 0]]
>>> transforms_list = [vision.Perspective(start_points, end_points, Inter.BILINEAR)]
>>> # apply the transform to dataset through map function
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> start_points = [[0, 63], [63, 63], [63, 0], [0, 0]]
>>> end_points = [[0, 32], [32, 32], [32, 0], [0, 0]]
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Perspective(start_points, end_points, Inter.BILINEAR)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据支持 *uint8* 和 *float32* 类型，输入数据的通道仅支持 1/3。输入数据的高度限制范围为 [6, 8192]、宽度限制范围为 [10, 4096]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode

```

(下页继续)

(续上页)

```

>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> start_points = [[0, 63], [63, 63], [63, 0], [0, 0]]
>>> end_points = [[0, 32], [32, 32], [32, 0], [0, 0]]
>>> perspective_op = vision.Perspective(start_points, end_points).device("Ascend")
>>> transforms_list = [perspective_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> start_points = [[0, 63], [63, 63], [63, 0], [0, 0]]
>>> end_points = [[0, 32], [32, 32], [32, 0], [0, 0]]
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Perspective(start_points, end_points, Inter.BILINEAR).device("Ascend
↳") (data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

mindspore.dataset.vision.Posterize

class mindspore.dataset.vision.Posterize(*bits*)

减少图像的颜色通道的比特位数，使图像变得高对比度和颜色鲜艳，类似于海报或印刷品的效果。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **bits** (int) - 每个颜色通道保留的位数，取值需在 [0, 8] 范围内。

异常：

- **TypeError** - 如果 *bits* 不是 int 类型。
- **ValueError** - 如果 *bits* 不在 [0, 8] 范围内。
- **RuntimeError** - 如果输入图像的形状不是 <H, W> 或 <H, W, C>。

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Posterize(4)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break

```

(下页继续)

(续上页)

```
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Posterize(4)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例:

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时, 输入数据仅支持 *uint8* 类型, 输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192], 宽度限制范围为 [6, 4096]。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU" , "Ascend"]。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> posterize_op = vision.Posterize(4).device("Ascend")
>>> transforms_list = [posterize_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↪ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
    ↪ numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Posterize(4).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

`mindspore.dataset.vision.RandAugment`

class `mindspore.dataset.vision.RandAugment` (*num_ops=2, magnitude=9, num_magnitude_bins=31, interpolation=Inter.NEAREST, fill_value=0*)

对输入图像应用 `RandAugment` 数据增强方法。

参考论文 [RandAugment: Learning Augmentation Strategies from Data](#)。

只支持 3 通道 RGB 图像。

参数：

- **num_ops** (int, 可选) - 顺序执行的数据增强变换个数。默认值：2。
- **magnitude** (int, 可选) - 所有变换的幅值，需小于 `num_magnitude_bins`。默认值：9。
- **num_magnitude_bins** (int, 可选) - 不同变换幅值的个数，需不小于 2。默认值：31。
- **interpolation** (`Inter`, 可选) - 图像插值方法。可选值详见 `mindspore.dataset.vision.Inter`。默认值：`Inter.NEAREST`。
- **fill_value** (Union[int, tuple[int, int, int]], 可选) - 变换后超出原图外区域的像素填充值，取值需在 [0, 255] 范围内。默认值：0。如果输入 int，将用于填充所有 RGB 通道。如果输入 tuple[int, int, int]，则分别用于填充 R、G、B 通道。

异常：

- **TypeError** - 如果 `num_ops` 不是 int 类型。
- **ValueError** - 如果 `num_ops` 为负数。
- **TypeError** - 如果 `magnitude` 不是 int 类型。
- **ValueError** - 如果 `magnitude` 非正数。
- **TypeError** - 如果 `num_magnitude_bins` 不是 int 类型。
- **ValueError** - 如果 `num_magnitude_bins` 小于 2。
- **TypeError** - 如果 `interpolation` 不是 `mindspore.dataset.vision.Inter` 类型。
- **TypeError** - 如果 `fill_value` 不是 int 或 tuple[int, int, int] 类型。
- **ValueError** - 如果 `fill_value` 取值不在 [0, 255] 范围。
- **RuntimeError** - 如果输入图像的形状不是 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandAugment()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
```

(下页继续)

(续上页)

```

...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandAugment(interpolation=Inter.BILINEAR, fill_value=255)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomAdjustSharpness

class mindspore.dataset.vision.RandomAdjustSharpness (*degree, prob=0.5*)

以给定的概率，随机调整输入图像的锐度。

参数：

- **degree** (float) - 锐度调整度，必须是非负的。0.0 度表示模糊图像，1.0 度表示原始图像，2.0 度表示清晰度增加 2 倍。
- **prob** (float, 可选) - 图像被锐化的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 如果 *degree* 的类型不为 float。
- **TypeError** - 如果 *prob* 的类型不为 float。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围。
- **ValueError** - 如果 *degree* 为负数。
- **RuntimeError** - 如果给定的张量形状不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomAdjustSharpness(2.0, 0.5)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>

```

(下页继续)

(续上页)

```
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomAdjustSharpness(2.0, 1.0)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.RandomAffine`

class `mindspore.dataset.vision.RandomAffine` (*degrees*, *translate=None*, *scale=None*, *shear=None*, *resample=Inter.NEAREST*, *fill_value=0*)

对输入图像应用随机仿射变换。

参数：

- **degrees** (`Union[int, float, sequence]`) - 旋转度数的范围。如果 *degrees* 是一个数字，它代表旋转范围是 $(-\text{degrees}, \text{degrees})$ 。如果 *degrees* 是一个序列，它代表旋转是 $(\min, \max)$ 。
- **translate** (`sequence`, 可选) - 一个序列 $(\text{tx\_min}, \text{tx\_max}, \text{ty\_min}, \text{ty\_max})$ 用于表示水平 (tx) 方向和垂直 (ty) 方向的最小/最大平移范围，取值范围 $[-1.0, 1.0]$ 。默认值：None，不对图像进行任何平移。水平和垂直偏移分别从以下范围中随机选择： $(\text{tx\_min} * \text{width}, \text{tx\_max} * \text{width})$ 和 $(\text{ty\_min} * \text{height}, \text{ty\_max} * \text{height})$ 。
 - 如果 *translate* 是一个包含 2 个值的元组或列表，则 $(\text{translate}[0], \text{translate}[1])$ 表示水平 (X) 方向的随机平移范围。
 - 如果 *translate* 是一个包含 4 个值的元组或列表，则 $(\text{translate}[0], \text{translate}[1])$ 表示水平 (X) 方向的随机平移范围， $(\text{translate}[2], \text{translate}[3])$ 表示垂直 (Y) 方向的随机平移范围。
- **scale** (`sequence`, 可选) - 图像的比例因子的随机范围，必须为非负数，使用原始比例。默认值：None。
- **shear** (`Union[float, Sequence[float, float], Sequence[float, float, float, float]]`, 可选) - 图像的剪切因子的随机范围，必须为正数。默认值：None，不应用任何剪切。
 - 如果是数字，则应用在 $(-\text{shear}, +\text{shear})$ 范围内平行于 X 轴的剪切。
 - 如果 *shear* 是一个包含 2 个值的元组或列表，则在 $(\text{shear}[0], \text{shear}[1])$ 范围内进行水平 (X) 方向的剪切变换。
 - 如果 *shear* 是一个包含 4 个值的元组或列表，则在 $(\text{shear}[0], \text{shear}[1])$ 范围内进行水平 (X) 方向的剪切变换，并在 $(\text{shear}[2], \text{shear}[3])$ 范围内进行垂直 (Y) 方向的剪切变换。
- **resample** (`Inter`, 可选) - 图像插值方法。可选值详见 `mindspore.dataset.vision.Inter`。默认值：`Inter.NEAREST`。
- **fill_value** (`Union[int, tuple[int]]`, 可选) - 用于填充输出图像中变换之外的区域。元组中必须有三个值，取值范围是 $[0, 255]$ 。默认值：0。

异常：

- **TypeError** - 如果 *degrees* 不是 `int`、`float` 或 `sequence` 类型。
- **TypeError** - 如果 *translate* 不是 `sequence` 类型。
- **TypeError** - 如果 *scale* 不是 `sequence` 类型。
- **TypeError** - 如果 *shear* 不是 `int`、`float` 或 `sequence` 类型。
- **TypeError** - 如果 *resample* 不是 `mindspore.dataset.vision.Inter` 的类型。

- **TypeError** - 如果 *fill_value* 不是 `int` 或 `tuple[int]` 类型。
- **ValueError** - 如果 *degrees* 为负数。
- **ValueError** - 如果 *translate* 不在范围 `[-1.0, 1.0]` 内。
- **ValueError** - 如果 *scale* 为负数。
- **ValueError** - 如果 *shear* 不是正数。
- **RuntimeError** - 如果输入图像的 *shape* 不是 `<H, W>` 或 `<H, W, C>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> random_affine_op = vision.RandomAffine(degrees=15,
...                                       translate=(-0.1, 0.1, 0, 0),
...                                       scale=(0.9, 1.1),
...                                       resample=Inter.NEAREST)
>>> transforms_list = [random_affine_op]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↪columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomAffine(degrees=15, translate=(-0.1, 0.1, 0, 0),
...                             scale=(0.9, 1.1), resample=Inter.NEAREST)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomAutoContrast

class mindspore.dataset.vision.RandomAutoContrast (*cutoff=0.0, ignore=None, prob=0.5*)

以给定的概率，自动调整图像的对比度。

参数：

- **cutoff** (float, 可选) - 输入图像直方图中，需要剔除的最亮和最暗像素的百分比。该值必须在 `[0.0, 50.0)` 范围内。默认值：`0.0`。
- **ignore** (Union[int, sequence], 可选) - 要忽略的背景像素值，该值必须在 `[0, 255]` 范围内。默认值：`None`。

- **prob** (float, 可选) - 图像被调整对比度的概率, 取值范围: [0.0, 1.0]。默认值: 0.5。

异常:

- **TypeError** - 如果 *cutoff* 不是 float 类型。
- **TypeError** - 如果 *ignore* 不是 int 或 sequence 类型。
- **TypeError** - 如果 *prob* 的类型不为 float。
- **ValueError** - 如果 *cutoff* 不在 [0, 50.0) 范围内。
- **ValueError** - 如果 *ignore* 不在 [0, 255] 范围内。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomAutoContrast(cutoff=0.0, ignore=None, prob=0.5)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomAutoContrast(cutoff=0.0, ignore=None, prob=1.0)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.RandomColor

class mindspore.dataset.vision.RandomColor (*degrees=(0.1, 1.9)*)

随机调整输入图像的颜色。此操作仅适用于 3 通道 RGB 图像。

参数:

- **degrees** (Sequence[float], 可选) - 色彩调节系数的范围, 必须为非负数。它应该是 (min, max) 格式。如果 min 与 max 相等, 则代表色彩变化步长固定。默认值: (0.1, 1.9)。

异常:

- **TypeError** - 如果 *degrees* 不是 Sequence[float] 类型。

- **ValueError** - 如果 *degrees* 为负数。
- **RuntimeError** - 如果输入图像的 *shape* 不是 $\langle H, W, C \rangle$ 。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomColor((0.5, 2.0))]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomColor((0.1, 1.9))(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomColorAdjust

class mindspore.dataset.vision.RandomColorAdjust (*brightness=(1, 1), contrast=(1, 1), saturation=(1, 1), hue=(0, 0)*)

随机调整输入图像的亮度、对比度、饱和度和色调。

说明：此操作默认通过 CPU 执行，也支持异构加速到 GPU 或 Ascend 上执行。

参数：

- **brightness** (Union[float, Sequence[float]], 可选) - 亮度调整因子。不能为负。默认值：(1, 1)。如果是浮点数，则从 [max(0, 1-brightness), 1+brightness] 范围内统一选择因子。如果它是一个序列，则代表是范围 [min, max]，从此范围中选择调整因子。
- **contrast** (Union[float, Sequence[float]], 可选) - 对比度调整因子。不能为负。默认值：(1, 1)。如果是浮点数，则从 [max(0, 1-contrast), 1+contrast] 范围内统一选择因子。如果它是一个序列，则代表是范围 [min, max]，从此范围中选择调整因子。
- **saturation** (Union[float, Sequence[float]], 可选) - 饱和度调整因子。不能为负。默认值：(1, 1)。如果是浮点数，则从 [max(0, 1-saturation), 1+saturation] 范围内统一选择因子。如果它是一个序列，则代表是范围 [min, max]，从此范围中选择调整因子。

- **hue** (Union[float, Sequence[float]], 可选) - 色调调整因子。默认值: (0, 0)。如果是浮点数, 则代表是范围 [-hue, hue], 从此范围中选择调整因子。注意 *hue* 取值应为 [0, 0.5]。如果它是一个序列, 则代表是范围 [min, max], 从此范围中选择调整因子。注意取值范围 min 和 max 是 [-0.5, 0.5] 范围内的浮点数, 并且 min 小于等于 max。

异常:

- **TypeError** - 如果 *brightness* 不是 float 或 Sequence[float] 类型。
- **TypeError** - 如果 *contrast* 不是 float 或 Sequence[float] 类型。
- **TypeError** - 如果 *saturation* 不是 float 或 Sequence[float] 类型。
- **TypeError** - 如果 *hue* 不是 float 或 Sequence[float] 类型。
- **ValueError** - 如果 *brightness* 为负数。
- **ValueError** - 如果 *contrast* 为负数。
- **ValueError** - 如果 *saturation* 为负数。
- **ValueError** - 如果 *hue* 不在 [-0.5, 0.5] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W, C>。

支持平台:

CPU GPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transform_op = vision.RandomColorAdjust(brightness=(0.5, 1),
...                                       contrast=(0.4, 1),
...                                       saturation=(0.3, 1))
>>> transforms_list = [transform_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomColorAdjust(brightness=(0.5, 1), contrast=(0.4, 1), saturation=(0.
->3, 1))(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.RandomCrop

class mindspore.dataset.vision.RandomCrop (*size, padding=None, pad_if_needed=False, fill_value=0, padding_mode=Border.CONSTANT*)

对输入图像进行随机区域的裁剪。如果输入图像尺寸小于输出尺寸，输入图像将在裁剪前被填充。

说明：如果在多个数据列上应用此处理，则需要确保每个数据列图像的 `shape` 相同。

参数：

- **size** (Union[int, Sequence[int]]) - 裁剪图像的输出尺寸大小。值必须为正。
 - 如果 `size` 是整数，则返回一个裁剪尺寸大小为 (`size, size`) 的正方形。
 - 如果 `size` 是一个长度为 2 的序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。
- **padding** (Union[int, Sequence[int]], 可选) - 图像各边填充的像素数。填充值必须为非负值。默认值：None。
 - 如果 `padding` 不为 None，则首先使用 `padding` 填充图像。
 - 如果 `padding` 是一个整数，代表为图像的所有方向填充该值大小的像素。
 - 如果 `padding` 是一个包含 2 个值的元组或列表，第一个值会用于填充图像的左侧和右侧，第二个值会用于填充图像的上侧和下侧。
 - 如果 `padding` 是一个包含 4 个值的元组或列表，则分别填充图像的左侧、上侧、右侧和下侧。
- **pad_if_needed** (bool, 可选) - 如果输入图像高度或者宽度小于 `size` 指定的输出图像尺寸大小，是否进行填充。默认值：False。
- **fill_value** (Union[int, tuple[int]], 可选) - 边框的像素强度，仅当 `padding_mode` 为 `Border.CONSTANT` 时有效。`fill_value` 值必须在 [0, 255] 范围内。默认值：0。
 - 如果是 3 元素元组，则分别用于填充 R、G、B 通道。
 - 如果是整数，则用于所有 RGB 通道。
- **padding_mode** (`Border`, 可选) - 边界填充方式。它可以是 `Border.CONSTANT`、`Border.EDGE`、`Border.REFLECT`、`Border.SYMMETRIC`。默认值：`Border.CONSTANT`。
 - **Border.CONSTANT** - 使用常量值进行填充。
 - **Border.EDGE** - 使用各边的边界像素值进行填充。
 - **Border.REFLECT** - 以各边的边界为轴进行镜像填充，忽略边界像素值。
 - **Border.SYMMETRIC** - 以各边的边界为轴进行对称填充，包括边界像素值。

异常：

- **TypeError** - 如果 `size` 不是 int 或 `Sequence[int]` 类型。
- **TypeError** - 如果 `padding` 不是 int 或 `Sequence[int]` 类型。
- **TypeError** - 如果 `pad_if_needed` 不是 bool 类型。
- **TypeError** - 如果 `fill_value` 不是 int 或 `tuple[int]` 类型。
- **TypeError** - 如果 `padding_mode` 不是 `mindspore.dataset.vision.Border` 的类型。
- **ValueError** - 如果 `size` 不是正数。
- **ValueError** - 如果 `padding` 为负数。
- **ValueError** - 如果 `fill_value` 不在 [0, 255] 范围内。

- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <..., H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Border
>>>
>>> # Use the transform in dataset pipeline mode
>>> random_crop_op = vision.RandomCrop(64, [16, 16, 16, 16], padding_mode=Border.EDGE)
>>> transforms_list = [random_crop_op]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(64, 64, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomCrop(8, [10, 10, 10, 10], padding_mode=Border.EDGE)(data)
>>> print(output.shape, output.dtype)
(8, 8, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomCropDecodeResize

class mindspore.dataset.vision.RandomCropDecodeResize (*size*, *scale*=(0.08, 1.0), *ratio*=(3. / 4., 4. / 3.), *interpolation*=Inter.BILINEAR, *max_attempts*=10)

“裁剪”、“解码”和“调整尺寸大小”的组合处理。该操作将在随机位置裁剪输入图像，以 RGB 模式对裁剪后的图像进行解码，并调整解码图像的尺寸大小。针对 JPEG 图像进行了优化，可以获得更好的性能。

参数：

- **size** (Union[int, Sequence[int]]) - 调整后图像的输出尺寸大小。大小值必须为正。如果 **size** 是整数，则返回一个裁剪尺寸大小为 (size, size) 的正方形。如果 **size** 是一个长度为 2 的序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。
- **scale** (Union[list, tuple], 可选) - 要裁剪的原始尺寸大小的各个尺寸的范围 [min, max)，必须为非负数。默认值：(0.08, 1.0)。
- **ratio** (Union[list, tuple], 可选) - 宽高比的范围 [min, max) 裁剪，必须为非负数。默认值：(3. / 4., 4. / 3.)。
- **interpolation** (Inter, 可选) - 图像插值方法。可选值详见 `mindspore.dataset.vision.Inter`。默认值：Inter.BILINEAR。
- **max_attempts** (int, 可选) - 生成随机裁剪位置的最大尝试次数，超过该次数时将使用中心裁剪，*max_attempts* 值必须为正数。默认值：10。

异常:

- **TypeError** - 如果 *size* 不是 int 或 Sequence[int] 类型。
- **TypeError** - 如果 *scale* 不是 tuple 或 list 类型。
- **TypeError** - 如果 *ratio* 不是 tuple 或 list 类型。
- **TypeError** - 如果 *interpolation* 不是 `mindspore.dataset.vision.Inter` 的类型。
- **TypeError** - 如果 *max_attempts* 不是 int 类型。
- **ValueError** - 如果 *size* 不是正数。
- **ValueError** - 如果 *scale* 为负数。
- **ValueError** - 如果 *ratio* 为负数。
- **ValueError** - 如果 *max_attempts* 不是正数。
- **RuntimeError** - 如果输入图像不是一维序列。

支持平台:

CPU

样例:

```
>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
...     def __init__(self):
...         self.data = []
...         img = Image.new("RGB", (300, 300), (255, 255, 255))
...         draw = ImageDraw.Draw(img)
...         draw.ellipse(((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
... width=5)
...         img.save("./1.jpg")
...         data = np.fromfile("./1.jpg", np.uint8)
...         self.data.append(data)
...
...     def __getitem__(self, index):
...         return self.data[0]
...
...     def __len__(self):
...         return 5
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> resize_crop_decode_op = vision.RandomCropDecodeResize(size=(50, 75),
... scale=(0.25, 0.5),
... interpolation=Inter.NEAREST,
... max_attempts=5)
>>> transforms_list = [resize_crop_decode_op]
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=[
... "image"])
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
```

(下页继续)

(续上页)

```

...     print(item["image"].shape, item["image"].dtype)
...     break
(50, 75, 3) uint8
>>> os.remove("./1.jpg")
>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255, 0))
>>> img.save("./2.jpg")
>>> data = np.fromfile("./2.jpg", np.uint8)
>>> output = vision.RandomCropDecodeResize(size=(50, 75), scale=(0, 10.0), ratio=(0.5, 0.5),
...                                     interpolation=Inter.BILINEAR, max_
-> attempts=1)(data)
>>> print(np.array(output).shape, np.array(output).dtype)
(50, 75, 3) uint8
>>> os.remove("./2.jpg")

```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.RandomCropWithBBBox`

class `mindspore.dataset.vision.RandomCropWithBBBox` (*size*, *padding=None*, *pad_if_needed=False*, *fill_value=0*, *padding_mode=Border.CONSTANT*)

在输入图像的随机位置进行裁剪，并相应地调整边界框。

参数：

- **size** (`Union[int, Sequence[int]]`) - 裁剪图像的输出尺寸大小。大小值必须为正。如果 **size** 是整数，则返回一个裁剪尺寸大小为 (**size**, **size**) 的正方形。如果 **size** 是一个长度为 2 的序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。
- **padding** (`Union[int, Sequence[int]]`, 可选) - 填充图像的像素数。填充值必须非负值。默认值：None。如果 **padding** 不为 None，则首先使用 **padding** 填充图像。如果 **padding** 是一个整数，代表为图像的所有方向填充该值大小的像素。如果 **padding** 是一个包含 2 个值的元组或列表，第一个值会用于填充图像的左侧和右侧，第二个值会用于填充图像的上侧和下侧。如果 **padding** 是一个包含 4 个值的元组或列表，则分别填充图像的左侧、上侧、右侧和下侧。
- **pad_if_needed** (`bool`, 可选) - 如果输入图像高度或者宽度小于 **size** 指定的输出图像尺寸大小，是否进行填充。默认值：False。
- **fill_value** (`Union[int, tuple[int]]`, 可选) - 边框的像素强度，仅当 **padding_mode** 为 `Border.CONSTANT` 时有效。如果是 3 元素元组，则分别用于填充 R、G、B 通道。如果是整数，则用于所有 RGB 通道。**fill_value** 值必须在 [0, 255] 范围内。默认值：0。
- **padding_mode** (`Border`, 可选) - 边界填充方式。它可以是 `Border.CONSTANT`、`Border.EDGE`、`Border.REFLECT`、`Border.SYMMETRIC`。默认值：`Border.CONSTANT`。
 - **Border.CONSTANT** - 使用常量值进行填充。
 - **Border.EDGE** - 使用各边的边界像素值进行填充。
 - **Border.REFLECT** - 以各边的边界为轴进行镜像填充，忽略边界像素值。
 - **Border.SYMMETRIC** - 以各边的边界为轴进行对称填充，包括边界像素值。

异常：

- **TypeError** - 如果 *size* 不是 int 或 Sequence[int] 类型。
- **TypeError** - 如果 *padding* 不是 int 或 Sequence[int] 类型。
- **TypeError** - 如果 *pad_if_needed* 不是 bool 类型。
- **TypeError** - 如果 *fill_value* 不是 int 或 tuple[int] 类型。
- **TypeError** - 如果 *padding_mode* 不是 `mindspore.dataset.vision.Border` 的类型。
- **ValueError** - 如果 *size* 不是正数。
- **ValueError** - 如果 *padding* 为负数。
- **ValueError** - 如果 *fill_value* 不在 [0, 255] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(np.
    ↳float32))
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[func],
    ...                                           input_columns=["image"],
    ...                                           output_columns=["image", "bbox"])
>>> random_crop_with_bbox_op = vision.RandomCropWithBBBox([64, 64], [20, 20, 20, 20])
>>> transforms_list = [random_crop_with_bbox_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↳columns=["image", "bbox"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
    ...     print(item["image"].shape, item["image"].dtype)
    ...     print(item["bbox"].shape, item["bbox"].dtype)
    ...     break
(64, 64, 3) float32
(1, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(data.
    ↳dtype))
>>> func_data, func_bboxes = func(data)
>>> output = vision.RandomCropWithBBBox([64, 64], [20, 20, 20, 20])(func_data, func_bboxes)
>>> print(output[0].shape, output[0].dtype)
(64, 64, 3) float32
>>> print(output[1].shape, output[1].dtype)
(1, 4) float32
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomEqualize

class mindspore.dataset.vision.**RandomEqualize** (*prob=0.5*)

以给定的概率，随机对输入图像进行直方图均衡化。

参数：

- **prob** (float, 可选) - 图像被均衡化的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 如果 *prob* 的类型不为 float。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomEqualize(0.5)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomEqualize(1.0)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomErasing

class mindspore.dataset.vision.**RandomErasing** (*prob=0.5, scale=(0.02, 0.33), ratio=(0.3, 3.3), value=0, inplace=False, max_attempts=10*)

以给定的概率，擦除输入 `numpy.ndarray` 图像上随机矩形区域内的像素。

请参阅论文 [Random Erasing Data Augmentation](#)。

参数：

- **prob** (float, 可选) - 执行随机擦除的概率，取值范围：[0.0, 1.0]。默认值：0.5。

- **scale** (Sequence[float, float], 可选) - 擦除区域面积相对原图比例的随机选取范围, 按照 (min, max) 顺序排列。默认值: (0.02, 0.33)。
- **ratio** (Sequence[float, float], 可选) - 擦除区域宽高比的随机选取范围, 按照 (min, max) 顺序排列。默认值: (0.3, 3.3)。
- **value** (Union[int, str, Sequence[int, int, int]]) - 擦除区域的像素填充值。若输入 int, 将以该值填充 RGB 通道; 若输入 Sequence[int, int, int], 将分别用于填充 R、G、B 通道; 若输入字符串 'random', 将以从标准正态分布获得的随机值擦除各个像素。默认值: 0。
- **inplace** (bool, 可选) - 是否直接在原图上执行擦除。默认值: False。
- **max_attempts** (int, 可选) - 生成随机擦除区域的最大尝试次数, 超过该次数时将返回原始图像。默认值: 10。

异常:

- **TypeError** - 当 *prob* 的类型不为 float。
- **TypeError** - 当 *scale* 的类型不为 Sequence[float, float]。
- **TypeError** - 当 *ratio* 的类型不为 Sequence[float, float]。
- **TypeError** - 当 *value* 的类型不为 int、str 或 Sequence[int, int, int]。
- **TypeError** - 当 *inplace* 的类型不为 bool。
- **TypeError** - 当 *max_attempts* 的类型不为 int。
- **ValueError** - 当 *prob* 取值不在 [0.0, 1.0] 范围内。
- **ValueError** - 当 *scale* 为负数。
- **ValueError** - 当 *ratio* 为负数。
- **ValueError** - 当 *value* 取值不在 [0, 255] 范围内。
- **ValueError** - 当 *max_attempts* 不为正数。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> transforms_list = Compose([vision.ToTensor(),
...                             vision.RandomErasing(value='random')])
>>> # apply the transform to dataset through map function
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
```

(下页继续)

(续上页)

```
>>> data = np.random.randint(254, 255, size=(3, 100, 100)).astype(np.uint8)
>>> output = vision.RandomErasing(prob=1.0, max_attempts=1)(data)
>>> print(output.shape, output.dtype)
(3, 100, 100) uint8
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.RandomGrayscale`

class `mindspore.dataset.vision.RandomGrayscale` (*prob=0.1*)

以给定的概率，将输入 PIL 图像转换为灰度图。

参数：

- **prob** (float, 可选) - 执行灰度转换的概率，取值范围：[0.0, 1.0]。默认值：0.1。

异常：

- **TypeError** - 当 *prob* 的类型不为 float。
- **ValueError** - 当 *prob* 取值不在 [0.0, 1.0] 范围内。

支持平台：

CPU

样例：

```
>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
...     def __init__(self):
...         self.data = []
...         img = Image.new("RGB", (300, 300), (255, 255, 255))
...         draw = ImageDraw.Draw(img)
...         draw.ellipse(((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
... width=5)
...         img.save("./1.jpg")
...         data = np.fromfile("./1.jpg", np.uint8)
...         self.data.append(data)
...
...     def __getitem__(self, index):
...         return self.data[0]
...
...     def __len__(self):
...         return 5
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> transforms_list = Compose([vision.Decode(to_pil=True),
```

(下页继续)

(续上页)

```

...             vision.RandomGrayscale(0.3),
...             vision.ToTensor())])
>>> # apply the transform to dataset through map function
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=
↳ "image")
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 300, 300) float32
>>> os.remove("./1.jpg")
>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255, 0))
>>> img.save("./2.jpg")
>>> data = Image.open("./2.jpg")
>>> output = vision.RandomGrayscale(1.0)(data)
>>> print(np.array(output).shape, np.array(output).dtype)
(300, 300, 3) uint8
>>> os.remove("./2.jpg")

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomHorizontalFlip

class mindspore.dataset.vision.RandomHorizontalFlip (*prob=0.5*)

以给定的概率，对输入图像进行水平随机翻转。

参数：

- **prob** (float, 可选) - 图像被翻转的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 如果 *prob* 不是 float 类型。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomHorizontalFlip(0.75)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])

```

(下页继续)

(续上页)

```
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomHorizontalFlip(1.0)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomHorizontalFlipWithBBBox

class mindspore.dataset.vision.RandomHorizontalFlipWithBBBox (prob=0.5)

以给定的概率，对输入图像及其边界框进行随机水平翻转。

参数：

- **prob** (float, 可选) - 图像被翻转的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 如果 *prob* 不是 float 类型。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(np.
    ↳float32))
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[func],
...     input_columns=["image"],
...     output_columns=["image", "bbox"])
>>> transforms_list = [vision.RandomHorizontalFlipWithBBBox(0.70)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list,
...     input_columns=["image", "bbox"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["bbox"].shape, item["bbox"].dtype)
...     break
(100, 100, 3) float32
```

(下页继续)

(续上页)

```

(1, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(data.
    ↳dtype))
>>> func_data, func_bboxes = func(data)
>>> output = vision.RandomHorizontalFlipWithBBox(1)(func_data, func_bboxes)
>>> print(output[0].shape, output[0].dtype)
(100, 100, 3) float32
>>> print(output[1].shape, output[1].dtype)
(1, 4) float32

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomInvert

class mindspore.dataset.vision.RandomInvert (*prob=0.5*)

以给定的概率，随机反转图像的颜色。

参数：

- **prob** (float, 可选) - 图像被反转颜色的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 如果 *prob* 的类型不为 float。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W, C>。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomInvert(0.5)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomInvert(1.0)(data)

```

(下页继续)

(续上页)

```
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomLighting

class mindspore.dataset.vision.**RandomLighting** (*alpha=0.05*)

将 AlexNet PCA 的噪声添加到图像中。Alexnet PCA 噪声的特征值和特征向量是由 ImageNet 数据集计算得出。

参数：

- **alpha** (float, 可选) - 图像的强度，必须是非负的。默认值：0.05。

异常：

- **TypeError** - 如果 *alpha* 的类型不为 float。
- **ValueError** - 如果 *alpha* 为负数。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomLighting(0.1)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomLighting(0.1)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomPerspective

class mindspore.dataset.vision.**RandomPerspective** (*distortion_scale=0.5, prob=0.5, interpolation=Inter.BICUBIC*)

以给定的概率，对输入 PIL 图像进行透视变换。

参数：

- **distortion_scale** (float, 可选) - 失真程度，取值范围为 [0.0, 1.0]。默认值：0.5。
- **prob** (float, 可选) - 执行透视变换的概率，取值范围：[0.0, 1.0]。默认值：0.5。
- **interpolation** (*Inter*, 可选) - 图像插值方法。可选值详见 *mindspore.dataset.vision.Inter*。默认值：Inter.BICUBIC。

异常：

- **TypeError** - 当 *distortion_scale* 的类型不为 float。
- **TypeError** - 当 *prob* 的类型不为 float。
- **TypeError** - 当 *interpolation* 的类型不为 *mindspore.dataset.vision.Inter*。
- **ValueError** - 当 *distortion_scale* 取值不在 [0.0, 1.0] 范围内。
- **ValueError** - 当 *prob* 取值不在 [0.0, 1.0] 范围内。

支持平台：

CPU

样例：

```
>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
...     def __init__(self):
...         self.data = []
...         img = Image.new("RGB", (300, 300), (255, 255, 255))
...         draw = ImageDraw.Draw(img)
...         draw.ellipse((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
... width=5)
...         img.save("./1.jpg")
...         data = np.fromfile("./1.jpg", np.uint8)
...         self.data.append(data)
...
...     def __getitem__(self, index):
...         return self.data[0]
...
...     def __len__(self):
...         return 5
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> transforms_list = Compose([vision.Decode(to_pil=True),
...                             vision.RandomPerspective(prob=0.1),
```

(下页继续)

(续上页)

```

... vision.ToTensor())
>>> # apply the transform to dataset through map function
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=
->"image")
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 300, 300) float32
>>> os.remove("./1.jpg")
>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255, 0))
>>> img.save("./2.jpg")
>>> data = Image.open("./2.jpg")
>>> output = vision.RandomPerspective(prob=1.0)(data)
>>> print(np.array(output).shape, np.array(output).dtype)
(300, 300, 3) uint8
>>> os.remove("./2.jpg")

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomPosterize

class mindspore.dataset.vision.RandomPosterize (bits=(8, 8))

随机减少图像的颜色通道的比特位数，使图像变得高对比度和颜色鲜艳。

参数：

- **bits** (Union[int, Sequence[int]], 可选) - 随机位数压缩的范围。位值必须在 [1,8] 范围内，并且在给定范围内至少包含一个整数值。它必须是 (min, max) 或整数格式。如果 min 与 max 相等，那么它是一个单一的位数压缩操作。默认值：(8, 8)。

异常：

- **TypeError** - 如果 bits 不是 int 或 Sequence[int] 类型。
- **ValueError** - 如果 bits 不在 [1, 8] 范围内。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomPosterize((6, 8))]

```

(下页继续)

(续上页)

```
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomPosterize(1)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomResizedCrop

class mindspore.dataset.vision.RandomResizedCrop (*size, scale=(0.08, 1.0), ratio=(3. / 4., 4. / 3.), interpolation=Inter.BILINEAR, max_attempts=10*)

对输入图像进行随机裁剪，并使用指定的`mindspore.dataset.vision.Inter` 插值方式去调整为指定的尺寸大小。

说明：如果输入图像不止一张，需要保证输入的多张图像尺寸大小一致。

参数：

- **size** (Union[int, Sequence[int]]) - 图像的输出尺寸大小。若输入整型，则放缩至 (size, size) 大小；若输入 2 元素序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。
- **scale** (Union[list, tuple], 可选) - 裁剪子图的尺寸大小相对原图比例的随机选取范围，需要在 [min, max) 区间。默认值：(0.08, 1.0)。
- **ratio** (Union[list, tuple], 可选) - 裁剪子图的宽高比的随机选取范围，需要在 [min, max) 区间。默认值：(3. / 4., 4. / 3.)。
- **interpolation** (*Inter*, 可选) - 图像插值方法。可选值详见`mindspore.dataset.vision.Inter`。默认值：`Inter.BILINEAR`。
- **max_attempts** (int, 可选) - 生成随机裁剪位置的最大尝试次数，超过该次数时将使用中心裁剪。默认值：10。

异常：

- **TypeError** - 当 *size* 的类型不为 int 或 Sequence[int]。
- **TypeError** - 当 *scale* 的类型不为 tuple 或 list。
- **TypeError** - 当 *ratio* 的类型不为 tuple 或 list。
- **TypeError** - 当 *interpolation* 的类型不为`mindspore.dataset.vision.Inter`。
- **TypeError** - 当 *max_attempts* 的类型不为 int。
- **ValueError** - 当 *size* 不为正数。
- **ValueError** - 当 *scale* 为负数。

- **ValueError** - 当 *ratio* 为负数。
- **ValueError** - 当 *max_attempts* 不为正数。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> resize_crop_op = vision.RandomResizedCrop(size=(50, 75), scale=(0.25, 0.5),
...                                           interpolation=Inter.BILINEAR)
>>> transforms_list = [resize_crop_op]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(50, 75, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomResizedCrop(size=(50, 75), scale=(0.25, 0.5), interpolation=Inter.
↳ BILINEAR)(data)
>>> print(output.shape, output.dtype)
(50, 75, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomResizedCropWithBBox

```
class mindspore.dataset.vision.RandomResizedCropWithBBox (size, scale=(0.08, 1.0), ratio=(3. / 4., 4. / 3.),
                                                         interpolation=Inter.BILINEAR,
                                                         max_attempts=10)
```

对输入图像进行随机裁剪且随机调整纵横比，并将处理后的图像调整为指定的尺寸大小，并相应地调整边界框。

参数：

- **size** (Union[int, Sequence[int]]) - 图像的输出尺寸大小。若输入整型，则放缩至 (size, size) 大小；若输入 2 元素序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。
- **scale** (Union[list, tuple], 可选) - 裁剪子图的尺寸大小相对原图比例的随机选取范围，需要在 [min, max) 区间。默认值：(0.08, 1.0)。
- **ratio** (Union[list, tuple], 可选) - 裁剪子图的宽高比的随机选取范围，需要在 [min, max) 区间。默认值：(3. / 4., 4. / 3.)。
- **interpolation** (*Inter*, 可选) - 图像插值方法。可选值详见 *mindspore.dataset.vision.Inter*。默认值：Inter.BILINEAR。

- **max_attempts** (int, 可选) - 生成随机裁剪位置的最大尝试次数, 超过该次数时将使用中心裁剪。默认值: 10。

异常:

- **TypeError** - 当 *size* 的类型不为 int 或 Sequence[int]。
- **TypeError** - 当 *scale* 的类型不为 tuple 或 list。
- **TypeError** - 当 *ratio* 的类型不为 tuple 或 list。
- **TypeError** - 当 *interpolation* 的类型不为 `mindspore.dataset.vision.Inter`。
- **TypeError** - 当 *max_attempts* 的类型不为 int。
- **ValueError** - 当 *size* 不为正数。
- **ValueError** - 当 *scale* 为负数。
- **ValueError** - 当 *ratio* 为负数。
- **ValueError** - 当 *max_attempts* 不为正数。
- **RuntimeError** 当输入图像的 shape 不为 <H, W> 或 <H, W, C>。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(np.
↳float32))
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[func],
...                                              input_columns=["image"],
...                                              output_columns=["image", "bbox"])
>>> bbox_op = vision.RandomResizedCropWithBBBox(size=50, interpolation=Inter.NEAREST)
>>> transforms_list = [bbox_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list,
...                                              input_columns=["image", "bbox"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["bbox"].shape, item["bbox"].dtype)
...     break
(50, 50, 3) float32
(1, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(data.
↳dtype))
>>> func_data, func_bboxes = func(data)
>>> output = vision.RandomResizedCropWithBBBox((16, 64), (0.5, 0.5), (0.5, 0.5))(func_data,
↳func_bboxes)
>>> print(output[0].shape, output[0].dtype)
```

(下页继续)

(续上页)

```
(16, 64, 3) float32
>>> print(output[1].shape, output[1].dtype)
(1, 4) float32
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomResize

class mindspore.dataset.vision.**RandomResize** (*size*)

使用随机选择的`mindspore.dataset.vision.Inter`插值方式，调整输入图像的尺寸大小。

参数：

- **size** (Union[int, Sequence[int]]) - 调整后图像的输出尺寸大小。值必须为正。若输入整型，则放缩至 (size, size) 大小；若输入 2 元素序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。

异常：

- **TypeError** - 如果 *size* 不是 int 或 Sequence[int] 类型。
- **ValueError** - 如果 *size* 不是正数。
- **RuntimeError** - 如果输入图像的 shape 不是 <H, W> 或 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> # 1) randomly resize image, keeping aspect ratio
>>> transforms_list1 = [vision.RandomResize(50)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list1, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(50, 50, 3) uint8
>>> # 2) randomly resize image to landscape style
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list2 = [vision.RandomResize((40, 60))]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list2, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(40, 60, 3) uint8
>>>
>>> # Use the transform in eager mode
```

(下页继续)

(续上页)

```
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomResize(10)(data)
>>> print(output.shape, output.dtype)
(10, 10, 3) uint8
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.RandomResizeWithBBox`

class `mindspore.dataset.vision.RandomResizeWithBBox` (*size*)

对输入图像使用随机选择的`mindspore.dataset.vision.Inter`插值方式去调整它的尺寸大小，并相应地调整边界框的尺寸大小。

参数：

- **size** (`Union[int, Sequence[int]]`) - 调整后图像的输出尺寸大小。若输入整型，则放缩至 (`size, size`) 大小；若输入 2 元素序列，则以 2 个元素分别为高和宽放缩至 (高度, 宽度) 大小。

异常：

- **TypeError** - 当 *size* 的类型不为 `int` 或 `Sequence[int]` 类型。
- **ValueError** - 当 *size* 不为正数。
- **RuntimeError** - 当输入图像的 `shape` 不为 `<H, W>` 或 `<H, W, C>`。

支持平台：

CPU

样例：

```
>>> import copy
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(np.
↳ float32))
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[func],
...                                              input_columns=["image"],
...                                              output_columns=["image", "bbox"])
>>> numpy_slices_dataset2 = copy.deepcopy(numpy_slices_dataset)
>>>
>>> # 1) randomly resize image with bounding boxes, keeping aspect ratio
>>> transforms_list1 = [vision.RandomResizeWithBBox(60)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list1,
...                                              input_columns=["image", "bbox"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["bbox"].shape, item["bbox"].dtype)
...     break
(60, 60, 3) float32
```

(下页继续)

(续上页)

```

(1, 4) float32
>>>
>>> # 2) randomly resize image with bounding boxes to portrait style
>>> transforms_list2 = [vision.RandomResizeWithBBox((80, 60))]
>>> numpy_slices_dataset2 = numpy_slices_dataset2.map(operations=transforms_list2,
...                                                    input_columns=["image", "bbox"])
>>> for item in numpy_slices_dataset2.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["bbox"].shape, item["bbox"].dtype)
...     break
(80, 60, 3) float32
(1, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(data.
... dtype))
>>> func_data, func_bboxes = func(data)
>>> output = vision.RandomResizeWithBBox(64)(func_data, func_bboxes)
>>> print(output[0].shape, output[0].dtype)
(64, 64, 3) float32
>>> print(output[1].shape, output[1].dtype)
(1, 4) float32

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomRotation

class mindspore.dataset.vision.RandomRotation (*degrees, resample=Inter.NEAREST, expand=False, center=None, fill_value=0*)

在指定的角度范围内，随机旋转输入图像。

参数：

- **degrees** (Union[int, float, sequence]) - 旋转角度的随机选取范围。若输入单个数字，则从 (-degrees, degrees) 中随机生成旋转角度；若输入 2 元素序列，需按 (min, max) 顺序排列。
- **resample** (*Inter*, 可选) - 图像插值方法。可选值详见 *mindspore.dataset.vision.Inter*。默认值：*Inter.NEAREST*。
- **expand** (bool, 可选) - 若为 True，将扩展图像尺寸大小使其足以容纳整个旋转图像；若为 False，则保持图像尺寸大小不变。请注意，扩展时将假设图像为中心旋转，且未进行平移。默认值：False。
- **center** (tuple, 可选) - 可选的旋转中心，以图像左上角为原点，旋转中心的位置按照 (宽度, 高度) 格式指定。默认值：None，表示中心旋转。
- **fill_value** (Union[int, tuple[int]], 可选) - 旋转图像之外区域的像素填充值。若输入 3 元素元组，将分别用于填充 R、G、B 通道；若输入整型，将以该值填充 RGB 通道。*fill_value* 值必须在 [0, 255] 范围内。默认值：0。

异常：

- **TypeError** - 当 *degrees* 的类型不为 int、float 或 sequence。
- **TypeError** - 当 *resample* 的类型不为 *mindspore.dataset.vision.Inter*。
- **TypeError** - 当 *expand* 的类型不为 bool。

- **TypeError** - 当 *center* 的类型不为 `tuple`。
- **TypeError** - 当 *fill_value* 的类型不为 `int` 或 `tuple[int]`。
- **ValueError** - 当 *fill_value* 取值不在 `[0, 255]` 范围内。
- **RuntimeError** - 当输入图像的 `shape` 不为 `<H, W>` 或 `<H, W, C>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> seed = ds.config.get_seed()
>>> ds.config.set_seed(12345)
>>> transforms_list = [vision.RandomRotation(degrees=5.0, resample=Inter.NEAREST,
    ↳ expand=True)]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(107, 107, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomRotation(degrees=90, resample=Inter.NEAREST, expand=True)(data)
>>> print(output.shape, output.dtype)
(119, 119, 3) uint8
>>> ds.config.set_seed(seed)
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.RandomSelectSubpolicy`

class `mindspore.dataset.vision.RandomSelectSubpolicy` (*policy*)

从策略列表中，随机选择一个子策略以应用于输入图像。

参数：

- **policy** (`list[list[tuple[TensorOperation, float]]]`) - 可供选择的子策略列表。子策略是一系列 (`operation, prob`) 格式的元组组成的列表，其中 *operation* 是数据处理操作，*prob* 是应用此操作的概率，*prob* 值必须在 `[0.0, 1.0]` 范围内。一旦选择了子策略，子策略中的每个操作都将根据其概率依次应用。

异常：

- **TypeError** - 当 *policy* 包含无效数据处理操作。

支持平台：

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> policy = [(vision.RandomRotation((45, 45)), 0.5),
...           (vision.RandomVerticalFlip(), 1),
...           (vision.RandomColorAdjust(), 0.8)],
...           [(vision.RandomRotation((90, 90)), 1),
...            (vision.RandomColorAdjust(), 0.2)]]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=vision.
↳ RandomSelectSubpolicy(policy),
...                                           input_columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> policy = [(vision.RandomRotation((90, 90)), 1), (vision.RandomColorAdjust(), 1)]
>>> output = vision.RandomSelectSubpolicy(policy)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.RandomSharpness

class mindspore.dataset.vision.**RandomSharpness** (*degrees=(0.1, 1.9)*)

在固定或随机的范围，调整输入图像的锐度。度数为 0.0 时，将返回模糊图像；度数为 1.0 时，将返回原始图像；度数为 2.0 时，将返回锐化图像。

参数:

- **degrees** (Union[list, tuple], 可选) - 锐度调节系数的随机选取范围，需为非负数，按照 (min, max) 顺序排列。如果 min 与 max 相等，将使用固定的调节系数进行处理。默认值: (0.1, 1.9)。

异常:

- **TypeError** - 如果 *degrees* 的类型不为 list 或 tuple。
- **ValueError** - 如果 *degrees* 为负数。
- **ValueError** - 如果 *degrees* 采用 (max, min) 格式而不是 (min, max)。

支持平台:

CPU

样例:

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomSharpness(degrees=(0.2, 1.9))]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomSharpness(degrees=(0, 0.6))(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomSolarize

class mindspore.dataset.vision.RandomSolarize (threshold=(0, 255))

从给定阈值范围内随机选择一个子范围，对位于给定子范围内的像素，将其像素值设置为 (255 - 原本像素值)。

参数：

- **threshold** (tuple, 可选) - 随机反转的阈值范围。默认值：(0, 255)。threshold 输入格式应该为 (min, max)，其中 min 和 max 是 [0, 255] 范围内的整数，并且 min <= max，那么属于 [min, max] 这个区间的像素值会被反转。如果 min 与 max 相等，则反转所有大于等于 min(或 max) 的像素值。

异常：

- **TypeError** - 当 threshold 的类型不为 tuple。
- **ValueError** - 当 threshold 取值不在 [0, 255] 范围内。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomSolarize(threshold=(10, 100))]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])

```

(下页继续)

(续上页)

```
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RandomSolarize(threshold=(1, 10))(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RandomVerticalFlip

class mindspore.dataset.vision.RandomVerticalFlip (*prob=0.5*)

以给定的概率，对输入图像在垂直方向进行随机翻转。

参数：

- **prob** (float, 可选) - 图像被翻转的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 如果 *prob* 不是 float 类型。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围。
- **RuntimeError** - 如果输入的 Tensor 不是 <H, W> 或 <H, W, C> 格式。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.RandomVerticalFlip(0.25)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.array([[0, 1, 2, 3, 4, 5]], dtype=np.uint8).reshape((2, 3))
>>> output = vision.RandomVerticalFlip(1.0)(data)
>>> print(output.shape, output.dtype)
(2, 3) uint8
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.RandomVerticalFlipWithBBox`

class `mindspore.dataset.vision.RandomVerticalFlipWithBBox` (*prob=0.5*)

以给定的概率，对输入图像和边界框在垂直方向进行随机翻转。

参数：

- **prob** (float, 可选) - 图像被翻转的概率，取值范围：[0.0, 1.0]。默认值：0.5。

异常：

- **TypeError** - 如果 *prob* 不是 float 类型。
- **ValueError** - 如果 *prob* 不在 [0.0, 1.0] 范围。
- **RuntimeError** - 如果输入的 Tensor 不是 <H, W> 或 <H, W, C> 格式。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(np.
    ↳float32))
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[func],
...                                                    input_columns=["image"],
...                                                    output_columns=["image", "bbox"])
>>> transforms_list = [vision.RandomVerticalFlipWithBBox(0.20)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↳columns=["image", "bbox"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["bbox"].shape, item["bbox"].dtype)
...     break
(100, 100, 3) float32
(1, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(data.
    ↳dtype))
>>> func_data, func_bboxes = func(data)
>>> output = vision.RandomVerticalFlipWithBBox(1)(func_data, func_bboxes)
>>> print(output[0].shape, output[0].dtype)
(100, 100, 3) float32
>>> print(output[1].shape, output[1].dtype)
(1, 4) float32
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.Rescale`

class `mindspore.dataset.vision.Rescale` (*rescale, shift*)

基于给定的缩放和平移因子，调整图像的像素大小。输出图像的像素大小为： $\text{output} = \text{image} * \text{rescale} + \text{shift}$ 。

说明：此操作默认通过 CPU 执行，也支持异构加速到 GPU 或 Ascend 上执行。

参数：

- **rescale** (float) - 缩放因子。
- **shift** (float) - 平移因子。

异常：

- **TypeError** - 当 *rescale* 的类型不为 float。
- **TypeError** - 当 *shift* 的类型不为 float。

支持平台：

CPU GPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Rescale(1.0 / 255.0, -1.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Rescale(1.0 / 255.0, -1.0)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) float32
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.Resize

class mindspore.dataset.vision.Resize (size, interpolation=Inter.LINEAR)

使用给定的`mindspore.dataset.vision.Inter`插值方式，将输入图像调整至给定的尺寸大小。

支持 Ascend 硬件加速，需要通过`.device("Ascend")`方式开启。

参数：

- **size** (Union[int, Sequence[int]]) - 图像的输出尺寸大小。若输入整型，将调整图像的较短边长度为 `size`，且保持图像的宽高比不变；若输入是 2 元素组成的序列，其输入格式需要是 (高度, 宽度)。
 - CPU 模式：通过`.device("CPU")`设定执行设备为 CPU 时，`size`取值范围：[1, 16777216]。
 - Ascend 模式：通过`.device("Ascend")`设定执行设备为 Ascend 时，`size`取值范围：[6, 32768]。
- **interpolation** (`Inter`, 可选) - 图像插值方法。可选值详见`mindspore.dataset.vision.Inter`。默认值：`Inter.LINEAR`。
 - Ascend 模式：通过`.device("Ascend")`设定执行设备为 Ascend 时，`Inter.ANTIALIAS`、`Inter.AREA`、`Inter.PILCUBIC`插值方法不支持。

异常：

- **TypeError** - 当 `size` 的类型不为 `int` 或 `Sequence[int]`。
- **TypeError** - 当 `interpolation` 的类型不为`mindspore.dataset.vision.Inter`。
- **ValueError** - 当 `size` 不为正数。
- **RuntimeError** - 如果输入的 Tensor 不是 <H, W> 或 <H, W, C> 格式。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> resize_op = vision.Resize([100, 75], Inter.BICUBIC)
>>> transforms_list = [resize_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 75, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Resize([5, 5], Inter.BICUBIC)(data)
>>> print(output.shape, output.dtype)
(5, 5, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target='CPU'*)

指定该变换执行的设备。

当执行设备是 Ascend 时，输入/输出数据的维度限制在 [4, 6] 和 [32768, 32768] 范围内。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> resize_op = vision.Resize([100, 75], Inter.BICUBIC).device("Ascend")
>>> transforms_list = [resize_op]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
    ↪columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
    ↪numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 75, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Resize([25, 25], Inter.BICUBIC).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(25, 25, 3) uint8
```

mindspore.dataset.vision.ResizedCrop

class mindspore.dataset.vision.**ResizedCrop** (*top, left, height, width, size, interpolation=Inter.BILINEAR*)

裁剪输入图像的指定区域，并缩放到指定尺寸大小。

支持 Ascend 硬件加速，需要通过 *.device("Ascend")* 方式开启。

参数：

- **top** (int) - 裁剪区域左上角位置的纵坐标。
- **left** (int) - 裁剪区域左上角位置的横坐标。
- **height** (int) - 裁剪区域的高度。

- **width** (int) - 裁剪区域的宽度。
- **size** (Union[int, Sequence[int, int]]) - 图像的输出尺寸大小。
 - 若输入 int, 将调整图像的较短边长度为 *size*, 且保持图像的宽高比不变。
 - 若输入 Sequence[int, int], 其输入格式需要是 (高度, 宽度)。
- **interpolation** (*Inter*, 可选) - 图像插值方法。可选值详见 *mindspore.dataset.vision.Inter*。默认值: *Inter.BILINEAR*。

异常:

- **TypeError** - 如果 *top* 不为 int 类型。
- **ValueError** - 如果 *top* 为负数。
- **TypeError** - 如果 *left* 不为 int 类型。
- **ValueError** - 如果 *left* 为负数。
- **TypeError** - 如果 *height* 不为 int 类型。
- **ValueError** - 如果 *height* 不为正数。
- **TypeError** - 如果 *width* 不为 int 类型。
- **ValueError** - 如果 *width* 不为正数。
- **TypeError** - 如果 *size* 不为 int 或 Sequence[int, int] 类型。
- **ValueError** - 如果 *size* 不为正数。
- **TypeError** - 如果 *interpolation* 不为 *mindspore.dataset.vision.Inter* 类型。
- **RuntimeError** - 如果输入图像的形状不是 <H, W> 或 <H, W, C>。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> transforms_list = [vision.ResizedCrop(0, 0, 64, 64, (100, 75), Inter.BILINEAR)]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 75, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.ResizedCrop(0, 0, 1, 1, (5, 5), Inter.BILINEAR)(data)
>>> print(output.shape, output.dtype)
(5, 5, 3) uint8
```


教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时，输入数据支持 *uint8* 和 *float32* 类型，输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 32768]，宽度限制范围为 [6, 32768]。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值："CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> resize_crop_op = vision.ResizedCrop(0, 0, 64, 64, (100, 75)).device("Ascend")
>>> transforms_list = [resize_crop_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳ numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 75, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.ResizedCrop(0, 0, 64, 64, (32, 16), Inter.BILINEAR).device("Ascend
↳ ") (data)
>>> print(output.shape, output.dtype)
(32, 16, 3) uint8
```

mindspore.dataset.vision.ResizeWithBBox

class mindspore.dataset.vision.**ResizeWithBBox** (*size*, *interpolation=Inter.LINEAR*)

将输入图像调整为给定的尺寸大小，并相应地调整边界框的大小。

参数：

- **size** (Union[int, Sequence[int]]) - 图像的输出尺寸大小。若输入整型，将调整图像的较短边长度为 *size*，且保持图像的宽高比不变；若输入是 2 元素组成的序列，其输入格式需要是 (高度, 宽度)。
- **interpolation** (*Inter*, 可选) - 图像插值方法。可选值详见 *mindspore.dataset.vision.Inter*。默认值：Inter.LINEAR。

异常：

- **TypeError** - 当 *size* 的类型不为 int 或 Sequence[int]。
- **TypeError** - 当 *interpolation* 的类型不为 *mindspore.dataset.vision.Inter*。
- **ValueError** - 当 *size* 不为正数。
- **RuntimeError** - 如果输入的 Tensor 不是 <H, W> 或 <H, W, C> 格式。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(np.
↳ float32))
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[func],
...                                                    input_columns=["image"],
...                                                    output_columns=["image", "bbox"])
>>> bbox_op = vision.ResizeWithBBox(50, Inter.NEAREST)
>>> transforms_list = [bbox_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image", "bbox"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     print(item["bbox"].shape, item["bbox"].dtype)
...     break
(50, 50, 3) float32
(1, 4) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.float32)
>>> func = lambda img: (data, np.array([[0, 0, data.shape[1], data.shape[0]]]).astype(data.
↳ dtype))
>>> func_data, func_bboxes = func(data)
>>> output = vision.ResizeWithBBox(100)(func_data, func_bboxes)
>>> print(output[0].shape, output[0].dtype)
(100, 100, 3) float32
```

(下页继续)

(续上页)

```
>>> print(output[1].shape, output[1].dtype)
(1, 4) float32
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.RgbToHsv

class mindspore.dataset.vision.**RgbToHsv** (*is_hwc=False*)

将输入的 RGB 格式 `numpy.ndarray` 图像转换为 HSV 格式。

参数：

- **is_hwc** (bool) - 若为 `True`，表示输入图像的 shape 为 `<H, W, C>` 或 `<N, H, W, C>`；否则为 `<C, H, W>` 或 `<N, C, H, W>`。默认值：`False`。

异常：

- **TypeError** - 当 `is_hwc` 的类型不为 `bool`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> transforms_list = Compose([vision.CenterCrop(20),
...                             vision.ToTensor(),
...                             vision.RgbToHsv()])
>>> # apply the transform to dataset through map function
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 20, 20) float64
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.RgbToHsv(is_hwc=True)(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) float64
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.Rotate

class mindspore.dataset.vision.Rotate (*degrees, resample=Inter.NEAREST, expand=False, center=None, fill_value=0*)

将输入图像旋转指定的度数。

支持 Ascend 硬件加速，需要通过 `.device("Ascend")` 方式开启。

参数：

- **degrees** (Union[int, float]) - 旋转角度。
- **resample** (*Inter*, 可选) - 图像插值方法。可选值详见 `mindspore.dataset.vision.Inter`。默认值：`Inter.NEAREST`。
- **expand** (bool, 可选) - 若为 `True`，将扩展图像尺寸大小使其足以容纳整个旋转图像；若为 `False`，则保持图像尺寸大小不变。请注意，扩展时将假设图像为中心旋转，且未进行平移。默认值：`False`。
- **center** (tuple, 可选) - 可选的旋转中心，以图像左上角为原点，旋转中心的位置按照 (宽度, 高度) 格式指定。默认值：`None`，表示中心旋转。
- **fill_value** (Union[int, tuple[int]], 可选) - 旋转图像之外区域的像素填充值。若输入 3 元素元组，将分别用于填充 R、G、B 通道；若输入整型，将以该值填充 RGB 通道。`fill_value` 值必须在 `[0, 255]` 范围内。默认值：`0`。

异常：

- **TypeError** - 当 `degrees` 的类型不为 `int` 或 `float`。
- **TypeError** - 当 `resample` 的类型不为 `mindspore.dataset.vision.Inter`。
- **TypeError** - 当 `expand` 的类型不为 `bool`。
- **TypeError** - 当 `center` 的类型不为 `tuple`。
- **TypeError** - 当 `fill_value` 的类型不为 `int` 或 `tuple[int]`。
- **ValueError** - 当 `fill_value` 取值不在 `[0, 255]` 范围内。
- **RuntimeError** - 当输入图像的 `shape` 不为 `<H, W>` 或 `<..., H, W, C>`。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> transforms_list = [vision.Rotate(degrees=30.0, resample=Inter.NEAREST, expand=True)]
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(137, 137, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
```

(下页继续)

(续上页)

```
>>> output = vision.Rotate(degrees=30.0, resample=Inter.NEAREST, expand=True)(data)
>>> print(output.shape, output.dtype)
(137, 137, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

- 当执行设备是 Ascend 时，输入数据支持 *uint8* 或者 *float32* 类型，输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 32768]，宽度限制范围为 [6, 32768]。
- 当执行设备是 Ascend 并且参数 *expand* 为 *True* 时，参数 *center* 不生效，图像会按照图像中心点旋转。

参数：

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常：

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 300, 400, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> rotate_op = vision.Rotate(degrees=90.0, resample=Inter.NEAREST, expand=True).device(
↳ "Ascend")
>>> transforms_list = [rotate_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳ numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(400, 300, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(300, 400, 3)).astype(np.uint8)
>>> output = vision.Rotate(degrees=90.0, resample=Inter.NEAREST, expand=True).device(
↳ "Ascend")(data)
>>> print(output.shape, output.dtype)
(400, 300, 3) uint8
```

mindspore.dataset.vision.SlicePatches

class mindspore.dataset.vision.SlicePatches (num_height=1, num_width=1, slice_mode=SliceMode.PAD, fill_value=0)

在水平和垂直方向上将 Tensor 切片为多个块。适合于 Tensor 高宽较大的使用场景。如果将 num_height 和 num_width 都设置为 1，则 Tensor 将保持不变。输出 Tensor 的数量等于 num_height * num_width。

参数：

- **num_height** (int, 可选) - 垂直方向的切块数量。默认值：1。
- **num_width** (int, 可选) - 水平方向的切块数量。默认值：1。
- **slice_mode** (SliceMode, 可选) - 表示填充或丢弃，它可以是 SliceMode.PAD、SliceMode.DROP。默认值：SliceMode.PAD。
- **fill_value** (int, 可选) - 如果 slice_mode 取值为 SliceMode.PAD，该值表示在右侧和底部的边界填充宽度（以像素数计）。fill_value 取值必须在 [0, 255] 范围内。默认值：0。

异常：

- **TypeError** - 当 num_height 不是 int。
- **TypeError** - 当 num_width 不是 int。
- **TypeError** - 当 slice_mode 的类型不为 mindspore.dataset.vision.SliceMode。
- **TypeError** - 当 fill_value 不是 int。
- **ValueError** - 当 num_height 不为正数。
- **ValueError** - 当 num_width 不为正数。
- **ValueError** - 当 fill_value 不在 [0, 255] 范围内。
- **RuntimeError** - 如果输入的 Tensor 不是 <H, W> 或 <H, W, C> 格式。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> # default padding mode
>>> num_h, num_w = (1, 4)
>>> slice_patches_op = vision.SlicePatches(num_h, num_w)
>>> transforms_list = [slice_patches_op]
>>> cols = ['img' + str(x) for x in range(num_h*num_w)]
>>>
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list,
...                                                input_columns=["image"],
...                                                output_columns=cols)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(len(item), item["img0"].shape, item["img0"].dtype)
...     break
4 (100, 25, 3) uint8
```

(下页继续)

(续上页)

```
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.SlicePatches(1, 2)(data)
>>> print(np.array(output).shape, np.array(output).dtype)
(2, 100, 50, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.Solarize

class mindspore.dataset.vision.Solarize(threshold)

通过反转阈值内的所有像素值，对输入图像进行曝光。

支持 Ascend 硬件加速，需要通过 .device("Ascend") 方式开启。

参数：

- **threshold** (Union[float, Sequence[float, float]]) - 反转的像素阈值范围，应该以 (min, max) 的格式提供，其中 min 和 max 是 [0, 255] 范围内的整数，并且 min <= max，那么属于 [min, max] 这个区间的像素值会被反转。如果只提供一个值或 min = max，则反转大于等于 min (max) 的所有像素值。

异常：

- **TypeError** - 如果 threshold 不为 float 或 Sequence[float, float] 类型。
- **ValueError** - 如果 threshold 的取值不在 [0, 255] 范围内。

支持平台：

CPU Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.Solarize(threshold=(10, 100))]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Solarize(threshold=(1, 10))(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

device (*device_target='CPU'*)

指定该变换执行的设备。

当执行设备是 Ascend 时, 输入数据仅支持 *uint8* 类型, 输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192], 宽度限制范围为 [6, 4096]。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> solarize_op = vision.Solarize(threshold=(10, 100)).device("Ascend")
>>> transforms_list = [solarize_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
->numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.Solarize(threshold=(10, 100)).device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

mindspore.dataset.vision.TenCrop

class mindspore.dataset.vision.**TenCrop** (*size, use_vertical_flip=False*)

在输入 PIL 图像的中心与四个角处, 分别裁剪指定尺寸大小的子图, 并将其翻转图一并返回。

参数:

- **size** (Union[int, Sequence[int, int]]) - 裁剪子图的尺寸大小。若输入 int, 则以该值为边长裁剪 (*size, size*) 尺寸大小的子图; 若输入 Sequence[int, int], 则以 2 个元素分别为高和宽裁剪子图。
- **use_vertical_flip** (bool, 可选) - 若为 True, 将对子图进行垂直翻转; 否则进行水平翻转。默认值: False。

异常:

- **TypeError** - 当 `size` 的类型不为 `int` 或 `Sequence[int, int]`。
- **TypeError** - 当 `use_vertical_flip` 的类型不为 `bool`。
- **ValueError** - 当 `size` 不为正数。

支持平台：

CPU

样例：

```
>>> import os
>>> import numpy as np
>>> from PIL import Image, ImageDraw
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> class MyDataset:
...     def __init__(self):
...         self.data = []
...         img = Image.new("RGB", (300, 300), (255, 255, 255))
...         draw = ImageDraw.Draw(img)
...         draw.ellipse(((0, 0), (100, 100)), fill=(255, 0, 0), outline=(255, 0, 0),
... width=5)
...         img.save("./1.jpg")
...         data = np.fromfile("./1.jpg", np.uint8)
...         self.data.append(data)
...
...     def __getitem__(self, index):
...         return self.data[0]
...
...     def __len__(self):
...         return 5
>>>
>>> my_dataset = MyDataset()
>>> generator_dataset = ds.GeneratorDataset(my_dataset, column_names="image")
>>> transforms_list = Compose([vision.Decode(to_pil=True),
...                             vision.TenCrop(size=200),
...                             # 4D stack of 10 images
...                             lambda *images: np.stack([vision.ToTensor()(image) for image_
... in images])])
>>> # apply the transform to dataset through map function
>>> generator_dataset = generator_dataset.map(operations=transforms_list, input_columns=
... "image")
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(10, 3, 200, 200) float32
>>> os.remove("./1.jpg")
>>>
>>> # Use the transform in eager mode
>>> img = Image.new("RGB", (300, 300), (255, 255, 255))
>>> draw = ImageDraw.Draw(img)
>>> draw.polygon([(50, 50), (150, 50), (100, 150)], fill=(0, 255, 0), outline=(0, 255, 0))
>>> img.save("./2.jpg")
>>> data = Image.open("./2.jpg")
>>> output = vision.TenCrop(size=200)(data)
```

(下页继续)

(续上页)

```
>>> print(len(output), np.array(output[0]).shape, np.array(output[0]).dtype)
10 (200, 200, 3) uint8
>>> os.remove("./2.jpg")
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.ToNumpy

class mindspore.dataset.vision.ToNumpy

将输入 PIL 图像转换为 numpy.ndarray 图像。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> # Use ToNumpy to explicitly select C++ implementation of subsequent op
>>> transforms_list = Compose([vision.RandomHorizontalFlip(0.5),
...                             vision.ToNumpy(),
...                             vision.Resize((50, 60))])
>>> # apply the transform to dataset through map function
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(50, 60, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = list(np.random.randint(0, 255, size=(32, 32, 3, 3)).astype(np.int32))
>>> output = vision.ToNumpy()(data)
>>> print(type(output), output.shape, output.dtype)
<class 'numpy.ndarray'> (32, 32, 3, 3) int32
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.ToPIL

class mindspore.dataset.vision.ToPIL

将 `numpy.ndarray` 格式的解码图像转换为 `PIL.Image.Image` 格式的图像。

异常:

- **TypeError** - 当输入图像的类型不为 `numpy.ndarray` 或 `PIL.Image.Image`。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> # data is already decoded, but not in PIL Image format
>>> transforms_list = Compose([vision.ToPIL(),
...                             vision.RandomHorizontalFlip(0.5),
...                             vision.ToTensor()])
>>> # apply the transform to dataset through map function
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.ToPIL()(data)
>>> print(type(output), np.array(output).shape, np.array(output).dtype)
<class 'PIL.Image.Image'> (100, 100, 3) uint8
```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.ToTensor

class mindspore.dataset.vision.ToTensor (*output_type=np.float32*)

将输入 `PIL` 图像或 `numpy.ndarray` 图像转换为指定类型的 `numpy.ndarray` 图像, 图像的像素值范围将从 `[0, 255]` 放缩为 `[0.0, 1.0]`, `shape` 将从 `<H, W, C>` 调整为 `<C, H, W>`。

参数:

- **output_type** (`Union[mindspore.dtype, numpy.dtype]`, 可选) - 输出图像的数据类型。默认值: `np.float32`。

异常:

- **TypeError** - 当输入图像的类型不为 `PIL.Image.Image` 或 `numpy.ndarray`。
- **TypeError** - 输入图像的维度不为 2 或 3。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> # create a list of transformations to be applied to the "image" column of each data row
>>> transforms_list = Compose([vision.RandomHorizontalFlip(0.5),
...                             vision.ToTensor()])
>>> # apply the transform to dataset through map function
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.ToTensor()(data)
>>> print(output.shape, output.dtype)
(3, 100, 100) float32
```

教程样例：

- 视觉变换样例库

`mindspore.dataset.vision.ToType`

class `mindspore.dataset.vision.ToType` (*data_type*)

将输入转换为指定的 MindSpore 数据类型或 NumPy 数据类型。

效果同 `mindspore.dataset.transforms.TypeCast`。

说明：此操作默认通过 CPU 执行，也支持异构加速到 GPU 或 Ascend 上执行。

参数：

- **data_type** (`Union[mindspore.dtype, numpy.dtype]`) - 输出图像的数据类型，例如 `numpy.float32`。

异常：

- **TypeError** - 当 *data_type* 的类型不为 `mindspore.dtype` 或 `numpy.dtype`。

支持平台：

CPU GPU Ascend

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> import numpy as np
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = Compose([vision.RandomHorizontalFlip(0.5),
...                             vision.ToTensor(),
...                             vision.ToType(np.float32)])
>>> # apply the transform to dataset through map function
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.array([2.71606445312564e-03, 6.3476562564e-03]).astype(np.float64)
>>> output = vision.ToType(np.float32)(data)
>>> print(output, output.dtype)
[0.00271606 0.00634766] float32

```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.TrivialAugmentWide

class mindspore.dataset.vision.TrivialAugmentWide (num_magnitude_bins=31, interpolation=Inter.NEAREST, fill_value=0)

对输入图像应用 TrivialAugmentWide 数据增强方法。

参考论文 [TrivialAugmentWide: Tuning-free Yet State-of-the-Art Data Augmentation](#)。

只支持 3 通道 RGB 图像。

参数：

- **num_magnitude_bins** (int, 可选) - 不同变换幅值的个数，需不小于 2。默认值：31。
- **interpolation** (*Inter*, 可选) - 图像插值方法。可选值详见 *mindspore.dataset.vision.Inter*。默认值：Inter.NEAREST。
- **fill_value** (Union[int, tuple[int, int, int]], 可选) - 变换后超出原图外区域的像素填充值，取值需在 [0, 255] 范围内。默认值：0。如果输入 int，将用于填充所有 RGB 通道。如果输入 tuple[int, int, int]，则分别用于填充 R、G、B 通道。

异常：

- **TypeError** - 如果 *num_magnitude_bins* 不是 int 类型。
- **ValueError** - 如果 *num_magnitude_bins* 小于 2。
- **TypeError** - 如果 *interpolation* 不是 *mindspore.dataset.vision.Inter* 类型。
- **TypeError** - 如果 *fill_value* 不是 int 或 tuple[int, int, int] 类型。

- **ValueError** - 如果 *fill_value* 取值不在 [0, 255] 范围。
- **RuntimeError** - 如果输入图像的形状不是 <H, W, C>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import Inter
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.TrivialAugmentWide(num_magnitude_bins=31,
...                                              interpolation=Inter.NEAREST,
...                                              fill_value=0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
->columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.TrivialAugmentWide()(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

教程样例：

- 视觉变换样例库

mindspore.dataset.vision.UniformAugment

class mindspore.dataset.vision.UniformAugment (*transforms*, *num_ops*=2)

从指定序列中均匀采样一批数据处理操作，并按顺序随机执行，即采样出的操作也可能不被执行。

序列中的所有数据处理操作要求具有相同的输入和输出类型。后一个操作能够处理前一个操作的输出数据。

参数：

- **transforms** (Sequence) - 数据处理操作序列。
- **num_ops** (int, 可选) - 均匀采样的数据处理操作数。默认值：2。

异常：

- **TypeError** - 当 *transforms* 的类型不为数据处理操作序列。
- **TypeError** - 当 *num_ops* 的类型不为 int。
- **ValueError** - 当 *num_ops* 不为正数。

支持平台：

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.transforms import Compose
>>>
>>> # Use the transform in dataset pipeline mode
>>> seed = ds.config.get_seed()
>>> ds.config.set_seed(12345)
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transform = [vision.CenterCrop(64),
...              vision.RandomColor(),
...              vision.RandomSharpness(),
...              vision.RandomRotation(30)]
>>> transforms_list = Compose([vision.UniformAugment(transform),
...                          vision.ToTensor()])
>>> # apply the transform to dataset through map function
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳ columns="image")
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(3, 100, 100) float32
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> transform = [vision.RandomCrop(size=[20, 40], padding=[32, 32, 32, 32]),
...              vision.RandomCrop(size=[20, 40], padding=[32, 32, 32, 32])]
>>> output = vision.UniformAugment(transform)(data)
>>> print(output.shape, output.dtype)
(20, 40, 3) uint8
>>> ds.config.set_seed(seed)
```

教程样例:

- 视觉变换样例库

mindspore.dataset.vision.VerticalFlip

class mindspore.dataset.vision.VerticalFlip

对输入图像进行垂直翻转。

支持 Ascend 硬件加速, 需要通过 `.device("Ascend")` 方式开启。

异常:

- **RuntimeError** - 如果输入图像的 shape 不是 `<H, W>` 或 `<..., H, W, C>`。

支持平台:

CPU Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
```

(下页继续)

(续上页)

```

>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> transforms_list = [vision.VerticalFlip()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.VerticalFlip()(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8

```

教程样例:

- 视觉变换样例库

device (*device_target*='CPU')

指定该变换执行的设备。

当执行设备是 Ascend 时, 输入数据支持 *uint8* 和 *float32* 类型, 输入数据的通道仅支持 1 和 3。输入数据的高度限制范围为 [4, 8192], 宽度限制范围为 [6, 4096]。

参数:

- **device_target** (str, 可选) - 算子将在指定的设备上运行。当前支持 "CPU" 和 "Ascend"。默认值: "CPU"。

异常:

- **TypeError** - 当 *device_target* 的类型不为 str。
- **ValueError** - 当 *device_target* 的取值不为 ["CPU", "Ascend"]。

支持平台:

CPU Ascend

样例:

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.vision as vision
>>>
>>> # Use the transform in dataset pipeline mode
>>> data = np.random.randint(0, 255, size=(1, 100, 100, 3)).astype(np.uint8)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data, ["image"])
>>> vertical_flip_op = vision.VerticalFlip().device("Ascend")
>>> transforms_list = [vertical_flip_op]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms_list, input_
↳columns=["image"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_
↳numpy=True):

```

(下页继续)

(续上页)

```
...     print(item["image"].shape, item["image"].dtype)
...     break
(100, 100, 3) uint8
>>>
>>> # Use the transform in eager mode
>>> data = np.random.randint(0, 255, size=(100, 100, 3)).astype(np.uint8)
>>> output = vision.VerticalFlip().device("Ascend")(data)
>>> print(output.shape, output.dtype)
(100, 100, 3) uint8
```

13.2.3 工具

<code>mindspore.dataset.vision.AutoAugmentPolicy</code>	不同数据集的自动增强策略。
<code>mindspore.dataset.vision.Border</code>	边界填充方式枚举类。
<code>mindspore.dataset.vision.ConvertMode</code>	图像色彩空间转换模式枚举类。
<code>mindspore.dataset.vision.ImageBatchFormat</code>	图像批处理输出格式枚举类。
<code>mindspore.dataset.vision.ImageReadMode</code>	图像文件读取方式枚举类。
<code>mindspore.dataset.vision.Inter</code>	图像插值方法枚举类。
<code>mindspore.dataset.vision.SliceMode</code>	Tensor 切片方式枚举类。
<code>mindspore.dataset.vision.encode_jpeg</code>	将输入的图像编码为 JPEG 数据。
<code>mindspore.dataset.vision.encode_png</code>	将输入的图像编码为 PNG 数据。
<code>mindspore.dataset.vision.get_image_num_channels</code>	获取输入图像通道数。
<code>mindspore.dataset.vision.get_image_size</code>	获取输入图像的大小为 [高度, 宽度]。
<code>mindspore.dataset.vision.read_file</code>	以二进制模式读取文件。
<code>mindspore.dataset.vision.read_image</code>	读取图像文件并解码为 3 通道 RGB 彩色数据或灰度数据。
<code>mindspore.dataset.vision.read_video</code>	从视频文件中读取视频、音频、元数据。
<code>mindspore.dataset.vision.read_video_timestamps</code>	读取视频文件的时间戳和帧率。
<code>mindspore.dataset.vision.write_file</code>	使用二进制模式将一维 uint8 类型数据数组写到文件。
<code>mindspore.dataset.vision.write_jpeg</code>	将图像数据保存为 JPEG 文件。
<code>mindspore.dataset.vision.write_png</code>	将图像数据保存为 PNG 文件。

mindspore.dataset.vision.AutoAugmentPolicy

class mindspore.dataset.vision.AutoAugmentPolicy

不同数据集的自动增强策略。可能的枚举值包括: AutoAugmentPolicy.IMAGENET、AutoAugmentPolicy.CIFAR10、AutoAugmentPolicy.SVHN。每个策略包含 25 对增强操作。使用 AutoAugment 时, 每个图像都会使用这些操作对中的一个随机转换。每对有 2 个不同的操作。下面显示了所有这些增强操作, 包括操作名称及其概率和随机参数。

- AutoAugmentPolicy.IMAGENET: ImageNet 的数据集自动增强策略。

```
Augmentation operations pair:
[("Posterize", 0.4, 8), ("Rotate", 0.6, 9)),          (("Solarize", 0.6, 5), (
↪ "AutoContrast", 0.6, None)),
  ("Equalize", 0.8, None), ("Equalize", 0.6, None)), ("Posterize", 0.6, 7), ("Posterize
↪ ", 0.6, 6)),
```

(下页继续)

(续上页)

```

(("Equalize", 0.4, None), ("Solarize", 0.2, 4)),      (("Equalize", 0.4, None), ("Rotate",
↪ 0.8, 8)),
(("Solarize", 0.6, 3), ("Equalize", 0.6, None)),      (("Posterize", 0.8, 5), ("Equalize",
↪ 1.0, None)),
(("Rotate", 0.2, 3), ("Solarize", 0.6, 8)),            (("Equalize", 0.6, None), (
↪ "Posterize", 0.4, 6)),
(("Rotate", 0.8, 8), ("Color", 0.4, 0)),              (("Rotate", 0.4, 9), ("Equalize", 0.
↪ 6, None)),
(("Equalize", 0.0, None), ("Equalize", 0.8, None)),    (("Invert", 0.6, None), ("Equalize",
↪ 1.0, None)),
(("Color", 0.6, 4), ("Contrast", 1.0, 8)),            (("Rotate", 0.8, 8), ("Color", 1.0,
↪ 2)),
(("Color", 0.8, 8), ("Solarize", 0.8, 7)),            (("Sharpness", 0.4, 7), ("Invert",
↪ 0.6, None)),
(("ShearX", 0.6, 5), ("Equalize", 1.0, None)),         (("Color", 0.4, 0), ("Equalize", 0.
↪ 6, None)),
(("Equalize", 0.4, None), ("Solarize", 0.2, 4)),      (("Solarize", 0.6, 5), (
↪ "AutoContrast", 0.6, None)),
(("Invert", 0.6, None), ("Equalize", 1.0, None)),     (("Color", 0.6, 4), ("Contrast", 1.
↪ 0, 8)),
(("Equalize", 0.8, None), ("Equalize", 0.6, None))]

```

- AutoAugmentPolicy.CIFAR10: Cifar10 的数据集自动增强策略。

```

Augmentation operations pair:
[("Invert", 0.1, None), ("Contrast", 0.2, 6)),        (("Rotate", 0.7, 2), ("TranslateX
↪ ", 0.3, 9)),
(("Sharpness", 0.8, 1), ("Sharpness", 0.9, 3)),      (("ShearY", 0.5, 8), (
↪ "TranslateY", 0.7, 9)),
(("AutoContrast", 0.5, None), ("Equalize", 0.9, None)), ("ShearY", 0.2, 7), ("Posterize
↪ ", 0.3, 7)),
(("Color", 0.4, 3), ("Brightness", 0.6, 7)),          (("Sharpness", 0.3, 9), (
↪ "Brightness", 0.7, 9)),
(("Equalize", 0.6, None), ("Equalize", 0.5, None)),   (("Contrast", 0.6, 7), (
↪ "Sharpness", 0.6, 5)),
(("Color", 0.7, 7), ("TranslateX", 0.5, 8)),          (("Equalize", 0.8, None), (
↪ "Invert", 0.1, None)),
(("TranslateY", 0.4, 3), ("Sharpness", 0.2, 6)),      (("Brightness", 0.9, 6), ("Color
↪ ", 0.2, 8)),
(("Solarize", 0.5, 2), ("Invert", 0.0, None)),        (("TranslateY", 0.9, 9), (
↪ "TranslateY", 0.7, 9)),
(("Equalize", 0.2, None), ("Equalize", 0.6, None)),   (("Color", 0.9, 9), ("Equalize",
↪ 0.6, None)),
(("AutoContrast", 0.8, None), ("Solarize", 0.2, 8)),   (("Brightness", 0.1, 3), ("Color
↪ ", 0.7, 0)),
(("Solarize", 0.4, 5), ("AutoContrast", 0.9, None)),
(("AutoContrast", 0.9, None), ("Solarize", 0.8, 3)),
(("TranslateY", 0.7, 9), ("AutoContrast", 0.9, None)),
(("Equalize", 0.3, None), ("AutoContrast", 0.4, None)),
(("Equalize", 0.2, None), ("AutoContrast", 0.6, None))]

```

- AutoAugmentPolicy.SVHN: SVHN 的数据集自动增强策略。

```

Augmentation operations pair:
[("ShearX", 0.9, 4), ("Invert", 0.2, None)),          (("ShearY", 0.9, 8), ("Invert", 0.
↪ 7, None)),

```

(下页继续)

(续上页)

```

(("Equalize", 0.6, None), ("Solarize", 0.6, 6)),      (("Invert", 0.9, None), ("Equalize",
↪", 0.6, None)),
(("Equalize", 0.6, None), ("Rotate", 0.9, 3)),      (("ShearX", 0.9, 4), (
↪"AutoContrast", 0.8, None)),
(("ShearY", 0.9, 8), ("Invert", 0.4, None)),      (("ShearY", 0.9, 5), ("Solarize",
↪0.2, 6)),
(("Invert", 0.9, None), ("AutoContrast", 0.8, None)), ("Equalize", 0.6, None), ("Rotate",
↪", 0.9, 3)),
(("ShearX", 0.9, 4), ("Solarize", 0.3, 3)),      (("ShearY", 0.8, 8), ("Invert", 0.
↪7, None)),
(("Equalize", 0.9, None), ("TranslateY", 0.6, 6)),      (("Invert", 0.9, None), ("Equalize",
↪", 0.6, None)),
(("Contrast", 0.3, 3), ("Rotate", 0.8, 4)),      (("Invert", 0.8, None), (
↪"TranslateY", 0.0, 2)),
(("ShearY", 0.7, 6), ("Solarize", 0.4, 8)),      (("Invert", 0.6, None), ("Rotate",
↪0.8, 4)),
(("ShearY", 0.3, 7), ("TranslateX", 0.9, 3)),      (("ShearX", 0.1, 6), ("Invert", 0.
↪6, None)),
(("Solarize", 0.7, 2), ("TranslateY", 0.6, 7)),      (("ShearY", 0.8, 4), ("Invert", 0.
↪8, None)),
(("ShearX", 0.7, 9), ("TranslateY", 0.8, 3)),      (("ShearY", 0.8, 5), (
↪"AutoContrast", 0.7, None)),
(("ShearX", 0.7, 2), ("Invert", 0.1, None))]

```

mindspore.dataset.vision.Border

class mindspore.dataset.vision.Border

边界填充方式枚举类。

可选枚举值为：Border.CONSTANT、Border.EDGE、Border.REFLECT、Border.SYMMETRIC。

- **Border.CONSTANT** - 使用常量值进行填充。
- **Border.EDGE** - 使用各边的边界像素值进行填充。
- **Border.REFLECT** - 以各边的边界为轴进行镜像填充，忽略边界像素值。例如，对 [1,2,3,4] 的两侧分别填充 2 个元素，结果为 [3,2,1,2,3,4,3,2]。
- **Border.SYMMETRIC** - 以各边的边界为轴进行对称填充，包括边界像素值。例如，对 [1,2,3,4] 的两侧分别填充 2 个元素，结果为 [2,1,1,2,3,4,4,3]。

说明：该类派生自 *str* 以支持 JSON 可序列化。

mindspore.dataset.vision.ConvertMode

class mindspore.dataset.vision.ConvertMode

图像色彩空间转换模式枚举类。

可选枚举值如下：

- **ConvertMode.COLOR_BGR2BGRA** - 将 BGR 图像转换为 BGRA 图像。
- **ConvertMode.COLOR_RGB2RGBA** - 将 RGB 图像转换为 RGBA 图像。
- **ConvertMode.COLOR_BGRA2BGR** - 将 BGRA 图像转换为 BGR 图像。

- **ConvertMode.COLOR_RGBA2RGB** - 将 RGBA 图像转换为 RGB 图像。
- **ConvertMode.COLOR_BGR2RGBA** - 将 BGR 图像转换为 RGBA 图像。
- **ConvertMode.COLOR_RGB2BGRA** - 将 RGB 图像转换为 BGRA 图像。
- **ConvertMode.COLOR_RGBA2BGR** - 将 RGBA 图像转换为 BGR 图像。
- **ConvertMode.COLOR_BGRA2RGB** - 将 BGRA 图像转换为 RGB 图像。
- **ConvertMode.COLOR_BGR2RGB** - 将 BGR 图像转换为 RGB 图像。
- **ConvertMode.COLOR_RGB2BGR** - 将 RGB 图像转换为 BGR 图像。
- **ConvertMode.COLOR_BGRA2RGBA** - 将 BGRA 图像转换为 RGBA 图像。
- **ConvertMode.COLOR_RGBA2BGRA** - 将 RGBA 图像转换为 BGRA 图像。
- **ConvertMode.COLOR_BGR2GRAY** - 将 BGR 图像转换为 GRAY 图像。
- **ConvertMode.COLOR_RGB2GRAY** - 将 RGB 图像转换为 GRAY 图像。
- **ConvertMode.COLOR_GRAY2BGR** - 将 GRAY 图像转换为 BGR 图像。
- **ConvertMode.COLOR_GRAY2RGB** - 将 GRAY 图像转换为 RGB 图像。
- **ConvertMode.COLOR_GRAY2BGRA** - 将 GRAY 图像转换为 BGRA 图像。
- **ConvertMode.COLOR_GRAY2RGBA** - 将 GRAY 图像转换为 RGBA 图像。
- **ConvertMode.COLOR_BGRA2GRAY** - 将 BGRA 图像转换为 GRAY 图像。
- **ConvertMode.COLOR_RGBA2GRAY** - 将 RGBA 图像转换为 GRAY 图像。

mindspore.dataset.vision.ImageBatchFormat

class mindspore.dataset.vision.ImageBatchFormat

图像批处理输出格式枚举类。

可选枚举值为：`ImageBatchFormat.NHWC`、`ImageBatchFormat.NCHW`。

- **ImageBatchFormat.NHWC** - 按批次 N、高度 H、宽度 W、通道 C 的顺序存储数据。
- **ImageBatchFormat.NCHW** - 按批次 N、通道 C、高度 H、宽度 W 的顺序存储数据。

mindspore.dataset.vision.ImageReadMode

class mindspore.dataset.vision.ImageReadMode

图像文件读取方式枚举类。

可选枚举值为：`ImageReadMode.UNCHANGED`、`ImageReadMode.GRAYSCALE`、`ImageReadMode.COLOR`。

- **ImageReadMode.UNCHANGED** - 按照图像原始格式读取。
- **ImageReadMode.GRAYSCALE** - 读取并转为单通道灰度数据。
- **ImageReadMode.COLOR** - 读取并换为 3 通道 RGB 彩色数据。

mindspore.dataset.vision.Inter

class mindspore.dataset.vision.Inter

图像插值方法枚举类。

可选值如下：

- **Inter.NEAREST** - 最近邻插值。
- **Inter.ANTIALIAS** - 抗锯齿插值。仅当输入为 *PIL.Image.Image* 时支持。
- **Inter.LINEAR** - 线性插值，实现同 *Inter.BILINEAR*。
- **Inter.BILINEAR** - 双线性插值。
- **Inter.CUBIC** - 三次插值，实现同 *Inter.BICUBIC*。
- **Inter.BICUBIC** - 双三次插值。
- **Inter.AREA** - 像素区域插值。仅当输入为 *numpy.ndarray* 时支持。
- **Inter.PILCUBIC** - 类 Pillow 实现的双三次插值。仅当输入为 *numpy.ndarray* 时支持。

mindspore.dataset.vision.SliceMode

class mindspore.dataset.vision.SliceMode

Tensor 切片方式枚举类。

可选枚举值为：SliceMode.PAD、SliceMode.DROP。

- **SliceMode.PAD** - 当图像无法进行整数块切分时，填充最后一个图像块至指定切分尺寸大小。
- **SliceMode.DROP** - 当图像无法进行整数块切分时，丢弃最后一个图像块。

mindspore.dataset.vision.encode_jpeg

mindspore.dataset.vision.encode_jpeg (image, quality=75)

将输入的图像编码为 JPEG 数据。

参数：

- **image** (Union[numpy.ndarray, mindspore.Tensor]) - 编码的图像。
- **quality** (int, 可选) - 生成的 JPEG 数据的质量，取值范围为 [1, 100]。默认值: 75。

返回：

- *numpy.ndarray*, 一维 uint8 类型数据。

异常：

- **TypeError** - 如果 *image* 不是 *numpy.ndarray* 或 *mindspore.Tensor* 类型。
- **TypeError** - 如果 *quality* 不是 int 类型。
- **RuntimeError** - 如果 *image* 的数据类型不是 uint8 类型。
- **RuntimeError** - 如果 *image* 的 shape 不是 <H, W> 或 <H, W, 1> 或 <H, W, 3>。
- **RuntimeError** - 如果 *quality* 小于 1 或大于 100。

支持平台：

CPU

样例:

```
>>> import mindspore.dataset.vision as vision
>>> import numpy as np
>>> # Generate a random image with height=120, width=340, channels=3
>>> image = np.random.randint(256, size=(120, 340, 3), dtype=np.uint8)
>>> jpeg_data = vision.encode_jpeg(image)
```

mindspore.dataset.vision.encode_png

`mindspore.dataset.vision.encode_png(image, compression_level=6)`

将输入的图像编码为 PNG 数据。

参数:

- **image** (Union[numpy.ndarray, mindspore.Tensor]) - 编码的图像。
- **compression_level** (int, 可选) - 编码压缩因子, 取值范围为 [0, 9]。默认值: 6。

返回:

- numpy.ndarray, 一维 uint8 类型数据。

异常:

- **TypeError** - 如果 *image* 不是 numpy.ndarray 或 mindspore.Tensor 类型。
- **TypeError** - 如果 *compression_level* 不是 int 类型。
- **RuntimeError** - 如果 *image* 的数据类型不是 uint8 类型。
- **RuntimeError** - 如果 *image* 的 shape 不是 <H, W> 或 <H, W, 1> 或 <H, W, 3>。
- **RuntimeError** - 如果 *compression_level* 小于 0 或大于 9。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset.vision as vision
>>> import numpy as np
>>> # Generate a random image with height=120, width=340, channels=3
>>> image = np.random.randint(256, size=(120, 340, 3), dtype=np.uint8)
>>> png_data = vision.encode_png(image)
```

mindspore.dataset.vision.get_image_num_channels

`mindspore.dataset.vision.get_image_num_channels(image)`

获取输入图像通道数。

参数:

- **image** (Union[numpy.ndarray, PIL.Image.Image]) - 需要获取通道数的图像。

返回:

int, 输入图像通道数。

异常:

- **RuntimeError** - *image* 参数的维度小于 2。
- **TypeError** - *image* 参数的类型既不是 `numpy.ndarray`, 也不是 `PIL Image`。

样例:

```
>>> import mindspore.dataset.vision as vision
>>> from PIL import Image
>>> image = Image.open("/path/to/image_file")
>>> num_channels = vision.get_image_num_channels(image)
```

`mindspore.dataset.vision.get_image_size`

`mindspore.dataset.vision.get_image_size(image)`

获取输入图像的大小为 [高度, 宽度]。

参数:

- **image** (`Union[numpy.ndarray, PIL.Image.Image]`) - 要获取大小的图像。

返回:

`list[int, int]`, 图像大小。

异常:

- **RuntimeError** - *image* 参数的维度小于 2。
- **TypeError** - *image* 参数的类型既不是 `numpy.ndarray`, 也不是 `PIL Image`。

样例:

```
>>> import mindspore.dataset.vision as vision
>>> from PIL import Image
>>> image = Image.open("/path/to/image_file")
>>> image_size = vision.get_image_size(image)
```

`mindspore.dataset.vision.read_file`

`mindspore.dataset.vision.read_file(filename)`

以二进制模式读取文件。

参数:

- **filename** (`str`) - 待读取文件路径。

返回:

- `numpy.ndarray`, 一维 `uint8` 类型数据。

异常:

- **TypeError** - 如果 *filename* 不是 `str` 类型。
- **RuntimeError** - 如果 *filename* 不存在或不是普通文件。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset.vision as vision
>>> output = vision.read_file("/path/to/file")
```

`mindspore.dataset.vision.read_image`

`mindspore.dataset.vision.read_image(filename, mode=ImageReadMode.UNCHANGED)`

读取图像文件并解码为 3 通道 RGB 彩色数据或灰度数据。支持的文件类型有 JPEG、PNG、BMP 和 TIFF。

参数：

- **filename** (str) - 待读取图像文件路径。
- **mode** (`ImageReadMode`, 可选) - 图像读取模式。它可以是 `ImageReadMode.UNCHANGED`、`ImageReadMode.GRAYSCALE`、`ImageReadMode.COLOR`。默认值：`ImageReadMode.UNCHANGED`。
 - **ImageReadMode.UNCHANGED** - 按照图像原始格式读取。
 - **ImageReadMode.GRAYSCALE** - 读取并转为单通道灰度数据。
 - **ImageReadMode.COLOR** - 读取并换为 3 通道 RGB 彩色数据。

返回：

- `numpy.ndarray`, 三维 `uint8` 类型数据, `shape` 为 (H, W, C)。

异常：

- **TypeError** - 如果 `filename` 不是 `str` 类型。
- **TypeError** - 如果 `mode` 不是 `mindspore.dataset.vision.ImageReadMode` 类型。
- **RuntimeError** - 如果 `filename` 不存在或不是普通文件或由于格式等原因无法正常读取。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset.vision as vision
>>> from mindspore.dataset.vision import ImageReadMode
>>> output = vision.read_image("/path/to/image_file", ImageReadMode.UNCHANGED)
```

`mindspore.dataset.vision.read_video`

`mindspore.dataset.vision.read_video(filename, start_pts=0, end_pts=None, pts_unit='pts')`

从视频文件中读取视频、音频、元数据。

支持的文件类型有 AVI、H264、H265、MOV、MP4 和 WMV。

参数：

- **filename** (str) - 待读取视频文件路径。
- **start_pts** (Union[float, Fraction, int], 可选) - 视频的开始时间戳。默认值：0。
- **end_pts** (Union[float, Fraction, int], 可选) - 视频的结束时间戳。默认值：None, 对应 2147483647。
- **pts_unit** (str, 可选) - 时间戳的单位, 它可以是 ["pts", "sec"] 中的任何一个。默认值："pts"。

返回：

- `numpy.ndarray`, 四维 `uint8` 视频数据。格式为 `[T, H, W, C]`。“T” 是帧数, “H” 是高度, “W” 是宽度, “C” 是 RGB 的通道。
- `numpy.ndarray`, 二维音频数据。格式为 `[C, L]`。“C” 是通道数, “L” 是一个通道中数据的点数。
- `dict`, 视频和音频的元数据。它包含 `float` 类型的 `video_fps` 数据和 `int` 类型的 `audio_fps` 数据。

异常:

- **TypeError** - 如果 `filename` 不是 `str` 类型。
- **TypeError** - 如果 `start_pts` 的类型不是 `Union[float, Fraction, int]` 类型。
- **TypeError** - 如果 `end_pts` 的类型不是 `Union[float, Fraction, int]` 类型。
- **TypeError** - 如果 `pts_unit` 不是 `str` 类型。
- **RuntimeError** - 如果 `filename` 不存在, 或不是普通文件, 或由于格式等原因无法正常读取。
- **ValueError** - 如果 `start_pts` 小于 0。
- **ValueError** - 如果 `end_pts` 小于 `start_pts`。
- **ValueError** - 如果 `pts_unit` 不在 `["pts", "sec"]` 中。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset.vision as vision
>>> video_output, audio_output, metadata_output = vision.read_video("/path/to/file")
```

`mindspore.dataset.vision.read_video_timestamps`

`mindspore.dataset.vision.read_video_timestamps(filename, pts_unit='pts')`

读取视频文件的时间戳和帧率。支持的文件类型有 AVI、H264、H265、MOV、MP4 和 WMV。

参数:

- **filename** (`str`) - 待读取视频文件路径。
- **pts_unit** (`str`, 可选) - 时间戳的单位, 它可以是 `["pts", "sec"]` 中的任何一个。默认值: `"pts"`。

返回:

- `list`, 当 `pts_unit` 为 `"pts"` 时返回 `list[int]`, 当 `pts_unit` 为 `"sec"` 时返回 `list[float]`。
- `float`, 视频的每秒帧数。

异常:

- **TypeError** - 如果 `filename` 不是 `str` 类型。
- **TypeError** - 如果 `pts_unit` 不是 `str` 类型。
- **RuntimeError** - 如果 `filename` 不存在, 或不是普通文件, 或由于格式等原因无法正常读取。
- **RuntimeError** - 如果 `pts_unit` 不在 `["pts", "sec"]` 中。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset.vision as vision
>>> video_timestamps, video_fps = vision.read_video_timestamps("/path/to/file")
```

`mindspore.dataset.vision.write_file`

`mindspore.dataset.vision.write_file(filename, data)`

使用二进制模式将一维 uint8 类型数据数组写到文件。

参数：

- **filename** (str) - 要写入的文件的路径。
- **data** (Union[numpy.ndarray, mindspore.Tensor]) - 要写入的一维 uint8 数据。

异常：

- **TypeError** - 如果 *filename* 不是 str 类型。
- **TypeError** - 如果 *data* 不是 numpy.ndarray 或 mindspore.Tensor 类型。
- **RuntimeError** - 如果 *filename* 不是普通文件。
- **RuntimeError** - 如果 *data* 的数据类型不是 uint8 类型。
- **RuntimeError** - 如果 *data* 的 shape 不是一维数组。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset.vision as vision
>>> import numpy as np
>>> # Generate a random data with 1024 bytes
>>> data = np.random.randint(256, size=(1024), dtype=np.uint8)
>>> vision.write_file("/path/to/file", data)
```

`mindspore.dataset.vision.write_jpeg`

`mindspore.dataset.vision.write_jpeg(filename, image, quality=75)`

将图像数据保存为 JPEG 文件。

参数：

- **filename** (str) - 要写入的文件的路径。
- **image** (Union[numpy.ndarray, mindspore.Tensor]) - 要写入的图像数据。
- **quality** (int, 可选) - 生成的 JPEG 文件的质量，取值范围为 [1, 100]。默认值: 75 。

异常：

- **TypeError** - 如果 *filename* 不是 str 类型。
- **TypeError** - 如果 *image* 不是 numpy.ndarray 或 mindspore.Tensor 类型。
- **TypeError** - 如果 *quality* 不是 int 类型。
- **RuntimeError** - 如果 *filename* 不存在或不是普通文件。
- **RuntimeError** - 如果 *image* 的数据类型不是 uint8 类型。

- **RuntimeError** - 如果 *image* 的 shape 不是 $\langle H, W \rangle$ 或 $\langle H, W, 1 \rangle$ 或 $\langle H, W, 3 \rangle$ 。
- **RuntimeError** - 如果 *quality* 小于 1 或大于 100。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset.vision as vision
>>> import numpy as np
>>> # Generate a random image with height=120, width=340, channels=3
>>> image = np.random.randint(256, size=(120, 340, 3), dtype=np.uint8)
>>> vision.write_jpeg("/path/to/file", image)
```

`mindspore.dataset.vision.write_png`

`mindspore.dataset.vision.write_png(filename, image, compression_level=6)`

将图像数据保存为 PNG 文件。

参数：

- **filename** (str) - 要写入的文件的路径。
- **image** (Union[numpy.ndarray, mindspore.Tensor]) - 要写入的图像数据。
- **compression_level** (int, 可选) - 生成 PNG 文件的压缩级别，取值范围为 [0, 9]。默认值：6。

异常：

- **TypeError** - 如果 *filename* 不是 str 类型。
- **TypeError** - 如果 *image* 不是 numpy.ndarray 或 mindspore.Tensor 类型。
- **TypeError** - 如果 *compression_level* 不是 int 类型。
- **RuntimeError** - 如果 *filename* 不存在或不是普通文件。
- **RuntimeError** - 如果 *image* 的数据类型不是 uint8 类型。
- **RuntimeError** - 如果 *image* 的 shape 不是 $\langle H, W \rangle$ 或 $\langle H, W, 1 \rangle$ 或 $\langle H, W, 3 \rangle$ 。
- **RuntimeError** - 如果 *compression_level* 小于 0 或大于 9。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset.vision as vision
>>> import numpy as np
>>> # Generate a random image with height=120, width=340, channels=3
>>> image = np.random.randint(256, size=(120, 340, 3), dtype=np.uint8)
>>> vision.write_png("/path/to/file", image)
```

13.3 文本

此模块用于文本数据增强，包括 *transforms* 和 *utils* 两个子模块。
transforms 是一个高性能文本数据增强模块，支持常见的文本数据增强处理。
utils 提供了一些文本处理的工具方法。
在 API 示例中，常用的模块导入方法如下：

```
import mindspore.dataset as ds
from mindspore.dataset import text
```

常用数据处理术语说明如下：

- *TensorOperation*，所有 C++ 实现的数据处理操作的基类。
- *TextTensorOperation*，所有文本数据处理操作的基类，派生自 *TensorOperation*。

数据增强操作可以放入数据处理 Pipeline 中执行，也可以 Eager 模式执行：

- Pipeline 模式用于流式处理大型数据集，示例可参考 [数据处理 Pipeline 介绍](#)。
- Eager 模式用于函数调用方式处理样本，示例可参考 [轻量化数据处理](#)。

13.3.1 样例库

快速上手使用文本变换的 API，跳转参考 [文本变换样例库](#)。此指南中展示了多个变换 API 的用法，以及输入输出结果。

13.3.2 变换

接口名	概述	说明
<code>mindspore.dataset.text.AddToken</code>	将分词 (token) 添加到序列的开头或结尾处。	
<code>mindspore.dataset.text.BasicTokenizer</code>	按照指定规则对输入的 UTF-8 编码字符串进行分词。	Windows 平台尚不支持 <i>BasicTokenizer</i>
<code>mindspore.dataset.text.BertTokenizer</code>	使用 Bert 分词器对字符串进行分词。	Windows 平台尚不支持 <i>BertTokenizer</i>
<code>mindspore.dataset.text.CaseFold</code>	将 UTF-8 编码字符串中的字符规范化为小写，相比 <code>str.lower</code> 支持更多字符。	Windows 平台尚不支持 <i>CaseFold</i>
<code>mindspore.dataset.text.FilterWikipediaXML</code>	将 Wikipedia XML 格式的文本转储过滤为仅由小写字母 (a-z, 从 A-Z 转换而来) 和空格 (从不连续) 组成的“干净”文本。	Windows 平台尚不支持 <i>FilterWikipediaXML</i>
<code>mindspore.dataset.text.JiebaTokenizer</code>	使用 Jieba 分词器对中文字符串进行分词。	隐式马尔可夫模型 (Hidden Markov Model) 分词和最大概率法 (Max Probability) 分词所使用的词典文件可通过
<code>mindspore.dataset.text.Lookup</code>	根据词表，将分词标记 (token) 映射到其索引值 (id)。	
<code>mindspore.dataset.text.Ngram</code>	从 1-D 的字符串生成 N-gram。	

下页继续

表 5 - 续上页

<code>mindspore.dataset.text.NormalizeUTF8</code>	对 UTF-8 编码的字符串进行规范化处理。	Windows 平台尚不支持 <code>NormalizeUTF8</code>
<code>mindspore.dataset.text.PythonTokenizer</code>	使用用户自定义的分词器对输入字符串进行分词。	
<code>mindspore.dataset.text.RegexReplace</code>	使用正则表达式将输入 UTF-8 字符串的部分字符串替换为指定文本。	Windows 平台尚不支持 <code>RegexReplace</code>
<code>mindspore.dataset.text.RegexTokenizer</code>	根据正则表达式对字符串进行分词。	Windows 平台尚不支持 <code>RegexTokenizer</code>
<code>mindspore.dataset.text.SentencePieceTokenizer</code>	使用 <code>SentencePiece</code> 分词器对字符串进行分词。	
<code>mindspore.dataset.text.SlidingWindow</code>	从给定数据构建 <code>Tensor</code> (当前仅支持 1 维), 其中 <code>axis</code> 轴上的每个元素都是从对应位置开始的指定宽度的数据切片。	
<code>mindspore.dataset.text.ToNumber</code>	将字符串的每个元素转换为数字。	
<code>mindspore.dataset.text.ToVectors</code>	根据输入向量表查找向量中的 <code>tokens</code> 。	
<code>mindspore.dataset.text.Truncate</code>	截断输入序列, 使其不超过最大长度。	
<code>mindspore.dataset.text.TruncateSequencePair</code>	对两列 1-D 字符串输入进行截断, 使其总长度小于指定长度。	
<code>mindspore.dataset.text.UnicodeCharTokenizer</code>	对输入字符串中的 <code>Unicode</code> 字符进行分词。	
<code>mindspore.dataset.text.UnicodeScriptTokenizer</code>	使用 <code>UnicodeScript</code> 分词器对 UTF-8 编码的字符串进行分词。	Windows 平台尚不支持 <code>UnicodeScriptTokenizer</code>
<code>mindspore.dataset.text.WhitespaceTokenizer</code>	基于 ICU4C 定义的空白字符 (' ', '\t', '\r', '\n') 对输入的 UTF-8 字符串进行分词。	Windows 平台尚不支持 <code>WhitespaceTokenizer</code>
<code>mindspore.dataset.text.WordpieceTokenizer</code>	将输入的字符串切分为子词。	

mindspore.dataset.text.AddToken

class mindspore.dataset.text.**AddToken** (*token*, *begin=True*)

将分词 (*token*) 添加到序列的开头或结尾处。

参数:

- **token** (str) - 待添加的分词 (*token*)。
- **begin** (bool, 可选) - 选择分词 (*token*) 插入的位置, 若为 `True` 则在序列开头插入, 否则在序列结尾插入。默认值: `True`。

异常:

- **TypeError** - 如果 *token* 的类型不为 `str`。
- **TypeError** - 如果 *begin* 的类型不为 `bool`。

支持平台:

CPU

样例:

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=[['a', 'b', 'c', 'd', 'e']], column_
↳names=["text"])
>>> # Data before
>>> # |           text           |
>>> # +-----+
>>> # | ['a', 'b', 'c', 'd', 'e'] |
>>> # +-----+
>>> add_token_op = text.AddToken(token='TOKEN', begin=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=add_token_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['TOKEN' 'a' 'b' 'c' 'd' 'e']
>>> # Data after
>>> # |           text           |
>>> # +-----+
>>> # | ['TOKEN', 'a', 'b', 'c', 'd', 'e'] |
>>> # +-----+
>>>
>>> # Use the transform in eager mode
>>> data = ["happy", "birthday", "to", "you"]
>>> output = text.AddToken(token='TOKEN', begin=True)(data)
>>> print(output)
['TOKEN' 'happy' 'birthday' 'to' 'you']

```

教程样例：

- 文本变换样例库

`mindspore.dataset.text.BasicTokenizer`

class `mindspore.dataset.text.BasicTokenizer` (*lower_case=False, keep_whitespace=False, normalization_form=NormalizeForm.NONE, preserve_unused_token=True, with_offsets=False*)

按照指定规则对输入的 UTF-8 编码字符串进行分词。

说明：Windows 平台尚不支持 *BasicTokenizer* 。

参数：

- **lower_case** (bool, 可选) - 是否对字符串进行小写转换处理。若为 `True`，会将字符串转换为小写并删除重音字符；若为 `False`，将只对字符串进行规范化处理，其模式由 *normalization_form* 指定。默认值：`False`。
- **keep_whitespace** (bool, 可选) - 是否在分词输出中保留空格。默认值：`False`。
- **normalization_form** (*NormalizeForm*, 可选) - 想要使用的规范化模式。可选值详见 *NormalizeForm*。默认值：`NormalizeForm.NFKC`。
- **preserve_unused_token** (bool, 可选) - 是否保留特殊词汇。若为 `True`，将不会对特殊词汇进行分词，如 `'[CLS]'`，`'[SEP]'`，`'[UNK]'`，`'[PAD]'`，`'[MASK]'` 等。默认值：`True`。
- **with_offsets** (bool, 可选) - 是否输出各 Token 在原字符串中的起始和结束偏移量。默认值：`False`。

异常:

- **TypeError** - 当 `lower_case` 的类型不为 `bool`。
- **TypeError** - 当 `keep_whitespace` 的类型不为 `bool`。
- **TypeError** - 当 `normalization_form` 的类型不为 `NormalizeForm`。
- **TypeError** - 当 `preserve_unused_token` 的类型不为 `bool`。
- **TypeError** - 当 `with_offsets` 的类型不为 `bool`。
- **RuntimeError** - 当输入 Tensor 的数据类型不为 `str`。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore.dataset.text import NormalizeForm
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Welcome To BeiJing!'], column_
↳names=["text"])
>>>
>>> # 1) If with_offsets=False, default output one column [{"text", dtype=str}]
>>> tokenizer_op = text.BasicTokenizer(lower_case=False,
...                                   keep_whitespace=False,
...                                   normalization_form=NormalizeForm.NONE,
...                                   preserve_unused_token=True,
...                                   with_offsets=False)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['Welcome' 'To' 'BeiJing' '!']
>>>
>>> # 2) If with_offsets=True, then output three columns [{"token", dtype=str},
>>> #                                                         ["offsets_start", dtype=uint32],
>>> #                                                         ["offsets_limit", dtype=uint32]}
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Welcome To BeiJing!'], column_
↳names=["text"])
>>> tokenizer_op = text.BasicTokenizer(lower_case=False,
...                                   keep_whitespace=False,
...                                   normalization_form=NormalizeForm.NONE,
...                                   preserve_unused_token=True,
...                                   with_offsets=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=tokenizer_op, input_columns=["text"],
...     output_columns=["token", "offsets_start", "offsets_limit"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["token"], item["offsets_start"], item["offsets_limit"])
['Welcome' 'To' 'BeiJing' '!'] [ 0 12 17 24] [ 7 14 24 25]
>>>
>>> # Use the transform in eager mode
>>> data = 'Welcome To BeiJing!'
>>> output = text.BasicTokenizer()(data)
>>> print(output)
['Welcome' 'To' 'BeiJing' '!']
```

教程样例：

- 文本变换样例库

mindspore.dataset.text.BertTokenizer

```
class mindspore.dataset.text.BertTokenizer (vocab, suffix_indicator='##', max_bytes_per_token=100,
                                             unknown_token='[UNK]', lower_case=False,
                                             keep_whitespace=False, normalization_form=NormalizeForm.NONE,
                                             preserve_unused_token=True, with_offsets=False)
```

使用 Bert 分词器对字符串进行分词。

说明：Windows 平台尚不支持 *BertTokenizer* 。

参数：

- **vocab** (*Vocab*) - 用于查词的词汇表。
- **suffix_indicator** (str, 可选) - 用于指示子词后缀的前缀标志。默认值：'##'。
- **max_bytes_per_token** (int, 可选) - 分词最大长度，超过此长度的词汇将不会被拆分。默认值：100。
- **unknown_token** (str, 可选) - 对未知词汇的分词输出。当设置为空字符串时，直接返回对应未知词汇作为分词输出；否则，返回该字符串作为分词输出。默认值：'[UNK]'。
- **lower_case** (bool, 可选) - 是否对字符串进行小写转换处理。若为 True，会将字符串转换为小写并删除重音字符；若为 False，将只对字符串进行规范化处理，其模式由 *normalization_form* 指定。默认值：False。
- **keep_whitespace** (bool, 可选) - 是否在分词输出中保留空格。默认值：False。
- **normalization_form** (*NormalizeForm*, 可选) - 想要使用的规范化模式。可选值详见 *NormalizeForm*。默认值：*NormalizeForm.NFKC*。
- **preserve_unused_token** (bool, 可选) - 是否保留特殊词汇。若为 True，将不会对特殊词汇进行分词，如 '[CLS]', '[SEP]', '[UNK]', '[PAD]', '[MASK]' 等。默认值：True。
- **with_offsets** (bool, 可选) - 是否输出各 Token 在原字符串中的起始和结束偏移量。默认值：False。

异常：

- **TypeError** - 当 *vocab* 的类型不为 *mindspore.dataset.text.Vocab*。
- **TypeError** - 当 *suffix_indicator* 的类型不为 str。
- **TypeError** - 当 *max_bytes_per_token* 的类型不为 int。
- **ValueError** - 当 *max_bytes_per_token* 为负数。
- **TypeError** - 当 *unknown_token* 的类型不为 str。
- **TypeError** - 当 *lower_case* 的类型不为 bool。
- **TypeError** - 当 *keep_whitespace* 的类型不为 bool。
- **TypeError** - 当 *normalization_form* 的类型不为 *NormalizeForm*。
- **TypeError** - 当 *preserve_unused_token* 的类型不为 bool。
- **TypeError** - 当 *with_offsets* 的类型不为 bool。

支持平台：

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore.dataset.text import NormalizeForm
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["床前明月光"], column_names=["text"])
>>>
>>> # 1) If with_offsets=False, default output one column [{"text", dtype=str}]
>>> vocab_list = ["床", "前", "明", "月", "光", "疑", "是", "地", "上", "霜", "举", "头", "望",
↳ "低",
...           "思", "故", "乡", "繁", "體", "字", "嘿", "哈", "大", "笑", "嘻", "i", "am",
↳ "mak",
...           "make", "small", "mistake", "##s", "during", "work", "##ing", "hour", "+",
↳ "/",
...           "-", "=", "12", "28", "40", "16", " ", "I", "[CLS]", "[SEP]", "[UNK]",
↳ "[PAD]", "[MASK]"]
>>> vocab = text.Vocab.from_list(vocab_list)
>>> tokenizer_op = text.BertTokenizer(vocab=vocab, suffix_indicator='##', max_bytes_per_
↳ token=100,
...                               unknown_token='[UNK]', lower_case=False, keep_
↳ whitespace=False,
...                               normalization_form=NormalizeForm.NONE, preserve_unused_
↳ token=True,
...                               with_offsets=False)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['床' '前' '明' '月' '光']
>>>
>>> # 2) If with_offsets=True, then output three columns [{"token", dtype=str},
>>> #                                     ["offsets_start", dtype=uint32],
>>> #                                     ["offsets_limit", dtype=uint32]}
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["床前明月光"], column_names=["text"])
>>> tokenizer_op = text.BertTokenizer(vocab=vocab, suffix_indicator='##', max_bytes_per_
↳ token=100,
...                               unknown_token='[UNK]', lower_case=False, keep_
↳ whitespace=False,
...                               normalization_form=NormalizeForm.NONE, preserve_unused_
↳ token=True,
...                               with_offsets=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=tokenizer_op,
...     input_columns=["text"],
...     output_columns=["token", "offsets_start", "offsets_limit"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["token"], item["offsets_start"], item["offsets_limit"])
['床' '前' '明' '月' '光'] [ 0  3  6  9 12] [ 3  6  9 12 15]
>>>
>>> # Use the transform in eager mode
>>> data = "床前明月光"
>>> vocab = text.Vocab.from_list(vocab_list)
>>> tokenizer_op = text.BertTokenizer(vocab=vocab)
>>> output = tokenizer_op(data)
>>> print(output)
['床' '前' '明' '月' '光']
```

教程样例：

- 文本变换样例库

mindspore.dataset.text.CaseFold

class mindspore.dataset.text.CaseFold

将 UTF-8 编码字符串中的字符规范化为小写，相比 `str.lower` 支持更多字符。

支持的输入规范化形式详见 [ICU_Normalizer2](#)。

说明：Windows 平台尚不支持 *CaseFold*。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Welcome To BeiJing!'], column_
↳names=["text"])
>>> case_op = text.CaseFold()
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=case_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
welcome to beijing!
>>>
>>> # Use the transform in eager mode
>>> data = 'Welcome To BeiJing!'
>>> output = text.CaseFold()(data)
>>> print(output)
welcome to beijing!
```

教程样例：

- 文本变换样例库

mindspore.dataset.text.FilterWikipediaXML

class mindspore.dataset.text.FilterWikipediaXML

将 Wikipedia XML 格式的文本转储过滤为仅由小写字母（a-z，从 A-Z 转换而来）和空格（从不连续）组成的“干净”文本。

说明：Windows 平台尚不支持 *FilterWikipediaXML*。

支持平台：

CPU

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["Welcome to China", "!!!", "ABC
↵"],
...                                              column_names=["text"], shuffle=False)
>>> replace_op = text.FilterWikipediaXML()
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=replace_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
...     break
welcome to china
>>>
>>> # Use the transform in eager mode
>>> data = "Welcome to China"
>>> output = replace_op(data)
>>> print(output)
welcome to china
```

教程样例:

- 文本变换样例库

`mindspore.dataset.text.JiebaTokenizer`

class `mindspore.dataset.text.JiebaTokenizer` (*hmm_path*, *mp_path*, *mode=JiebaMode.MIX*, *with_offsets=False*)
使用 Jieba 分词器对中文字符串进行分词。

说明: 隐式马尔可夫模型 (Hidden Markov Model) 分词和最大概率法 (Max Probability) 分词所使用的词典文件可通过 `cppjieba` 开源仓 获取, 请保证文件的有效性与完整性。

参数:

- **hmm_path** (str) - 隐式马尔可夫模型分词所使用的词典文件路径。
- **mp_path** (str) - 最大概率法分词所使用的词典文件路径。
- **mode** (*JiebaMode*, 可选) - 想要使用的分词算法。可选值详见 *JiebaMode*。
- **with_offsets** (bool, 可选) - 是否输出各 Token 在原字符串中的起始和结束偏移量。默认值: `False`。

异常:

- **TypeError** - 当 *hmm_path* 不为 str 类型。
- **TypeError** - 当 *mp_path* 不为 str 类型。
- **TypeError** - 当 *mode* 不为 *JiebaMode* 类型。
- **TypeError** - 当 *with_offsets* 不为 bool 类型。

支持平台:

CPU

样例:

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore.dataset.text import JiebaMode
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["床前明月光"], column_names=["text"])
>>>
>>> # 1) If with_offsets=False, return one data column {"text", dtype=str}}
>>> # The paths to jieba_hmm_file and jieba_mp_file can be downloaded directly from the
↳mindspore repository.
>>> # Refer to https://gitee.com/mindspore/mindspore/blob/v2.6.0-rc1/tests/ut/data/dataset/
↳jiebadict/hmm_model.utf8
>>> # and https://gitee.com/mindspore/mindspore/blob/v2.6.0-rc1/tests/ut/data/dataset/
↳jiebadict/jieba.dict.utf8
>>> jieba_hmm_file = "tests/ut/data/dataset/jiebadict/hmm_model.utf8"
>>> jieba_mp_file = "tests/ut/data/dataset/jiebadict/jieba.dict.utf8"
>>> tokenizer_op = text.JiebaTokenizer(jieba_hmm_file, jieba_mp_file, mode=JiebaMode.MP,
↳with_offsets=False)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['床' '前' '明' '月' '光']
>>>
>>> # 2) If with_offsets=True, return three columns {"token", dtype=str}, {"offsets_start",
↳dtype=uint32},
>>> # {"offsets_limit", dtype=uint32}}
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["床前明月光"], column_names=["text"])
>>> tokenizer_op = text.JiebaTokenizer(jieba_hmm_file, jieba_mp_file, mode=JiebaMode.MP,
↳with_offsets=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op, input_columns=[
↳"text"],
...                                               output_columns=["token", "offsets_start",
↳"offsets_limit"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["token"], item["offsets_start"], item["offsets_limit"])
['床' '前' '明' '月' '光'] [0 3 6] [ 3  6 15]
>>>
>>> # Use the transform in eager mode
>>> data = "床前明月光"
>>> output = text.JiebaTokenizer(jieba_hmm_file, jieba_mp_file, mode=JiebaMode.MP)(data)
>>> print(output)
['床' '前' '明' '月' '光']

```

教程样例：

- 文本变换样例库

add_dict (user_dict)

添加指定的词映射字典到 Vocab 中。

参数：

- **user_dict** (Union[str, dict[str, int]]) - 待添加到 Vocab 中的词映射。若输入类型为 str，表示存储待添加词映射的文件路径，文件的每一行需包含两个字段，间隔一个空格，其中第一个字段表示词本身，第二个字段需为数字，表示词频。无效的行将被忽略，且不返回错误或告警。若输入类型为 dict[str, int]，表示存储待添加词映射的字典，其中键名为词本身，键值为词频。

样例：

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore.dataset.text import JiebaMode
>>>
>>> jieba_hmm_file = "/path/to/jieba/hmm/file"
>>> jieba_mp_file = "/path/to/jieba/mp/file"
>>> user_dict = {"男默女泪": 10}
>>> jieba_op = text.JiebaTokenizer(jieba_hmm_file, jieba_mp_file, mode=JiebaMode.MP)
>>> jieba_op.add_dict(user_dict)
>>>
>>> text_file_list = ["/path/to/text_file_dataset_file"]
>>> text_file_dataset = ds.TextFileDataset(dataset_files=text_file_list)
>>> text_file_dataset = text_file_dataset.map(operations=jieba_op, input_columns=["text
↵"])

```

add_word(word, freq=None)

添加一个指定的词映射到 Vocab 中。

参数:

- **word** (str) - 待添加到 Vocab 中的词。
- **freq** (int, 可选) - 待添加词的词频。词频越高, 单词被分词的机会就越大。默认值: None, 使用默认词频。

样例:

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore.dataset.text import JiebaMode
>>>
>>> jieba_hmm_file = "/path/to/jieba/hmm/file"
>>> jieba_mp_file = "/path/to/jieba/mp/file"
>>> jieba_op = text.JiebaTokenizer(jieba_hmm_file, jieba_mp_file, mode=JiebaMode.MP)
>>> sentence_piece_vocab_file = "/path/to/sentence/piece/vocab/file"
>>> with open(sentence_piece_vocab_file, 'r') as f:
...     for line in f:
...         word = line.split(',')[0]
...         jieba_op.add_word(word)
>>>
>>> text_file_list = ["/path/to/text_file_dataset_file"]
>>> text_file_dataset = ds.TextFileDataset(dataset_files=text_file_list)
>>> text_file_dataset = text_file_dataset.map(operations=jieba_op, input_columns=["text
↵"])

```

mindspore.dataset.text.Lookup

class mindspore.dataset.text.**Lookup**(vocab, unknown_token=None, data_type=mstype.int32)

根据词表, 将分词标记 (token) 映射到其索引值 (id)。

参数:

- **vocab** (*Vocab*) - 词表对象, 用于存储分词和索引的映射。
- **unknown_token** (str, 可选) - 备用词汇, 用于要查找的单词不在词汇表时进行替换。如果单词不在词汇表中, 则查找结果将替换为 *unknown_token* 的值。如果单词不在词汇表中, 且未指定 *unknown_token*, 将抛出运行时错误。默认值: None, 不指定该参数。
- **data_type** (mindspore.dtype, 可选) - Lookup 输出的数据类型。默认值: *mstype.int32*。

异常:

- **TypeError** - 参数 `vocab` 类型不为 `mindspore.dataset.text.Vocab`。
- **TypeError** - 参数 `unknown_token` 类型不为 `str`。
- **TypeError** - 参数 `data_type` 类型不为 `mindspore.dtype`。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["with"], column_names=["text"])
>>> # Load vocabulary from list
>>> vocab = text.Vocab.from_list(["?", "##", "with", "the", "test", "符号"])
>>> # Use Lookup operation to map tokens to ids
>>> lookup = text.Lookup(vocab)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[lookup])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
2
>>>
>>> # Use the transform in eager mode
>>> vocab = text.Vocab.from_list(["?", "##", "with", "the", "test", "符号"])
>>> data = "with"
>>> output = text.Lookup(vocab=vocab, unknown_token="test")(data)
>>> print(output)
2
```

教程样例:

- 文本变换样例库

mindspore.dataset.text.Ngram

class mindspore.dataset.text.Ngram(*n*, *left_pad*=(" ", 0), *right_pad*=(" ", 0), *separator*='')

从 1-D 的字符串生成 N-gram。

关于 N-gram 是什么以及它是如何工作的, 请参阅 [N-gram](#)。

参数:

- **n** (list[int]) - n-gram 中的 *n*, 它是一个正整数列表。例如 *n*=[4, 3], 结果将是 Tensor 包含一个 4-gram 和一个 3-gram 的字符串。如果输入的字符不足以构造一个 n-gram, 则返回一个空字符串。例如在 ["mindspore", "best"] 应用 3-gram 将导致生成一个空字符串。
- **left_pad** (tuple, 可选) - 指定序列的左侧填充, 传入 tuple 的形式为 ("pad_token", pad_width)。pad_width 的上限值为 *n* - 1。例如, 指定 *left_pad*=("_", 2) 将用 "_" 填充序列的左侧。默认值: (" ", 0)。
- **right_pad** (tuple, 可选) - 指定序列的右侧填充, 传入 tuple 的形式为 ("pad_token", pad_width)。pad_width 的上限值为 *n* - 1。例如, 指定 *right_pad*=("_", 2) 将用 "_" 填充序列的右侧。默认值: (" ", 0)。
- **separator** (str, 可选) - 指定用于将字符串连接在一起的分隔符。例如, 如果对 ["mindspore", "amazing"] 应用 2-gram 并指定分隔符为 "-", 结果将是 ["mindspore-amazing"]。默认值: " ", 使用空格作为分隔符。

异常：

- **TypeError** - 参数 *n* 包含的值类型不为 `int`。
- **ValueError** - 参数 *n* 包含的值不为正数。
- **ValueError** - 参数 *left_pad* 不是一个长度 2 的 `Tuple[str, int]`。
- **ValueError** - 参数 *right_pad* 不是一个长度 2 的 `Tuple[str, int]`。
- **TypeError** - 参数 *separator* 的类型不是 `str`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> def gen(texts):
...     for line in texts:
...         yield(np.array(line.split(" "), dtype=str),)
>>> data = ["WildRose Country", "Canada's Ocean Playground", "Land of Living Skies"]
>>> generator_dataset = ds.GeneratorDataset(gen(data), ["text"])
>>> ngram_op = text.Ngram(3, separator="-")
>>> generator_dataset = generator_dataset.map(operations=ngram_op)
>>> for item in generator_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
...     break
['']
>>>
>>> # Use the transform in eager mode
>>> output = ngram_op(data)
>>> print(output)
["WildRose Country-Canada's Ocean Playground-Land of Living Skies"]
```

教程样例：

- 文本变换样例库

`mindspore.dataset.text.NormalizeUTF8`

class `mindspore.dataset.text.NormalizeUTF8` (*normalize_form=NormalizeForm.NFKC*)

对 UTF-8 编码的字符串进行规范化处理。

说明：Windows 平台尚不支持 `NormalizeUTF8`。

参数：

- **normalize_form** (`NormalizeForm`, 可选) - 想要使用的规范化模式。可选值详见 `NormalizeForm`。默认值：`NormalizeForm.NFKC`。

异常：

- **TypeError** - 当 *normalize_form* 不为 `NormalizeForm` 类型。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore.dataset.text import NormalizeForm
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["$", "đ", "q̇", "fi", "25", "î"],
...                                              column_names=["text"], shuffle=False)
>>> normalize_op = text.NormalizeUTF8(normalize_form=NormalizeForm.NFC)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=normalize_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
...     break
$
>>>
>>> # Use the transform in eager mode
>>> data = ["$", "đ", "q̇", "fi", "25", "î"]
>>> output = text.NormalizeUTF8(NormalizeForm.NFKC)(data)
>>> print(output)
['$' 'đ' 'q̇' 'fi' '25' 'î']
```

教程样例:

- 文本变换样例库

mindspore.dataset.text.PythonTokenizer

class mindspore.dataset.text.PythonTokenizer(tokenizer)

使用用户自定义的分词器对输入字符串进行分词。

参数:

- **tokenizer** (Callable) - Python 可调用对象, 要求接收一个 string 参数作为输入, 并返回一个包含多个 string 的列表作为返回值。

异常:

- **TypeError** - 参数 *tokenizer* 不是一个可调用的 Python 对象。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> def my_tokenizer(line):
...     return line.split()
>>>
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Hello world'], column_names=["text"])
```

(下页继续)

(续上页)

```
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=text.PythonTokenizer(my_
↳tokenizer))
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['Hello' 'world']
>>>
>>> # Use the transform in eager mode
>>> data = np.array('Hello world'.encode())
>>> output = text.PythonTokenizer(my_tokenizer)(data)
>>> print(output)
['Hello' 'world']
```

教程样例：

- 文本变换样例库

mindspore.dataset.text.RegexReplace

class mindspore.dataset.text.RegexReplace (pattern, replace, replace_all=True)

使用正则表达式将输入 UTF-8 字符串的部分字符串替换为指定文本。

说明：Windows 平台尚不支持 *RegexReplace* 。

参数：

- **pattern** (str) - 正则表达式，即用来描述或匹配一系列符合某个句法规则的字符串。
- **replace** (str) - 用来替换匹配元素的字符串。
- **replace_all** (bool, 可选) - 是否替换全部匹配元素。若为 False，则只替换第一个成功匹配的元素；否则，替换所有匹配的元素。默认值：True。

异常：

- **TypeError** - 当 *pattern* 不为 str 类型。
- **TypeError** - 当 *replace* 不为 str 类型。
- **TypeError** - 当 *replace_all* 不为 bool 类型。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['apple orange apple orange apple'],
...                                              column_names=["text"])
>>> regex_replace = text.RegexReplace('apple', 'orange')
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=regex_replace)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
```

(下页继续)

(续上页)

```
orange orange orange orange orange
>>>
>>> # Use the transform in eager mode
>>> data = 'onetwoonetwoone'
>>> output = text.RegexReplace(pattern="one", replace="two", replace_all=True)(data)
>>> print(output)
twotwotwotwotwo
```

教程样例：

- 文本变换样例库

mindspore.dataset.text.RegexTokenizer

class mindspore.dataset.text.RegexTokenizer(*delim_pattern*, *keep_delim_pattern*=", with_offsets=False)

根据正则表达式对字符串进行分词。

有关支持的正则表达式的模式，请参阅 <https://unicode-org.github.io/icu/userguide/strings/regexp.html>。

说明：Windows 平台尚不支持 *RegexTokenizer* 。

参数：

- **delim_pattern** (str) - 以正则表达式表示的分隔符，字符串将被正则匹配的分隔符分割。
- **keep_delim_pattern** (str, 可选) - 如果被 *delim_pattern* 匹配的字符串也能被 *keep_delim_pattern* 匹配，就可以此分隔符作为标记 (token) 保存。默认值：'' (空字符)，即分隔符不会作为输出标记保留。
- **with_offsets** (bool, 可选) - 是否输出各 token 在原字符串中的起始和结束偏移量。默认值：False。

异常：

- **TypeError** - 参数 *delim_pattern* 的类型不是 str。
- **TypeError** - 参数 *keep_delim_pattern* 的类型不是 str。
- **TypeError** - 参数 *with_offsets* 的类型不是 bool。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Welcome |, To |, BeiJing!'],
...                                              column_names=["text"])
>>>
>>> # 1) If with_offsets=False, default output is one column [{"text", dtype=str}]
>>> delim_pattern = r"[ |,]"
>>> tokenizer_op = text.RegexTokenizer(delim_pattern, with_offsets=False)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
```

(下页继续)

(续上页)

```

...     print(item["text"])
['Welcome' 'To' 'BeiJing!']
>>>
>>> # 2) If with_offsets=True, then output three columns [{"token", dtype=str},
>>> #                                                         ["offsets_start", dtype=uint32],
>>> #                                                         ["offsets_limit", dtype=uint32]}
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Welcome |, To |, BeiJing!'],
...                                              column_names=["text"])
>>> tokenizer_op = text.RegexTokenizer(delim_pattern, with_offsets=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=tokenizer_op,
...     input_columns=["text"],
...     output_columns=["token", "offsets_start", "offsets_limit"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["token"], item["offsets_start"], item["offsets_limit"])
['Welcome' 'To' 'BeiJing!'] [ 0 13 21] [ 7 15 29]
>>>
>>> # Use the transform in eager mode
>>> data = 'Welcome      To      BeiJing!'
>>> output = text.RegexTokenizer(delim_pattern="To", keep_delim_pattern="To", with_
    ↳ offsets=True)(data)
>>> print(output)
(array(['Welcome      ', 'To', '      BeiJing!'], dtype='<U12'),
 array([ 0, 12, 14], dtype=uint32), array([12, 14, 25], dtype=uint32))

```

教程样例：

- 文本变换样例库

mindspore.dataset.text.SentencePieceTokenizer

class mindspore.dataset.text.SentencePieceTokenizer (*mode*, *out_type*)

使用 SentencePiece 分词器对字符串进行分词。

参数：

- **mode** (Union[str, SentencePieceVocab]) - SentencePiece 模型。如果输入是字符串类型，则代表要加载的 SentencePiece 模型文件的路径；如果输入是 SentencePieceVocab 类型，则要求是构造好的 *SentencePieceVocab* 对象。
- **out_type** (*SPieceTokenizerOutType*) - 分词器输出的类型，可以取值为 *SPieceTokenizerOutType.STRING* 或 *SPieceTokenizerOutType.INT*。
 - *SPieceTokenizerOutType.STRING*，表示 SentencePice 分词器的输出类型是 str。
 - *SPieceTokenizerOutType.INT*，表示 SentencePice 分词器的输出类型是 int。

异常：

- **TypeError** - 参数 *mode* 的类型不是字符串或 *mindspore.dataset.text.SentencePieceVocab*。
- **TypeError** - 参数 *out_type* 的类型不是 *mindspore.dataset.text.SPieceTokenizerOutType*。

支持平台：

CPU

样例：

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore.dataset.text import SentencePieceModel, SPieceTokenizerOutType
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Hello world'], column_names=["text"])
>>> # The paths to sentence_piece_vocab_file can be downloaded directly from the mindspore_
↳ repository. Refer to
>>> # https://gitee.com/mindspore/mindspore/blob/v2.6.0-rc1/tests/ut/data/dataset/test_
↳ sentencepiece/vocab.txt
>>> sentence_piece_vocab_file = "tests/ut/data/dataset/test_sentencepiece/vocab.txt"
>>> vocab = text.SentencePieceVocab.from_file([sentence_piece_vocab_file], 512, 0.9995,
...                                         SentencePieceModel.UNIGRAM, {})
>>> tokenizer = text.SentencePieceTokenizer(vocab, out_type=SPieceTokenizerOutType.STRING)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['_H' 'el' 'lo' '_' 'w' 'or' 'l' 'd']
>>>
>>> # Use the transform in eager mode
>>> data = "Hello world"
>>> vocab = text.SentencePieceVocab.from_file([sentence_piece_vocab_file], 100, 0.9995,
...                                         SentencePieceModel.UNIGRAM, {})
>>> output = text.SentencePieceTokenizer(vocab, out_type=SPieceTokenizerOutType.STRING)(data)
>>> print(output)
['_H' 'e' 'l' 'l' 'o' '_' 'w' 'or' 'l' 'd']

```

教程样例：

- 文本变换样例库

mindspore.dataset.text.SlidingWindow

class mindspore.dataset.text.SlidingWindow (width, axis=0)

从给定数据构建 Tensor (当前仅支持 1 维)，其中 axis 轴上的每个元素都是从对应位置开始的指定宽度的数据切片。

参数：

- **width** (int) - 窗口的宽度。其值必须大于零。
- **axis** (int, 可选) - 沿着哪一个轴计算滑动窗口。默认值：0。

异常：

- **TypeError** - 参数 width 的类型不为 int。
- **ValueError** - 参数 width 的值不为正数。
- **TypeError** - 参数 axis 的类型不为 int。

支持平台：

CPU

样例：

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>

```

(下页继续)

(续上页)

```

>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=[[1, 2, 3, 4, 5]], column_names=["col1
↳"])
>>> # Data before
>>> # |      col1      |
>>> # +-----+
>>> # | [[1, 2, 3, 4, 5]] |
>>> # +-----+
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=text.SlidingWindow(3, 0))
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["col1"])
[[1 2 3] [2 3 4] [3 4 5]]
>>> # Data after
>>> # |      col1      |
>>> # +-----+
>>> # | [[1, 2, 3], |
>>> # | [2, 3, 4], |
>>> # | [3, 4, 5]] |
>>> # +-----+
>>>
>>> # Use the transform in eager mode
>>> data = ["happy", "birthday", "to", "you"]
>>> output = text.SlidingWindow(2, 0)(data)
>>> print(output)
[['happy' 'birthday'] ['birthday' 'to'] ['to' 'you']]

```

教程样例：

- 文本变换样例库

mindspore.dataset.text.ToNumber

class mindspore.dataset.text.ToNumber(*data_type*)

将字符串的每个元素转换为数字。

字符串根据以下链接中指定的规则进行转换，除了任何表示负数的字符串不能转换为无符号整数类型外，规则链接如下：https://en.cppreference.com/w/cpp/string/basic_string/stof，https://en.cppreference.com/w/cpp/string/basic_string/stoul。

参数：

- **data_type** (mindspore.dtype) - 要转换为的数值类型，需要是在 *mindspore.dtype* 定义的数值类型。

异常：

- **TypeError** - 参数 *data_type* 不是 *mindspore.dtype* 类型。
- **RuntimeError** - 字符串类型转换失败，或类型转换时出现溢出。

支持平台：

CPU

样例：

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>> from mindspore import dtype as mstype
>>>

```

(下页继续)

(续上页)

```
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=[["1", "2", "3"]], column_names=["text
↳"])
>>> to_number_op = text.ToNumber(mstype.int8)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=to_number_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
[1 2 3]
>>>
>>> # Use the transform in eager mode
>>> data = ["1", "2", "3"]
>>> output = text.ToNumber(mstype.uint32)(data)
>>> print(output)
[1 2 3]
```

教程样例：

- 文本变换样例库

mindspore.dataset.text.ToVectors

class mindspore.dataset.text.ToVectors (vectors, unk_init=None, lower_case_backup=False)

根据输入向量表查找向量中的 tokens。

参数：

- **vectors** (Vectors) - 向量对象。
- **unk_init** (sequence, 可选) - 用于初始化向量外 (OOV) 令牌的序列。默认值：None，用零向量初始化。
- **lower_case_backup** (bool, 可选) - 是否查找小写的 token。如果为 False，则将查找原始大小写中的每个 token。如果为 True，则将首先查找原始大小写中的每个 token，如果在属性 stoi (字符->索引映射) 的键中找不到，则将查找小写中的 token。默认值：False。

异常：

- **TypeError** - 如果 **unk_init** 不是序列。
- **TypeError** - 如果 **unk_init** 的元素不是 float 或 int 类型。
- **TypeError** - 如果 **lower_case_backup** 不是 bool 类型。

支持平台：

CPU

样例：

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["happy", "birthday", "to", "you"],
↳column_names=["text"])
>>> # Load vectors from file
>>> # The paths to vectors_file can be downloaded directly from the mindspore repository.
↳Refer to
>>> # https://gitee.com/mindspore/mindspore/blob/v2.6.0-rc1/tests/ut/data/dataset/
↳testVectors/vectors.txt
```

(下页继续)

(续上页)

```

>>> vectors_file = "tests/ut/data/dataset/testVectors/vectors.txt"
>>> vectors = text.Vectors.from_file(vectors_file)
>>> # Use ToVectors operation to map tokens to vectors
>>> to_vectors = text.ToVectors(vectors)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=[to_vectors])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
...     break
[0. 0. 0. 0. 0. 0.]
>>>
>>> # Use the transform in eager mode
>>> data = ["happy"]
>>> output = text.ToVectors(vectors)(data)
>>> print(output)
[0. 0. 0. 0. 0. 0.]

```

教程样例：

- 文本变换样例库

mindspore.dataset.text.Truncate

class mindspore.dataset.text.Truncate(max_seq_len)

截断输入序列，使其不超过最大长度。

参数：

- **max_seq_len** (int) - 最大截断长度。

异常：

- **TypeError** - 如果 *max_seq_len* 的类型不是 int。
- **ValueError** - 如果 *max_seq_len* 的值小于或等于 0。
- **RuntimeError** - 如果输入张量的数据类型不是 bool、int、float、double 或者 str。

支持平台：

CPU

样例：

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=[['a', 'b', 'c', 'd', 'e']], column_
↳ names=["text"],
...                                     shuffle=False)
>>> # Data before
>>> # |          col1          |
>>> # +-----+
>>> # | ['a', 'b', 'c', 'd', 'e'] |
>>> # +-----+
>>> truncate = text.Truncate(4)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=truncate, input_columns=["text
↳ "])

```

(下页继续)

(续上页)

```

>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['a' 'b' 'c' 'd']
>>> # Data after
>>> # |          col1          |
>>> # +-----+
>>> # | ['a', 'b', 'c', 'd'] |
>>> # +-----+
>>>
>>> # Use the transform in eager mode
>>> data = ["happy", "birthday", "to", "you"]
>>> output = text.Truncate(2)(data)
>>> print(output)
['happy' 'birthday']

```

教程样例：

- 文本变换样例库

`mindspore.dataset.text.TruncateSequencePair`

class `mindspore.dataset.text.TruncateSequencePair` (*max_length*)

对两列 1-D 字符串输入进行截断，使其总长度小于指定长度。

参数：

- **max_length** (int) - 字符串最大输出总长。当其大于或等于两列输入字符串总长时，不进行截断；否则，优先截取两列输入中的较长者，直至其总长等于该值。

异常：

- **TypeError** - 当 *max_length* 不为 int 类型。

支持平台：

CPU

样例：

```

>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=([[1, 2, 3]], [[4, 5]]), column_names=[
↳ "col1", "col2"])
>>> # Data before
>>> # |   col1   |   col2   |
>>> # +-----+
>>> # | [1, 2, 3] | [4, 5] |
>>> # +-----+
>>> truncate_sequence_pair_op = text.TruncateSequencePair(max_length=4)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=truncate_sequence_pair_op,
...                                              input_columns=["col1", "col2"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["col1"], item["col2"])
[1 2] [4 5]
>>> # Data after

```

(下页继续)

(续上页)

```

>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["北 京", "123", "欢 迎", "你"],
...                                               column_names=["text"], shuffle=False)
>>>
>>> # 1) If with_offsets=False, default output one column [{"text", dtype=str}]
>>> tokenizer_op = text.UnicodeScriptTokenizer(keep_whitespace=True, with_offsets=False)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
...     break
['北' ' ' '京']
>>>
>>> # 2) If with_offsets=True, then output three columns [{"token", dtype=str},
>>> #                                                         ["offsets_start", dtype=uint32],
>>> #                                                         ["offsets_limit", dtype=uint32]}
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["北 京", "123", "欢 迎", "你"],
...                                               column_names=["text"], shuffle=False)
>>> tokenizer_op = text.UnicodeScriptTokenizer(keep_whitespace=True, with_offsets=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=tokenizer_op,
...     input_columns=["text"],
...     output_columns=["token", "offsets_start", "offsets_limit"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["token"], item["offsets_start"], item["offsets_limit"])
...     break
['北' ' ' '京'] [0 3 4] [3 4 7]
>>>
>>> # Use the transform in eager mode
>>> data = "北 京"
>>> unicode_script_tokenizer_op = text.UnicodeScriptTokenizer(keep_whitespace=True, with_
↳ offsets=False)
>>> output = unicode_script_tokenizer_op(data)
>>> print(output)
['北' ' ' '京']

```

教程样例：

- 文本变换样例库

mindspore.dataset.text.WhitespaceTokenizer

class mindspore.dataset.text.WhitespaceTokenizer (*with_offsets=False*)

基于 ICU4C 定义的空白字符 (' ', '\t', '\r', '\n') 对输入的 UTF-8 字符串进行分词。

说明：Windows 平台尚不支持 *WhitespaceTokenizer* 。

参数：

- **with_offsets** (bool, 可选) - 是否输出各 Token 在原字符串中的起始和结束偏移量。默认值：False。

异常：

- **TypeError** - 参数 *with_offsets* 的类型不为 bool。

支持平台：

CPU

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Welcome To BeiJing!'], column_
↳names=["text"])
>>>
>>> # 1) If with_offsets=False, default output one column [{"text", dtype=str}]
>>> tokenizer_op = text.WhitespaceTokenizer(with_offsets=False)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
['Welcome' 'To' 'BeiJing!']
>>>
>>> # 2) If with_offsets=True, then output three columns [{"token", dtype=str},
>>> # ["offsets_start", dtype=uint32],
>>> # ["offsets_limit", dtype=uint32]}
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=['Welcome To BeiJing!'], column_
↳names=["text"])
>>> tokenizer_op = text.WhitespaceTokenizer(with_offsets=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(
...     operations=tokenizer_op,
...     input_columns=["text"],
...     output_columns=["token", "offsets_start", "offsets_limit"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["token"], item["offsets_start"], item["offsets_limit"])
['Welcome' 'To' 'BeiJing!'] [ 0 12 17] [ 7 14 25]
>>>
>>> # Use the transform in eager mode
>>> data = 'Welcome To BeiJing!'
>>> output = text.WhitespaceTokenizer(with_offsets=True)(data)
>>> print(output)
(array(['Welcome', 'To', 'BeiJing!'], dtype='<U8'), array([ 0, 12, 17], dtype=uint32),
array([ 7, 14, 25], dtype=uint32))
```

教程样例:

- 文本变换样例库

`mindspore.dataset.text.WordpieceTokenizer`

class `mindspore.dataset.text.WordpieceTokenizer` (`vocab`, `suffix_indicator='##'`, `max_bytes_per_token=100`, `unknown_token='[UNK]'`, `with_offsets=False`)

将输入的字符串切分为子词。

参数:

- **vocab** (*Vocab*) - 用于查词的词汇表。
- **suffix_indicator** (str, 可选) - 用于指示子词后缀的前缀标志。默认值: '##'。
- **max_bytes_per_token** (int, 可选) - 分词最大长度, 超过此长度的词汇将不会被拆分。默认值: 100。
- **unknown_token** (str, 可选) - 对未知词汇的分词输出。当设置为空字符串时, 直接返回对应未知词汇作为分词输出; 否则, 返回该字符串作为分词输出。默认值: '[UNK]'。

- **with_offsets** (bool, 可选) - 是否输出各 Token 在原字符串中的起始和结束偏移量。默认值: False。

异常:

- **TypeError** - 当 *vocab* 不为 `mindspore.dataset.text.Vocab` 类型。
- **TypeError** - 当 *suffix_indicator* 的类型不为 `str`。
- **TypeError** - 当 *max_bytes_per_token* 的类型不为 `int`。
- **TypeError** - 当 *unknown_token* 的类型不为 `str`。
- **TypeError** - 当 *with_offsets* 的类型不为 `bool`。
- **ValueError** - 当 *max_bytes_per_token* 为负数。

支持平台:

CPU

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> # Use the transform in dataset pipeline mode
>>> seed = ds.config.get_seed()
>>> ds.config.set_seed(12345)
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["happy", "birthday", "to", "you"],
↳column_names=["text"])
>>>
>>> vocab_list = ["book", "cholera", "era", "favor", "##ite", "my", "is", "love", "dur", "##
↳ing", "the"]
>>> vocab = text.Vocab.from_list(vocab_list)
>>>
>>> # If with_offsets=False, default output one column {"text", dtype=str}
>>> tokenizer_op = text.WordpieceTokenizer(vocab=vocab, unknown_token='[UNK]',
...                                       max_bytes_per_token=100, with_offsets=False)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op)
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["text"])
...     break
['[UNK]']
>>>
>>> # If with_offsets=True, then output three columns {"token", dtype=str}, ["offsets_start
↳", dtype=uint32],
>>> #                                       ["offsets_limit", dtype=uint32]}
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=["happy", "birthday", "to", "you"],
↳column_names=["text"])
>>> tokenizer_op = text.WordpieceTokenizer(vocab=vocab, unknown_token='[UNK]',
...                                       max_bytes_per_token=100, with_offsets=True)
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=tokenizer_op, input_columns=[
↳"text"],
...                                       output_columns=["token", "offsets_start",
↳"offsets_limit"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["token"], item["offsets_start"], item["offsets_limit"])
...     break
['[UNK]'] [0] [5]
>>>
>>> # Use the transform in eager mode
>>> data = ["happy", "birthday", "to", "you"]
```

(下页继续)

(续上页)

```
>>> vocab_list = ["book", "cholera", "era", "favor", "**ite", "my", "is", "love", "dur",
↳ "**ing", "the"]
>>> vocab = text.Vocab.from_list(vocab_list)
>>> output = text.WordpieceTokenizer(vocab=vocab, suffix_indicator="y", unknown_token='[UNK]
↳')(data)
>>> print(output)
['[UNK]' '[UNK]' '[UNK]' '[UNK]']
>>> ds.config.set_seed(seed)
```

教程样例：

- 文本变换样例库

13.3.3 工具

接口名	概述	说明
<code>mindspore.dataset.text.CharNGram</code>	CharNGram 预训练词向量。	
<code>mindspore.dataset.text.FastText</code>	FastText 预训练词向量。	
<code>mindspore.dataset.text.GloVe</code>	Global Vectors (GloVe) 预训练词向量。	
<code>mindspore.dataset.text.JiebaMode</code>	Jieba 分词器可使用的分词算法。	
<code>mindspore.dataset.text.NormalizeForm</code>	Unicode 规范化模式。	
<code>mindspore.dataset.text.SentencePieceModel</code>	SentencePiece 的子词算法。	
<code>mindspore.dataset.text.SentencePieceVocab</code>	用于执行分词的 SentencePiece 对象。	
<code>mindspore.dataset.text.SPieceTokenizerLoadType</code>	SentencePiece 分词器的模型输入类型。	
<code>mindspore.dataset.text.SPieceTokenizerOutType</code>	<code>mindspore.dataset.text.SentencePieceTokenizer</code> 输出类型的枚举值。	
<code>mindspore.dataset.text.Vectors</code>	预训练词向量。	
<code>mindspore.dataset.text.Vocab</code>	创建用于训练 NLP 模型的 Vocab。	
<code>mindspore.dataset.text.to_bytes</code>	基于 <code>encoding</code> 字符集对每个元素进行编码，将 <code>string</code> 的 NumPy 数组转换为 <code>bytes</code> 的数组。	
<code>mindspore.dataset.text.to_str</code>	基于 <code>encoding</code> 字符集对每个元素进行解码，借此将 <code>bytes</code> 的 NumPy 数组转换为 <code>string</code> 的数组。	

mindspore.dataset.text.CharNGram

class mindspore.dataset.text.CharNGram

CharNGram 预训练词向量。

将单词或句子表示为字符 n-gram 数量向量，然后使用非线性变换得到低维词向量表示。

classmethod from_file (file_path, max_vectors=None)

加载 CharNGram 预训练向量集文件。

参数：

- **file_path** (str) - CharNGram 预训练向量集文件路径。
- **max_vectors** (int, 可选) - 加载预训练向量的数量上限。大多数预训练向量集按词频降序排列。因此，如果内存不足以存放整个向量集，或者出于其他原因，可以通过该值限制加载的向量数量。默认值：None，没有上限。

返回：

CharNGram，预训练词向量。

异常：

- **TypeError** - 当 *file_path* 不为 str 类型。
- **RuntimeError** - 当 *file_path* 文件路径不存在或没有访问权限。
- **TypeError** - 当 *max_vectors* 不为 int 类型。
- **ValueError** - 当 *max_vectors* 为负数。

样例：

```
>>> import mindspore.dataset.text as text
>>>
>>> char_n_gram = text.CharNGram.from_file("/path/to/char_n_gram/file", max_vectors=None)
>>> to_vectors = text.ToVectors(char_n_gram)
>>> # Look up a token into vectors according CharNGram model.
>>> word_vector = to_vectors(["word1", "word2"])
```

mindspore.dataset.text.FastText

class mindspore.dataset.text.FastText

FastText 预训练词向量。

通过 FastText，可以创建一种无监督或有监督学习算法，以获得单词的向量表示。

classmethod from_file (file_path, max_vectors=None)

加载 FastText 预训练向量集文件。

参数：

- **file_path** (str) - FastText 预训练向量集文件路径。文件后缀需为 *.vec。
- **max_vectors** (int, 可选) - 加载预训练向量的数量上限。大多数预训练向量集按词频降序排列。因此，如果内存不足以存放整个向量集，或者出于其他原因，可以通过该值限制加载的向量数量。默认值：None，没有上限。

返回：

FastText，预训练词向量。

异常:

- **TypeError** - 当 `file_path` 不为 `str` 类型。
- **RuntimeError** - 当 `file_path` 文件路径不存在或没有访问权限。
- **TypeError** - 当 `max_vectors` 不为 `int` 类型。
- **ValueError** - 当 `max_vectors` 为负数。

样例:

```
>>> import mindspore.dataset.text as text
>>> fast_text = text.FastText.from_file("/path/to/fast_text/file", max_vectors=None)
>>> to_vectors = text.ToVectors(fast_text)
>>> # Look up a token into vectors according FastText model.
>>> word_vector = to_vectors(["word1", "word2"])
```

`mindspore.dataset.text.GloVe`

class `mindspore.dataset.text.GloVe`

Global Vectors (GloVe) 预训练词向量。

GloVe 是一种无监督学习算法，用于获取词的向量表示。

classmethod `from_file(file_path, max_vectors=None)`

加载 GloVe 预训练向量集文件。

参数:

- **file_path** (`str`) - GloVe 预训练向量集文件路径。文件名类似 `glove.*.txt`。
- **max_vectors** (`int`, 可选) - 加载预训练向量的数量上限。大多数预训练向量集按词频降序排列。因此，如果内存不足以存放整个向量集，或者出于其他原因，可以通过该值限制加载的向量数量。默认值: `None`，没有上限。

返回:

GloVe，预训练词向量。

异常:

- **TypeError** - 当 `file_path` 不为 `str` 类型。
- **RuntimeError** - 当 `file_path` 文件路径不存在或没有访问权限。
- **TypeError** - 当 `max_vectors` 不为 `int` 类型。
- **ValueError** - 当 `max_vectors` 为负数。

样例:

```
>>> import mindspore.dataset.text as text
>>> glove = text.GloVe.from_file("/path/to/glove/file", max_vectors=None)
>>> to_vectors = text.ToVectors(glove)
>>> # Look up a token into vectors according GloVe model.
>>> word_vector = to_vectors(["word1", "word2"])
```


`mindspore.dataset.text.JiebaMode`

class `mindspore.dataset.text.JiebaMode`

Jieba 分词器可使用的分词算法。

可选值如下：

- `JiebaMode.MIX`：最大概率法（Max Probability）和隐式马尔可夫模型（Hidden Markov Model）混合分词。
- `JiebaMode.MP`：最大概率法（Max Probability）分词。
- `JiebaMode.HMM`：隐式马尔可夫模型（Hidden Markov Model）分词。

`mindspore.dataset.text.NormalizeForm`

class `mindspore.dataset.text.NormalizeForm`

Unicode 规范化模式。

可选值如下：

- **`NormalizeForm.NONE`** - 不进行规范化处理。
- **`NormalizeForm.NFC`** - 先以标准等价方式分解，再以标准等价方式重组。
- **`NormalizeForm.NFKC`** - 先以兼容等价方式分解，再以标准等价方式重组。
- **`NormalizeForm.NFD`** - 以标准等价方式分解。
- **`NormalizeForm.NFKD`** - 以兼容等价方式分解。

`mindspore.dataset.text.SentencePieceModel`

class `mindspore.dataset.text.SentencePieceModel`

`SentencePiece` 的子词算法。

可选值如下：

- `SentencePieceModel.UNIGRAM`：单字符语言模型 子词算法。
- `SentencePieceModel.BPE`：字节对编码 子词算法。
- `SentencePieceModel.CHAR`：基于字符的子词算法。
- `SentencePieceModel.WORD`：基于单词的子词算法。

`mindspore.dataset.text.SentencePieceVocab`

class `mindspore.dataset.text.SentencePieceVocab`

用于执行分词的 `SentencePiece` 对象。

classmethod `from_dataset` (*dataset, col_names, vocab_size, character_coverage, model_type, params*)

从数据集构建 `SentencePiece`。

参数：

- **`dataset`** (`Dataset`) - 表示用于构建 `SentencePiece` 对象的数据集。
- **`col_names`** (`list`) - 表示列名称的列表。
- **`vocab_size`** (`int`) - 表示词汇大小。

- **character_coverage** (float) - 表示模型涵盖的字符数量。推荐值: 0.9995 (适用于具有丰富字符集的语言, 如日文或中文)、1.0 (适用于具有小字符集的其他语言)。
- **model_type** (*SentencePieceModel*) - 想要使用的子词算法。可选值详见*SentencePieceModel*。
- **params** (dict) - 表示没有传入参数的字典。

返回:

SentencePieceVocab, 从数据集构建的 Vocab 对象。

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> from mindspore.dataset.text import SentencePieceVocab, SentencePieceModel
>>> dataset = ds.TextFileDataset("/path/to/sentence/piece/vocab/file", shuffle=False)
>>> vocab = SentencePieceVocab.from_dataset(dataset, ["text"], 5000, 0.9995,
...                                     SentencePieceModel.UNIGRAM, {})
>>> # Build tokenizer based on vocab
>>> tokenizer = text.SentencePieceTokenizer(vocab, out_type=text.SPieceTokenizerOutType.
->STRING)
>>> txt = "Today is Tuesday."
>>> token = tokenizer(txt)
```

classmethod from_file (*file_path, vocab_size, character_coverage, model_type, params*)

从文件中构建一个 SentencePiece 对象。

参数:

- **file_path** (list) - 表示包含 SentencePiece 文件路径的一个列表。
- **vocab_size** (int) - 表示词汇大小。
- **character_coverage** (float) - 表示模型涵盖的字符数量。推荐值: 0.9995 (适用于具有丰富字符集的语言, 如日文或中文)、1.0 (适用于具有小字符集的其他语言)。
- **model_type** (*SentencePieceModel*) - 想要使用的子词算法。可选值详见*SentencePieceModel*。
- **params** (dict) - 表示没有传入参数的字典 (参数派生自 SentencePiece 库)。

返回:

SentencePieceVocab, 表示从文件中构建的 Vocab 对象。

样例:

```
>>> from mindspore.dataset.text import SentencePieceVocab, SentencePieceModel
>>> vocab = SentencePieceVocab.from_file(["/path/to/sentence/piece/vocab/file"], 5000, 0.
->9995,
...                                     SentencePieceModel.UNIGRAM, {})
>>> # Build tokenizer based on vocab model
>>> tokenizer = text.SentencePieceTokenizer(vocab, out_type=text.SPieceTokenizerOutType.
->STRING)
>>> txt = "Today is Friday."
>>> token = tokenizer(txt)
```

classmethod save_model (*vocab, path, filename*)

将模型保存到给定的文件路径。

参数:

- **vocab** (SentencePieceVocab) - 表示一个 SentencePiece 对象。

- **path** (str) - 表示存储模型的路径。
- **filename** (str) - 表示文件名称。

样例:

```
>>> from mindspore.dataset.text import SentencePieceVocab, SentencePieceModel
>>> vocab = SentencePieceVocab.from_file(["/path/to/sentence/piece/vocab/file"], 5000, 0.
↳ 9995,
...                                     SentencePieceModel.UNIGRAM, {})
>>> SentencePieceVocab.save_model(vocab, "./", "m.model")
```

mindspore.dataset.text.SPieceTokenizerLoadType

class mindspore.dataset.text.**SPieceTokenizerLoadType**

SentencePiece 分词器的模型输入类型。

可选值如下:

- **SPieceTokenizerLoadType.FILE**: 从指定文件路径加载模型。
- **SPieceTokenizerLoadType.MODEL**: 从指定词汇表对象加载模型。

mindspore.dataset.text.SPieceTokenizerOutType

class mindspore.dataset.text.**SPieceTokenizerOutType**

mindspore.dataset.text.SentencePieceTokenizer 输出类型的枚举值。

可选的枚举值包括: **SPieceTokenizerOutType.STRING** 和 **SPieceTokenizerOutType.INT**。

- **SPieceTokenizerOutType.STRING** - 表示 SentencePiece 分词器的输出类型为 string。
- **SPieceTokenizerOutType.INT** - 表示 SentencePiece 分词器的输出类型为 int。

mindspore.dataset.text.Vectors

class mindspore.dataset.text.**Vectors**

预训练词向量。

classmethod **from_file** (*file_path*, *max_vectors=None*)

加载预训练向量集文件。

参数:

- **file_path** (str) - 预训练向量集文件路径。
- **max_vectors** (int, 可选) - 加载预训练向量的数量上限。大多数预训练向量集按词频降序排列。因此, 如果内存不足以存放整个向量集, 或者出于其他原因, 可以通过该值限制加载的向量数量。默认值: *None*, 没有上限。

返回:

CharNGram, 预训练词向量。

异常:

- **TypeError** - 当 *file_path* 不为 str 类型。
- **RuntimeError** - 当 *file_path* 文件路径不存在或没有访问权限。

- **TypeError** - 当 `max_vectors` 不为 `int` 类型。
- **ValueError** - 当 `max_vectors` 为负数。

样例:

```
>>> import mindspore.dataset.text as text
>>> vector = text.Vectors.from_file("/path/to/vectors/file", max_vectors=None)
>>> to_vectors = text.ToVectors(vector)
>>> # Look up a token into vectors according Vector model.
>>> word_vector = to_vectors(["word1", "word2"])
```

`mindspore.dataset.text.Vocab`

class `mindspore.dataset.text.Vocab`

创建用于训练 NLP 模型的 `Vocab`。

`Vocab` 是数据集中可能出现的所有 `Token` 的集合，保存了各 `Token` 与其 `ID` 之间的映射关系。

classmethod `from_dataset` (*dataset*, *columns=None*, *freq_range=None*, *top_k=None*, *special_tokens=None*, *special_first=True*)

从给定数据集创建 `Vocab`。

以数据集中的样本作为语料库创建 `Vocab`，其中的 `Token` 按词频从高到低进行排列，具有相同词频的 `Token` 则按字母序进行排列。

参数:

- **dataset** (`Dataset`) - 用于创建 `Vocab` 的数据集。
- **columns** (`list[str]`, 可选) - 用于创建 `Vocab` 的数据列列名。默认值: `None`，使用所有列。
- **freq_range** (`tuple[int, int]`, 可选) - 用于创建 `Vocab` 的词频范围，需包含两个元素，分别表示最低频率与最高频率，在此范围内的 `Token` 将会保留。当最低或最高频率为 `None` 时，表示没有最低或最高频率限制。默认值: `None`，没有词频范围限制。
- **top_k** (`int`, 可选) - 只选取词频最高的前指定个 `Token` 构建 `Vocab`。此操作将在词频筛选后进行。如果该值大于总词数，则所有 `Token` 都将保留。默认值: `None`，没有 `Token` 个数限制。
- **special_tokens** (`list[str]`, 可选) - 追加到 `Vocab` 中的 `Token` 列表。默认值: `None`，不追加 `Token`。
- **special_first** (`bool`, 可选) - 是否将追加的 `Token` 添加到 `Vocab` 首部，否则添加到 `Vocab` 尾部。默认值: `True`。

返回:

`Vocab`，从数据集创建的 `Vocab`。

异常:

- **TypeError** - 当 `columns` 不为 `list[str]` 类型。
- **TypeError** - 当 `freq_range` 不为 `tuple[int, int]` 类型。
- **ValueError** - 当 `freq_range` 中的元素值为负数。
- **TypeError** - 当 `top_k` 不为 `int` 类型。
- **ValueError** - 当 `top_k` 不为正数。
- **TypeError** - 当 `special_tokens` 不为 `list[str]` 类型。
- **ValueError** - 当 `special_tokens` 中存在重复元素。
- **TypeError** - 当 `special_first` 不为 `bool` 类型。

样例:

```
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> dataset = ds.TextFileDataset("/path/to/sentence/piece/vocab/file", shuffle=False)
>>> vocab = text.Vocab.from_dataset(dataset, "text", freq_range=None, top_k=None,
...                               special_tokens=["<pad>", "<unk>"],
...                               special_first=True)
>>> # Use the vocab to look up string to id
>>> lookup = text.Lookup(vocab, "<unk>")
>>> id = lookup("text1")
```

classmethod from_dict (word_dict)

从给定字典创建 Vocab。

参数:

- **word_dict** (dict[str, int]) - 存储各 Token 与其 ID 之间映射关系的字典。

返回:

Vocab, 从字典创建的 Vocab。

异常:

- **TypeError** - 当 word_dict 不为 dict[str, int] 类型。
- **ValueError** - 当 word_dict 中的键值为负数。

样例:

```
>>> import mindspore.dataset.text as text
>>> vocab = text.Vocab.from_dict({"home": 3, "behind": 2, "the": 4, "world": 5, "<unk>": 6})
>>>
>>> # look up ids to string
>>> tokens = vocab.ids_to_tokens([3, 4, 5])
>>> print(tokens)
['home', 'the', 'world']
```

classmethod from_file (file_path, delimiter=",", vocab_size=None, special_tokens=None, special_first=True)

从给定文件创建 Vocab。

参数:

- **file_path** (str) - 用于创建 Vocab 的文件路径。
- **delimiter** (str, 可选) - 文件行中 Token 的分隔符。分隔符前的字符串将被视为一个 Token。默认值: ",", 整行将被视为一个 Token。
- **vocab_size** (int, 可选) - Vocab 包含的 Token 数上限。默认值: None, 没有 Token 数上限。
- **special_tokens** (list[str], 可选) - 追加到 Vocab 中的 Token 列表。默认值: None, 不追加 Token。
- **special_first** (bool, 可选) - 是否将追加的 Token 添加到 Vocab 首部, 否则添加到 Vocab 尾部。默认值: True。

返回:

Vocab, 从文件创建的 Vocab。

异常:

- **TypeError** - 当 file_path 不为 str 类型。

- **TypeError** - 当 *delimiter* 不为 `str` 类型。
- **ValueError** - 当 *vocab_size* 不为正数。
- **TypeError** - 当 *special_tokens* 不为 `list[str]` 类型。
- **ValueError** - 当 *special_tokens* 中存在重复元素。
- **TypeError** - 当 *special_first* 不为 `bool` 类型。

样例:

```
>>> import mindspore.dataset.text as text
>>> # Assume vocab file contains the following content:
>>> # --- begin of file ---
>>> # apple,apple2
>>> # banana, 333
>>> # cat,00
>>> # --- end of file ---
>>>
>>> # Read file through this API and specify "," as delimiter.
>>> # The delimiter will break up each line in file, then the first element is taken to_
↳ be the word.
>>> vocab = text.Vocab.from_file("/path/to/simple/vocab/file", ",", None, ["<pad>", "
↳ <unk>"], True)
>>>
>>> # Finally, there are 5 words in the vocab: "<pad>", "<unk>", "apple", "banana", "cat
↳ ".
>>> vocabulary = vocab.vocab()
>>>
>>> # look up strings to ids
>>> ids = vocab.tokens_to_ids(["apple", "banana"])
```

classmethod from_list (*word_list*, *special_tokens=None*, *special_first=True*)

从给定 Token 列表创建 Vocab。

参数:

- **word_list** (`list[str]`) - 用于创建 Vocab 的 Token 列表。
- **special_tokens** (`list[str]`, 可选) - 追加到 Vocab 中的 Token 列表。默认值: `None`, 不追加 Token。
- **special_first** (`bool`, 可选) - 是否将追加的 Token 添加到 Vocab 首部, 否则添加到 Vocab 尾部。默认值: `True`。

返回:

Vocab, 从 Token 列表创建的 Vocab。

异常:

- **TypeError** - 当 *word_list* 不为 `list[str]` 类型。
- **ValueError** - 当 *word_list* 中存在重复元素。
- **TypeError** - 当 *special_tokens* 不为 `list[str]` 类型。
- **ValueError** - 当 *special_tokens* 中存在重复元素。
- **TypeError** - 当 *special_first* 不为 `bool` 类型。

样例:

```
>>> import mindspore.dataset.text as text
>>> vocab = text.Vocab.from_list(["w1", "w2", "w3"], special_tokens=["<unk>"], special_
↳first=True)
>>> # look up strings to ids
>>> ids = vocab.tokens_to_ids(["w1", "w3"])
```

ids_to_tokens (*ids*)

查找指定 ID 对应的 Token。

参数:

- **ids** (Union[int, list[int], numpy.ndarray]) - 待查找的某个 ID 或 ID 列表。若 ID 不存在, 则返回空字符串。

返回:

Union[str, list[str]], 指定 ID 对应的 Token。

异常:

- **TypeError** - 当 *ids* 不为 Union[int, list[int], numpy.ndarray] 类型。
- **ValueError** - 当 *ids* 中的元素为负数。

样例:

```
>>> import mindspore.dataset.text as text
>>> vocab = text.Vocab.from_list(["w1", "w2", "w3"], special_tokens=["<unk>"], special_
↳first=True)
>>> token = vocab.ids_to_tokens(1)
>>> print(token)
w1
```

tokens_to_ids (*tokens*)

查找指定 Token 对应的 ID。

参数:

- **tokens** (Union[str, list[str], numpy.ndarray]) - 待查找的某个 Token 或 Token 列表。如果 Token 不存在, 则返回-1。

返回:

Union[int, list[int]], 指定 Token 对应的 ID。

异常:

- **TypeError** - 当 *tokens* 不为 Union[str, list[str], numpy.ndarray] 类型。

样例:

```
>>> import mindspore.dataset.text as text
>>> vocab = text.Vocab.from_list(["w1", "w2", "w3"], special_tokens=["<unk>"], special_
↳first=True)
>>> ids = vocab.tokens_to_ids(["w1", "w3"])
>>> print(ids)
[1, 3]
```

vocab ()

获取 Token 与其 ID 之间映射关系的字典。

返回:

dict[str, int], Token 与 ID 之间映射关系的字典。

样例:

```
>>> import mindspore.dataset.text as text
>>> vocab = text.Vocab.from_list(["word_1", "word_2", "word_3", "word_4"])
>>> vocabory_dict = vocab.vocab()
>>> print(sorted(vocabory_dict.items()))
[('word_1', 0), ('word_2', 1), ('word_3', 2), ('word_4', 3)]
```

`mindspore.dataset.text.to_bytes`

`mindspore.dataset.text.to_bytes(array, encoding='utf8')`

基于 *encoding* 字符集对每个元素进行编码, 将 *string* 的 NumPy 数组转换为 *bytes* 的数组。

参数:

- **array** (numpy.ndarray) - 表示 *string* 类型的数组, 代表字符串。
- **encoding** (str) - 表示用于编码的字符集。默认值: 'utf8'。

返回:

numpy.ndarray, 表示 *bytes* 的 NumPy 数组。

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> data = np.array([[ "1", "2", "3"]], dtype=np.str_)
>>> dataset = ds.NumpySlicesDataset(data, column_names=["text"])
>>> result = []
>>> for item in dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     result.append(text.to_bytes(item["text"]))
>>> print(result)
[array([b'1', b'2', b'3'], dtype='<S1')]
```

`mindspore.dataset.text.to_str`

`mindspore.dataset.text.to_str(array, encoding='utf8')`

基于 *encoding* 字符集对每个元素进行解码, 借此将 *bytes* 的 NumPy 数组转换为 *string* 的数组。

参数:

- **array** (numpy.ndarray) - 表示 *bytes* 类型的数组, 代表字符串。
- **encoding** (str) - 表示用于解码的字符集。默认值: 'utf8'。

返回:

numpy.ndarray, 表示 *str* 的 NumPy 数组。

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.text as text
>>>
>>> data = np.array([[ "1", "2", "3"]], dtype=np.bytes_)
```

(下页继续)

(续上页)

```
>>> dataset = ds.NumpySlicesDataset(data, column_names=["text"])
>>> result = []
>>> for item in dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     result.append(text.to_str(item["text"]))
>>> print(result)
[array(['1', '2', '3'], dtype='<U1')]
```

13.4 音频

此模块用于音频数据增强，包括 *transforms* 和 *utils* 两个子模块。*transforms* 是一个高性能音频数据增强模块，支持常见的音频数据增强操作。*utils* 提供了一些音频处理的工具方法。

API 样例中常用的导入模块如下：

```
import mindspore.dataset as ds
import mindspore.dataset.audio as audio
from mindspore.dataset.audio import utils
```

常用数据处理术语说明如下：

- *TensorOperation*，所有 C++ 实现的数据处理操作的基类。
- *AudioTensorOperation*，所有音频数据处理操作的基类，派生自 *TensorOperation*。

数据增强操作可以放入数据处理 *Pipeline* 中执行，也可以 *Eager* 模式执行：

- *Pipeline* 模式用于流式处理大型数据集，示例可参考 [数据处理 Pipeline 介绍](#)。
- *Eager* 模式用于函数调用方式处理样本，示例可参考 [轻量化数据处理](#)。

13.4.1 样例库

快速上手使用音频变换的 API，跳转参考 [音频变换样例库](#)。此指南中展示了多个变换 API 的用法，以及输入输出结果。

13.4.2 变换

<code>mindspore.dataset.audio.AllpassBiquad</code>	给音频波形施加双极点全通滤波器，其中心频率和带宽由入参指定。
<code>mindspore.dataset.audio.AmplitudeToDB</code>	将输入音频从振幅/功率标度转换为分贝标度。
<code>mindspore.dataset.audio.Angle</code>	计算复数序列的角度。
<code>mindspore.dataset.audio.BandBiquad</code>	给音频波形施加双极点带通滤波器。
<code>mindspore.dataset.audio.BandpassBiquad</code>	给音频波形施加双极点巴特沃斯（Butterworth）带通滤波器。
<code>mindspore.dataset.audio.BandrejectBiquad</code>	给音频波形施加双极点巴特沃斯（Butterworth）带阻滤波器。
<code>mindspore.dataset.audio.BassBiquad</code>	给音频波形施加低音控制效果，即双极点低频搁架滤波器。
<code>mindspore.dataset.audio.Biquad</code>	给音频波形施加双二阶滤波器。
<code>mindspore.dataset.audio.ComplexNorm</code>	计算复数序列的范数。
<code>mindspore.dataset.audio.ComputeDeltas</code>	计算频谱的 delta 系数，也叫差分系数。

下页继续

表 7 - 续上页

<code>mindspore.dataset.audio.Contrast</code>	给音频波形施加对比度增强效果。
<code>mindspore.dataset.audio.DBToAmplitude</code>	将音频波形从分贝转换为功率或振幅。
<code>mindspore.dataset.audio.DCShift</code>	对输入音频波形施加直流移位。
<code>mindspore.dataset.audio.DeemphBiquad</code>	给音频波形施加 CD (IEC 60908) 去重音 (一种高音衰减搁置滤波器) 效果。
<code>mindspore.dataset.audio.DetectPitchFrequency</code>	检测音调频率。
<code>mindspore.dataset.audio.Dither</code>	通过消除非线性截断失真, 来抖动增加存储在特定深度音频的动态感知范围。
<code>mindspore.dataset.audio.EqualizerBiquad</code>	给音频波形施加双二次均衡器滤波器。
<code>mindspore.dataset.audio.Fade</code>	向波形添加淡入和/或淡出。
<code>mindspore.dataset.audio.Filtfilt</code>	对音频波形施加正反向 IIR 滤波。
<code>mindspore.dataset.audio.Flanger</code>	给音频施加镶边效果。
<code>mindspore.dataset.audio.FrequencyMasking</code>	给音频波形施加频域掩码。
<code>mindspore.dataset.audio.Gain</code>	放大或衰减整个音频波形。
<code>mindspore.dataset.audio.GriffinLim</code>	使用 Griffin-Lim 算法从线性幅度频谱图中计算信号波形。
<code>mindspore.dataset.audio.HighpassBiquad</code>	给音频波形上施加双二阶高通滤波器。
<code>mindspore.dataset.audio.InverseMelScale</code>	使用转换矩阵从梅尔频率 STFT 求解普通频率的 STFT。
<code>mindspore.dataset.audio.InverseSpectrogram</code>	计算输入频谱的反向频谱, 以恢复原始音频信号。
<code>mindspore.dataset.audio.LFCC</code>	计算音频信号的线性频率倒谱系数。
<code>mindspore.dataset.audio.LFilter</code>	根据指定的差分方程施加 IIR 滤波器。
<code>mindspore.dataset.audio.LowpassBiquad</code>	给音频波形施加双极点低通滤波器。
<code>mindspore.dataset.audio.Magphase</code>	将 <code>shape</code> 为 <code>(..., 2)</code> 的复值光谱图分离, 输出幅度和相位。
<code>mindspore.dataset.audio.MaskAlongAxis</code>	对音频波形应用掩码。
<code>mindspore.dataset.audio.MaskAlongAxisIID</code>	对音频波形沿 <code>axis</code> 轴应用掩码。
<code>mindspore.dataset.audio.MelScale</code>	将普通 STFT 转换为梅尔尺度的 STFT。
<code>mindspore.dataset.audio.MelSpectrogram</code>	计算原始音频信号的梅尔频谱。
<code>mindspore.dataset.audio.MFCC</code>	计算音频信号的梅尔频率倒谱系数。
<code>mindspore.dataset.audio.MuLawDecoding</code>	解码 <code>mu-law</code> 编码的信号, 参考 <code>mu-law 算法</code> 。
<code>mindspore.dataset.audio.MuLawEncoding</code>	基于 <code>mu-law</code> 压缩的信号编码。
<code>mindspore.dataset.audio.Overdrive</code>	给音频波形施加过载效果。
<code>mindspore.dataset.audio.Phaser</code>	给音频波形施加相位效果。
<code>mindspore.dataset.audio.PhaseVocoder</code>	对给定的 STFT 频谱, 在不改变音高的情况下以一定比率进行加速。
<code>mindspore.dataset.audio.Resample</code>	将信号从一个频率重采样至另一个频率。
<code>mindspore.dataset.audio.PitchShift</code>	将波形的音调移动 <code>n_steps</code> 步长。
<code>mindspore.dataset.audio.RiaaBiquad</code>	对输入音频波形施加 RIAA 均衡。
<code>mindspore.dataset.audio.SlidingWindowCmn</code>	对每个话语应用滑动窗口倒谱均值 (和可选方差) 归一化。
<code>mindspore.dataset.audio.SpectralCentroid</code>	计算每个通道沿时间轴的频谱中心。
<code>mindspore.dataset.audio.Spectrogram</code>	从音频信号创建其频谱。
<code>mindspore.dataset.audio.TimeMasking</code>	给音频波形施加时域掩码。
<code>mindspore.dataset.audio.TimeStretch</code>	以给定的比例拉伸音频短时傅里叶 (Short Time Fourier Transform, STFT) 频谱的时域, 但不改变音频的音高。
<code>mindspore.dataset.audio.TrebleBiquad</code>	给音频波形施加高音音调控制效果。
<code>mindspore.dataset.audio.Vad</code>	语音活动检测器。
<code>mindspore.dataset.audio.Vol</code>	调整波形的音量。

mindspore.dataset.audio.AllpassBiquad

class mindspore.dataset.audio.AllpassBiquad (sample_rate, central_freq, Q=0.707)

给音频波形施加双极点全通滤波器，其中心频率和带宽由入参指定。

全通滤波器能够改变音频频率与相位的关系，而不改变频率与幅度的关系。其系统函数为：

$$H(s) = \frac{s^2 - \frac{s}{Q} + 1}{s^2 + \frac{s}{Q} + 1}$$

接口实现方式类似于 SoX 库。

说明：待处理音频 shape 需为 <..., time>。

参数：

- **sample_rate** (int) - 采样频率（单位：Hz），不能为零。
- **central_freq** (float) - 中心频率（单位：Hz）。
- **Q** (float, 可选) - 品质因子，能够反映带宽与采样频率和中心频率的关系，取值范围为 (0, 1]。默认值：0.707。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *central_freq* 的类型不为 float。
- **TypeError** - 当 *Q* 的类型不为 float。
- **ValueError** - 当 *Q* 取值不在 (0, 1] 范围内。
- **RuntimeError** - 当输入音频的 shape 不为 <..., time>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode.
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.AllpassBiquad(44100, 200.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=["audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
```

(下页继续)

(续上页)

```
>>> output = audio.AllpassBiquad(44100, 200.0)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.AmplitudeToDB

class mindspore.dataset.audio.**AmplitudeToDB** (*stype=ScaleType.POWER, ref_value=1.0, amin=1e-10, top_db=80.0*)

将输入音频从振幅/功率标度转换为分贝标度。

说明：待处理音频 shape 需为 <..., freq, time>。

参数：

- **stype** (*ScaleType*, 可选) - 输入音频的原始标度，取值可为 *ScaleType.MAGNITUDE* 或 *ScaleType.POWER*。默认值：*ScaleType.POWER*。
- **ref_value** (float, 可选) - 系数参考值。默认值：1.0，用于计算分贝系数 *db_multiplier*，公式为 $db\_multiplier = \log_{10}(\max(ref\_value, amin))$ 。
- **amin** (float, 可选) - 波形取值下界，低于该值的波形将会被裁切，取值必须大于 0。默认值：1e-10。
- **top_db** (float, 可选) - 最小截止分贝值，取值为非负数。默认值：80.0。

异常：

- **TypeError** - 当 *stype* 的类型不为 *mindspore.dataset.audio.ScaleType*。
- **TypeError** - 当 *ref_value* 的类型不为 float。
- **ValueError** - 当 *ref_value* 不为正数。
- **TypeError** - 当 *amin* 的类型不为 float。
- **ValueError** - 当 *amin* 不为正数。
- **TypeError** - 当 *top_db* 的类型不为 float。
- **ValueError** - 当 *top_db* 不为正数。
- **RuntimeError** - 当输入音频的 shape 不为 <..., freq, time>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 400 // 2 + 1, 30]) # 5 samples
```

(下页继续)

(续上页)

```

>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.AmplitudeToDB(stype=audio.ScaleType.POWER)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(201, 30) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([400 // 2 + 1, 30]) # 1 sample
>>> output = audio.AmplitudeToDB(stype=audio.ScaleType.POWER)(waveform)
>>> print(output.shape, output.dtype)
(201, 30) float64

```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Angle

class mindspore.dataset.audio.Angle

计算复数序列的角度。

说明：待处理音频 shape 需为 $\langle \dots, \text{complex}=2 \rangle$ 。第零维代表实部，第一维代表虚部。

异常：

- **RuntimeError** - 当输入音频的 shape 不为 $\langle \dots, \text{complex}=2 \rangle$ 。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Angle()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 2]) # 1 sample
>>> output = audio.Angle()(waveform)

```

(下页继续)

(续上页)

```
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.BandBiquad

class mindspore.dataset.audio.**BandBiquad** (*sample_rate, central_freq, Q=0.707, noise=False*)

给音频波形施加双极点带通滤波器。

带通滤波器的频率响应在中心频率附近呈对数下降。下降的斜率由带宽决定。频带两端处输出音频的幅度将是原始幅度的一半。

接口实现方式类似于 [SoX 库](#)。

说明：待处理音频 shape 需为 <..., time>。

参数：

- **sample_rate** (int) - 采样频率（单位：Hz），不能为零。
- **central_freq** (float) - 中心频率（单位：Hz）。
- **Q** (float, 可选) - 品质因子，能够反映带宽与采样频率和中心频率的关系，取值范围为 (0, 1]。默认值：0.707。
- **noise** (bool, 可选) - 若为 True，则使用非音调音频（如打击乐）模式；若为 False，则使用音调音频（如语音、歌曲或器乐）模式。默认值：False。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *central_freq* 的类型不为 float。
- **TypeError** - 当 *Q* 的类型不为 float。
- **ValueError** - 当 *Q* 取值不在 (0, 1] 范围内。
- **TypeError** - 当 *noise* 的类型不为 bool。
- **RuntimeError** - 当输入音频的 shape 不为 <..., time>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
```

(下页继续)

(续上页)

```

>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.BandBiquad(44100, 200.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.BandBiquad(44100, 200.0)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64

```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.BandpassBiquad

class mindspore.dataset.audio.**BandpassBiquad**(*sample_rate, central_freq, Q=0.707, const_skirt_gain=False*)

给音频波形施加双极点巴特沃斯（Butterworth）带通滤波器。

巴特沃斯滤波器的特点是通频带内的频率响应曲线最大限度平坦，没有纹波，而在阻频带则逐渐下降为零。

巴特沃斯带通滤波器的系统函数为：

$$H(s) = \begin{cases} \frac{s}{s^2 + \frac{s}{Q} + 1}, & \text{if const\_skirt\_gain=True;} \\ \frac{Q}{s^2 + \frac{s}{Q} + 1}, & \text{if const\_skirt\_gain=False.} \end{cases}$$

接口实现方式类似于 SoX 库。

说明：待处理音频 shape 需为 <..., time>。

参数：

- **sample_rate** (int) - 采样频率（单位：Hz），不能为零。
- **central_freq** (float) - 中心频率（单位：Hz）。
- **Q** (float, 可选) - 品质因子，能够反映带宽与采样频率和中心频率的关系，取值范围为 (0, 1]。默认值：0.707。
- **const_skirt_gain** (bool, 可选) - 若为 True，则使用恒定裙边增益（峰值增益为 Q）；若为 False，则使用恒定的 0dB 峰值增益。默认值：False。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *central_freq* 的类型不为 float。
- **TypeError** - 当 *Q* 的类型不为 float。

- **ValueError** - 当 Q 取值不在 (0, 1] 范围内。
- **TypeError** - 当 `const_skirt_gain` 的类型不为 bool。
- **RuntimeError** - 当输入音频的 shape 不为 $\langle \cdots, \text{time} \rangle$ 。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.BandpassBiquad(44100, 200.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.BandpassBiquad(44100, 200.0)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.BandrejectBiquad`

class `mindspore.dataset.audio.BandrejectBiquad` (*sample_rate*, *central_freq*, $Q=0.707$)

给音频波形施加双极点巴特沃斯（Butterworth）带阻滤波器。

巴特沃斯滤波器的特点是通频带内的频率响应曲线最大限度平坦，没有纹波，而在阻频带则逐渐下降为零。

巴特沃斯带阻滤波器的系统函数为：

$$H(s) = \frac{s^2 + 1}{s^2 + \frac{s}{Q} + 1}$$

接口实现方式类似于 `SoX` 库。

说明：待处理音频 shape 需为 $\langle \cdots, \text{time} \rangle$ 。

参数：

- **sample_rate** (int) - 采样频率（单位：Hz），不能为零。
- **central_freq** (float) - 中心频率（单位：Hz）。

- **Q** (float, 可选) - 品质因子，能够反映带宽与采样频率和中心频率的关系，取值范围为 (0, 1]。默认值：0.707。

异常：

- **TypeError** - 当 `sample_rate` 的类型不为 `int`。
- **ValueError** - 当 `sample_rate` 的数值为 0。
- **TypeError** - 当 `central_freq` 的类型不为 `float`。
- **TypeError** - 当 `Q` 的类型不为 `float`。
- **ValueError** - 当 `Q` 取值不在 (0, 1] 范围内。
- **RuntimeError** - 当输入音频的 `shape` 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.BandrejectBiquad(44100, 200.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.BandrejectBiquad(44100, 200.0)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.BassBiquad`

class `mindspore.dataset.audio.BassBiquad` (`sample_rate`, `gain`, `central_freq=100.0`, `Q=0.707`)

给音频波形施加低音控制效果，即双极点低频搁架滤波器。

低频搁架滤波器能够通过所有频率，但将低于搁架的频率提升或衰减指定量。其系统函数为：

$$H(s) = A \frac{s^2 + \frac{\sqrt{A}}{Q}s + A}{As^2 + \frac{\sqrt{A}}{Q}s + 1}$$

接口实现方式类似于 `SoX` 库。

说明：待处理音频 shape 需为 $\langle \dots, \text{time} \rangle$ 。

参数：

- **sample_rate** (int) - 采样频率（单位：Hz），不能为零。
- **gain** (float) - 期望提升（或衰减）的音频增益（单位：dB）。
- **central_freq** (float, 可选) - 中心频率（单位：Hz）。默认值：100.0。
- **Q** (float, 可选) - 品质因子，能够反映带宽与采样频率和中心频率的关系，取值范围为 (0, 1]。默认值：0.707。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *gain* 的类型不为 float。
- **TypeError** - 当 *central_freq* 的类型不为 float。
- **TypeError** - 当 *Q* 的类型不为 float。
- **ValueError** - 当 *Q* 取值不在 (0, 1] 范围内。
- **RuntimeError** - 当输入音频的 shape 不为 $\langle \dots, \text{time} \rangle$ 。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.BassBiquad(44100, 100.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.BassBiquad(44100, 200.0)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.Biquad`

class `mindspore.dataset.audio.Biquad` (*b0, b1, b2, a0, a1, a2*)

给音频波形施加双二阶滤波器。

具体的数学公式与参数详见 [数字双二阶滤波器](#)。

参数：

- **b0** (float) - 当前输入的分子系数， $x[n]$ 。
- **b1** (float) - 一个时间间隔前输入的分子系数 $x[n-1]$ 。
- **b2** (float) - 两个时间间隔前输入的分子系数 $x[n-2]$ 。
- **a0** (float) - 当前输出 $y[n]$ 的分母系数，该值不能为 0，通常为 1。
- **a1** (float) - 当前输出 $y[n-1]$ 的分母系数。
- **a2** (float) - 当前输出 $y[n-2]$ 的分母系数。

异常：

- **TypeError** - 如果 *b0* 不是 float 类型。
- **TypeError** - 如果 *b1* 不是 float 类型。
- **TypeError** - 如果 *b2* 不是 float 类型。
- **TypeError** - 如果 *a0* 不是 float 类型。
- **TypeError** - 如果 *a1* 不是 float 类型。
- **TypeError** - 如果 *a2* 不是 float 类型。
- **ValueError** - 如果 *a0* 为 0。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Biquad(0.01, 0.02, 0.13, 1, 0.12, 0.3)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.Biquad(0.01, 0.02, 0.13, 1, 0.12, 0.3)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.ComplexNorm`

class `mindspore.dataset.audio.ComplexNorm` (*power=1.0*)

计算复数序列的范数。

说明：待处理音频 `shape` 需为 `<..., complex=2>`。第零维代表实部，第一维代表虚部。

参数：

- **power** (float, 可选) - 范数的幂，取值必须非负。默认值：1.0。

异常：

- **TypeError** - 当 *power* 的类型不为 float。
- **ValueError** - 当 *power* 为负数。
- **RuntimeError** - 当输入音频的 `shape` 不为 `<..., complex=2>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.ComplexNorm()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 2]) # 1 samples
>>> output = audio.ComplexNorm()(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.ComputeDeltas

class mindspore.dataset.audio.**ComputeDeltas** (win_length=5, pad_mode=BorderType.EDGE)

计算频谱的 delta 系数，也叫差分系数。

delta 系数能够帮助理解功率谱中的动态信息。能够通过下列公式进行计算。

$$d_t = \frac{\sum_{n=1}^N n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^N n^2}$$

其中， d_t 是 t 时刻的 delta 值， c_t 是 t 时刻的频谱系数， N 是 $(\text{win\_length} - 1) // 2$ 。

参数：

- **win_length** (int, 可选) - 用于计算 delta 值的窗口长度，必须不小于 3。默认值：5。
- **pad_mode** (*BorderType*, 可选) - 边界填充模式，可为 *BorderType.CONSTANT*、*BorderType.EDGE*、*BorderType.REFLECT* 或 *BorderType.SYMMETRIC*。默认值：*BorderType.EDGE*。
 - *BorderType.CONSTANT*，使用常量值填充。
 - *BorderType.EDGE*，使用各边的边界像素值填充。
 - *BorderType.REFLECT*，以各边的边界为轴进行镜像填充，忽略边界像素值。例如，向 [1, 2, 3, 4] 的两边分别填充 2 个元素，结果为 [3, 2, 1, 2, 3, 4, 3, 2]。
 - *BorderType.SYMMETRIC*，以各边的边界为轴进行对称填充，包括边界像素值。例如，向 [1, 2, 3, 4] 的两边分别填充 2 个元素，结果为 [2, 1, 1, 2, 3, 4, 4, 3]。

异常：

- **TypeError** - 当 *win_length* 的类型不为 int。
- **ValueError** - 当 *win_length* 小于 3。
- **TypeError** - 当 *pad_mode* 的类型不为 *mindspore.dataset.audio.BorderType*。
- **RuntimeError** - 当输入音频的 shape 不为 $\langle \dots, \text{freq}, \text{time} \rangle$ 。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 400 // 2 + 1, 30]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.ComputeDeltas(win_length=7, pad_mode=audio.BorderType.EDGE)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(201, 30) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([400 // 2 + 1, 30]) # 1 sample
>>> output = audio.ComputeDeltas(win_length=7, pad_mode=audio.BorderType.EDGE)(waveform)
```

(下页继续)

(续上页)

```
>>> print(output.shape, output.dtype)
(201, 30) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Contrast

class mindspore.dataset.audio.**Contrast** (*enhancement_amount=75.0*)

给音频波形施加对比度增强效果。

与音频压缩相比，该效果通过修改音频信号使其听起来更响亮。

接口实现方式类似于 [SoX 库](#)。

说明：待处理音频 shape 需为 `<..., time>`。

参数：

- **enhancement_amount** (float, 可选) - 控制音频增益的量，取值范围为 [0,100]。默认值：75.0。请注意当 *enhancement_amount* 等于 0 时，对比度增强效果仍然会很显著。

异常：

- **TypeError** - 当 *enhancement_amount* 的类型不为 float。
- **ValueError** - 当 *enhancement_amount* 取值不在 [0, 100] 范围内。
- **RuntimeError** - 当输入音频的 shape 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Contrast()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.Contrast()(waveform)
```

(下页继续)

(续上页)

```
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.DBToAmplitude

class mindspore.dataset.audio.DBToAmplitude(*ref, power*)

将音频波形从分贝转换为功率或振幅。

参数：

- **ref** (float) - 输出波形的缩放系数。
- **power** (float) - 如果 *power* 等于 1，则将分贝值转为功率；如果为 0.5，则将分贝值转为振幅。

异常：

- **TypeError** - 如果 *ref* 不是 float 类型。
- **TypeError** - 如果 *power* 不是 float 类型。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.DBToAmplitude(0.5, 0.5)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.DBToAmplitude(0.5, 0.5)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.DCShift

class mindspore.dataset.audio.DCShift (shift, limiter_gain=None)

对输入音频波形施加直流移位。可以从音频中删除直流偏移 (DC Offset)。

参数：

- **shift** (float) - 音频的移位量，值必须在 [-2.0, 2.0] 范围内。
- **limiter_gain** (float, 可选) - 防止截断，仅在波峰生效。值应远小于 1，如 0.05 或 0.02。默认值：None，将被设置为 *shift*。

异常：

- **TypeError** - 如果 *shift* 不是 float 类型。
- **ValueError** - 如果 *shift* 不在 [-2.0, 2.0] 范围内。
- **TypeError** - 如果 *limiter_gain* 不是 float 类型。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.DCShift(0.5, 0.02)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.DCShift(0.5, 0.02)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.DeemphBiquad`

class `mindspore.dataset.audio.DeemphBiquad` (*sample_rate*)

给音频波形施加 CD (IEC 60908) 去重音 (一种高音衰减搁置滤波器) 效果。

接口实现方式类似于 `SoX` 库。

参数：

- **sample_rate** (int) - 波形的采样频率，只能为 44100 或 48000 (Hz)。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 不为 44100 或 48000。
- **RuntimeError** - 当输入音频的 shape 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 8]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.DeemphBiquad(44100)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(8,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([8]) # 1 sample
>>> output = audio.DeemphBiquad(44100)(waveform)
>>> print(output.shape, output.dtype)
(8,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.DetectPitchFrequency`

class `mindspore.dataset.audio.DetectPitchFrequency` (*sample_rate*, *frame_time*=0.01, *win_length*=30, *freq_low*=85, *freq_high*=3400)

检测音调频率。基于归一化互相关函数和中位平滑来实现。

参数：

- **sample_rate** (int) - 波形的采样频率，如 44100 (单位：Hz)，值不能为 0。
- **frame_time** (float, 可选) - 帧的持续时间，值必须大于零。默认值：0.01。

- **win_length** (int, 可选) - 中位平滑的窗口长度（以帧数为单位），该值必须大于零。默认值：30。
- **freq_low** (int, 可选) - 可检测的最低频率（Hz），该值必须大于零。默认值：85。
- **freq_high** (int, 可选) - 可检测的最高频率（Hz），该值必须大于零。默认值：3400。

异常：

- **TypeError** - 如果 *sample_rate* 不是 int 类型。
- **ValueError** - 如果 *sample_rate* 为 0。
- **TypeError** - 如果 *frame_time* 不是 float 类型。
- **ValueError** - 如果 *frame_time* 不为正数。
- **TypeError** - 如果 *win_length* 不是 int 类型。
- **ValueError** - 如果 *win_length* 不为正数。
- **TypeError** - 如果 *freq_low* 不是 int 类型。
- **ValueError** - 如果 *freq_low* 不为正数。
- **TypeError** - 如果 *freq_high* 不是 int 类型。
- **ValueError** - 如果 *freq_high* 不为正数。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.DetectPitchFrequency(30, 0.1, 3, 5, 25)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(5,) float32
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.DetectPitchFrequency(30, 0.1, 3, 5, 25)(waveform)
>>> print(output.shape, output.dtype)
(5,) float32
```

教程样例：

- [音频变换样例库](#)

`mindspore.dataset.audio.Dither`

class `mindspore.dataset.audio.Dither` (*density_function*=*DensityFunction.TPDF*, *noise_shaping*=*False*)

通过消除非线性截断失真，来抖动增加存储在特定位置的音频的动态感知范围。

参数：

- **density_function** (*DensityFunction*, 可选) - 连续随机变量的密度函数。可为 `DensityFunction.TPDF` (三角概率密度函数)、`DensityFunction.RPDF` (矩形概率密度函数) 或 `DensityFunction.GPDF` (高斯概率密度函数)。默认值: `DensityFunction.TPDF`。
- **noise_shaping** (bool, 可选) - 是否通过滤波操作，来消除频谱能量的量化误差。默认值: `False`。

异常：

- **TypeError** - 当 *density_function* 的类型不为 `mindspore.dataset.audio.DensityFunction`。
- **TypeError** - 当 *noise_shaping* 的类型不为 `bool`。
- **RuntimeError** - 当输入音频的 *shape* 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Dither()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.Dither()(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.EqualizerBiquad

class mindspore.dataset.audio.**EqualizerBiquad** (*sample_rate*, *center_freq*, *gain*, *Q*=0.707)

给音频波形施加双二次均衡器滤波器。

接口实现方式类似于 [SoX 库](#)。

参数：

- **sample_rate** (int) - 波形的采样频率，如 44100 (单位：Hz)，值不能为 0。
- **center_freq** (float) - 中心频率（单位：Hz）。
- **gain** (float) - 期望提升（或衰减）的音频增益（单位：dB）。
- **Q** (float, 可选) - 品质因子，能够反映带宽与采样频率和中心频率的关系，取值范围为 (0, 1]。默认值：0.707。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *center_freq* 的类型不为 float。
- **TypeError** - 当 *gain* 的类型不为 float。
- **TypeError** - 当 *Q* 的类型不为 float。
- **ValueError** - 当 *Q* 取值不在 (0, 1] 范围内。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.EqualizerBiquad(44100, 1500, 5.5, 0.7)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.EqualizerBiquad(44100, 1500, 5.5, 0.7)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- [音频变换样例库](#)

`mindspore.dataset.audio.Fade`

class `mindspore.dataset.audio.Fade` (*fade_in_len=0, fade_out_len=0, fade_shape=FadeShape.LINEAR*)
向波形添加淡入和/或淡出。

参数：

- **fade_in_len** (int, 可选) - 淡入长度（时间帧），必须是非负。默认值：0。
- **fade_out_len** (int, 可选) - 淡出长度（时间帧），必须是非负。默认值：0。
- **fade_shape** (*FadeShape*, 可选) - 淡入淡出形状，可以选择 `FadeShape` 提供的模式。默认值：`FadeShape.LINEAR`。
 - `FadeShape.QUARTER_SINE`，表示淡入淡出形状为四分之一正弦模式。
 - `FadeShape.HALF_SINE`，表示淡入形状为半正弦模式。
 - `FadeShape.LINEAR`，表示淡入淡出形状为线性模式。
 - `FadeShape.LOGARITHMIC`，表示淡入淡出形状为对数模式。
 - `FadeShape.EXPONENTIAL`，表示淡入淡出形状为指数模式。

异常：

- **RuntimeError** - 如果 *fade_in_len* 超过音频波形长度。
- **RuntimeError** - 如果 *fade_out_len* 超过音频波形长度。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Fade(fade_in_len=3, fade_out_len=2, fade_shape=audio.FadeShape.
↳LINEAR)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↳"audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.Fade(fade_in_len=3, fade_out_len=2, fade_shape=audio.FadeShape.
↳LINEAR)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Filtfilt

class mindspore.dataset.audio.**Filtfilt** (*a_coeffs*, *b_coeffs*, *clamp=True*)

对音频波形施加正反向 IIR 滤波。

参数：

- **a_coeffs** (Sequence[float]) - 不同维度的差分方程分母系数。低维的延迟系数在前，例如 [a0, a1, a2, ...]。序列长度必须与 *b_coeffs* 相同（根据需要填充 0 值）。
- **b_coeffs** (Sequence[float]) - 不同维度的差分方程分子系数。低维的延迟系数在前，例如 [b0, b1, b2, ...]。序列长度必须与 *a_coeffs* 相同（根据需要填充 0 值）。
- **clamp** (bool, 可选) - 如果为 True，将输出信号截断在 [-1, 1] 范围内。默认值：True。

异常：

- **TypeError** - 当 *a_coeffs* 的类型不为 Sequence[float]。
- **TypeError** - 当 *b_coeffs* 的类型不为 Sequence[float]。
- **ValueError** - 当 *a_coeffs* 与 *b_coeffs* 维度不同。
- **TypeError** - 当 *clamp* 的类型不为 bool。
- **RuntimeError** - 当输入音频的 shape 不为 <..., time>。

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Filtfilt(a_coeffs=[0.1, 0.2, 0.3], b_coeffs=[0.1, 0.2, 0.3])]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.Filtfilt(a_coeffs=[0.1, 0.2, 0.3], b_coeffs=[0.1, 0.2, 0.3])(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Flanger

class mindspore.dataset.audio.**Flanger** (*sample_rate*, *delay*=0.0, *depth*=2.0, *regen*=0.0, *width*=71.0, *speed*=0.5, *phase*=25.0, *modulation*=*Modulation.SINUSOIDAL*, *interpolation*=*Interpolation.LINEAR*)

给音频施加镶边效果。

接口实现方式类似于 [SoX 库](#)。

参数：

- **sample_rate** (int) - 波形采样频率，例如 44100 (Hz)。
- **delay** (float, 可选) - 期望的延迟时间，单位为毫秒，取值范围为 [0, 30]。默认值：0.0。
- **depth** (float, 可选) - 期望的延迟深度，单位为毫秒，取值范围为 [0, 10]。默认值：2.0。
- **regen** (float, 可选) - 期望的反馈增益，单位为 dB，取值范围为 [-95, 95]。默认值：0.0。
- **width** (float, 可选) - 期望的延迟增益，单位为 dB，取值范围为 [0, 100]。默认值：71.0。
- **speed** (float, 可选) - 调制速度，单位为 Hz，取值范围为 [0.1, 10]。默认值：0.5。
- **phase** (float, 可选) - 各通道的相位偏移百分比，取值范围为 [0, 100]。默认值：25.0。
- **modulation** (*Modulation*, 可选) - 调制方法，可为 *Modulation.SINUSOIDAL* 或 *Modulation.TRIANGULAR*。默认值：*Modulation.SINUSOIDAL*。
- **interpolation** (*Interpolation*, 可选) - 插值方法，可为 *Interpolation.LINEAR* 或 *Interpolation.QUADRATIC*。默认值：*Interpolation.LINEAR*。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 为零。
- **TypeError** - 当 *delay* 的类型不为 float。
- **ValueError** - 当 *delay* 取值不在 [0, 30] 范围内。
- **TypeError** - 当 *depth* 的类型不为 float。
- **ValueError** - 当 *depth* 取值不在 [0, 10] 范围内。
- **TypeError** - 当 *regen* 的类型不为 float。
- **ValueError** - 当 *regen* 取值不在 [-95, 95] 范围内。
- **TypeError** - 当 *width* 的类型不为 float。
- **ValueError** - 当 *width* 取值不在 [0, 100] 范围内。
- **TypeError** - 当 *speed* 的类型不为 float。
- **ValueError** - 当 *speed* 取值不在 [0.1, 10] 范围内。
- **TypeError** - 当 *phase* 的类型不为 float。
- **ValueError** - 当 *phase* 取值不在 [0, 100] 范围内。
- **TypeError** - 当 *modulation* 的类型不为 *mindspore.dataset.audio.Modulation*。
- **TypeError** - 当 *interpolation* 的类型不为 *mindspore.dataset.audio.Interpolation*。
- **RuntimeError** - 当输入音频的 shape 不为 <..., channel, time>。

支持平台：

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 4, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Flanger(44100)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(4, 16) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([4, 16]) # 1 sample
>>> output = audio.Flanger(44100)(waveform)
>>> print(output.shape, output.dtype)
(4, 16) float64
```

教程样例:

- 音频变换样例库

mindspore.dataset.audio.FrequencyMasking

```
class mindspore.dataset.audio.FrequencyMasking (iid_masks=False, freq_mask_param=0, mask_start=0,
                                                mask_value=0.0)
```

给音频波形施加频域掩码。

说明: 待处理音频 shape 需为 <..., freq, time>。

参数:

- **iid_masks** (bool, 可选) - 是否施加随机掩码。默认值: False。
- **freq_mask_param** (int, 可选) - 当 **iid_masks** 为 True 时, 掩码长度将从 [0, freq_mask_param] 中均匀采样; 当 **iid_masks** 为 False 时, 直接使用该值作为掩码长度。取值范围为 [0, freq_length], 其中 *freq_length* 为音频波形在频域的长度。默认值: 0。
- **mask_start** (int, 可选) - 添加掩码的起始位置, 只有当 **iid_masks** 为 True 时, 该值才会生效。取值范围为 [0, freq_length - frequency_mask_param], 其中 *freq_length* 为音频波形在频域的长度。默认值: 0。
- **mask_value** (float, 可选) - 掩码填充值。默认值: 0.0。

异常:

- **TypeError** - 当 *iid_masks* 的类型不为 bool。
- **TypeError** - 当 *freq_mask_param* 的类型不为 int。
- **ValueError** - 当 *freq_mask_param* 大于音频频域长度。

- **TypeError** - 当 `mask_start` 的类型不为 `int`。
- **ValueError** - 当 `mask_start` 为负数。
- **TypeError** - 当 `mask_value` 的类型不为 `float`。
- **ValueError** - 当 `mask_value` 为负数。
- **RuntimeError** - 当输入音频的 `shape` 不为 `<..., freq, time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.FrequencyMasking(iid_masks=True, freq_mask_param=1)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16, 2) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 2]) # 1 sample
>>> output = audio.FrequencyMasking(iid_masks=True, freq_mask_param=1)(waveform)
>>> print(output.shape, output.dtype)
(16, 2) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Gain

class mindspore.dataset.audio.Gain (gain_db=1.0)

放大或衰减整个音频波形。

参数：

- **gain_db** (float) - 增益调整，单位为分贝 (dB)。默认值：1.0。

异常：

- **TypeError** - 当 `gain_db` 的类型不为 `float`。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 8]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Gain(1.2)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(8,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([8]) # 1 sample
>>> output = audio.Gain(1.2)(waveform)
>>> print(output.shape, output.dtype)
(8,) float64

```

教程样例:

- 音频变换样例库

`mindspore.dataset.audio.GriffinLim`

```

class mindspore.dataset.audio.GriffinLim (n_fft=400, n_iter=32, win_length=None, hop_length=None,
                                          window_type=WindowType.HANN, power=2.0, momentum=0.99,
                                          length=None, rand_init=True)

```

使用 Griffin-Lim 算法从线性幅度频谱图中计算信号波形。

有关 Griffin-Lim 算法更多的描述, 详见论文 [A fast Griffin-Lim algorithm](#) 与 [Signal estimation from modified short-time Fourier transform](#)。

参数:

- **n_fft** (int, 可选) - FFT 的长度。默认值: 400。
- **n_iter** (int, 可选) - 相位恢复的迭代次数。默认值: 32。
- **win_length** (int, 可选) - Griffin-Lim 的窗口大小。默认值: None, 将设置为 `n_fft` 的值。
- **hop_length** (int, 可选) - STFT 窗口之间的跳数长度。默认值: None, 将设置为 `win_length/2`。
- **window_type** (`WindowType`, 可选) - Griffin-Lim 的窗口类型, 可以是 `WindowType.BARTLETT`, `WindowType.BLACKMAN`, `WindowType.HAMMING`, `WindowType.HANN` 或 `WindowType.KAISER`。默认值: `WindowType.HANN`, 目前 macOS 上不支持 `kaiser` 窗口。
- **power** (float, 可选) - 幅度谱图的指数。默认值: 2.0。
- **momentum** (float, 可选) - 快速 Griffin-Lim 的动量。默认值: 0.99。
- **length** (int, 可选) - 预期输出波形的长度。默认值: None, 将设置为 `stft` 矩阵的最后一个维度的值。
- **rand_init** (bool, 可选) - 随机相位初始化或全零相位初始化标志。默认值: True。

异常:

- **TypeError** - 如果 `n_fft` 的类型不为 int。

- **ValueError** - 如果 `n_fft` 不为正数。
- **TypeError** - 如果 `n_iter` 的类型不为 `int`。
- **ValueError** - 如果 `n_iter` 不为正数。
- **TypeError** - 如果 `win_length` 的类型不为 `int`。
- **ValueError** - 如果 `win_length` 为负数。
- **TypeError** - 如果 `hop_length` 的类型不为 `int`。
- **ValueError** - 如果 `hop_length` 为负数。
- **TypeError** - 如果 `window_type` 的类型不为 `mindspore.dataset.audio.WindowType`。
- **TypeError** - 如果 `power` 的类型不为 `float`。
- **ValueError** - 如果 `power` 不为正数。
- **TypeError** - 如果 `momentum` 的类型不为 `float`。
- **ValueError** - 如果 `momentum` 为负数。
- **TypeError** - 如果 `length` 的类型不为 `int`。
- **ValueError** - 如果 `length` 为负数。
- **TypeError** - 如果 `rand_init` 的类型不为 `bool`。
- **RuntimeError** - 当 `n_fft` 指定的 FFT 长度不小于 `length` 指定的输出波形长度。
- **RuntimeError** - 当 `win_length` 指定的窗口长度不小于 `n_fft` 指定的 FFT 长度。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 201, 6]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.GriffinLim(n_fft=400)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(1000,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([201, 6]) # 1 sample
>>> output = audio.GriffinLim(n_fft=400)(waveform)
>>> print(output.shape, output.dtype)
(1000,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.HighpassBiquad

class mindspore.dataset.audio.HighpassBiquad (sample_rate, cutoff_freq, Q=0.707)

给音频波形上施加双二阶高通滤波器。

接口实现方式类似于 SoX 库。

参数：

- **sample_rate** (int) - 波形的采样频率，如 44100（单位：Hz），不能为 0。
- **cutoff_freq** (float) - 中心频率（单位：Hz）。
- **Q** (float, 可选) - 品质因子，取值范围为 (0, 1]。默认值：0.707。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *cutoff_freq* 的类型不为 float。
- **TypeError** - 当 *Q* 的类型不为 float。
- **ValueError** - 当 *Q* 取值不在 (0, 1] 范围内。
- **RuntimeError** - 当输入音频的 shape 不为 <..., time>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.HighpassBiquad(44100, 1500, 0.7)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.HighpassBiquad(44100, 1500, 0.7)(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.InverseMelScale

```
class mindspore.dataset.audio.InverseMelScale (n_stft, n_mels=128, sample_rate=16000, f_min=0.0,  

f_max=None, max_iter=100000, tolerance_loss=1e-5,  

tolerance_change=1e-8, sgdargs=None,  

norm=NormType.NONE, mel_type=MelType.HTK)
```

使用转换矩阵从梅尔频率 STFT 求解普通频率的 STFT。

参数：

- **n_stft** (int) - STFT 中的频段数。
- **n_mels** (int, 可选) - 梅尔滤波器的数量。默认值：128。
- **sample_rate** (int, 可选) - 音频信号采样频率。默认值：16000。
- **f_min** (float, 可选) - 最小频率。默认值：0.0。
- **f_max** (float, 可选) - 最大频率。默认值：None，将设置为 *sample_rate/2*。
- **max_iter** (int, 可选) - 最大优化迭代次数。默认值：100000。
- **tolerance_loss** (float, 可选) - 当达到损失值时停止优化。默认值：1e-5。
- **tolerance_change** (float, 可选) - 指定损失差异，当达到损失差异时停止优化。默认值：1e-8。
- **sgdargs** (dict, 可选) - SGD 优化器的参数。默认值：None，将设置为 {'sgd_lr': 0.1, 'sgd_momentum': 0.9}。
- **norm** (*NormType*, 可选) - 标准化方法，可以是 *NormType.SLANEY* 或 *NormType.NONE*。默认值：*NormType.NONE*，不使用标准化。
- **mel_type** (*MelType*, 可选) - 要使用的梅尔比例，可以是 *MelType.SLAN* 或 *MelType.HTK*。默认值：*MelType.HTK*。

异常：

- **TypeError** - 如果 *n_fft* 的类型不为 int。
- **ValueError** - 如果 *n_fft* 不为正数。
- **TypeError** - 如果 *n_mels* 的类型不为 int。
- **ValueError** - 如果 *n_mels* 不为正数。
- **TypeError** - 如果 *sample_rate* 的类型不为 int。
- **ValueError** - 如果 *sample_rate* 不为正数。
- **TypeError** - 如果 *f_min* 的类型不为 float。
- **ValueError** - 如果 *f_min* 大于等于 *f_max*。
- **TypeError** - 如果 *f_max* 的类型不为 float。
- **ValueError** - 如果 *f_max* 为负数。
- **TypeError** - 如果 *max_iter* 的类型不为 int。
- **ValueError** - 如果 *max_iter* 为负数。
- **TypeError** - 如果 *tolerance_loss* 的类型不为 float。
- **ValueError** - 如果 *tolerance_loss* 为负数。
- **TypeError** - 如果 *tolerance_change* 的类型不为 float。
- **ValueError** - 如果 *tolerance_change* 为负数。
- **TypeError** - 如果 *sgdargs* 的类型不为 dict。

- **TypeError** - 如果 *norm* 的类型不为 `mindspore.dataset.audio.NormType`。
- **TypeError** - 如果 *mel_type* 的类型不为 `mindspore.dataset.audio.MelType`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.randn(5, 8, 3, 2) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.InverseMelScale(20, 3, 16000, 0, 8000, 10)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(8, 20, 2) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.randn([8, 3, 2]) # 1 sample
>>> output = audio.InverseMelScale(20, 3, 16000, 0, 8000, 10)(waveform)
>>> print(output.shape, output.dtype)
(8, 20, 2) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.InverseSpectrogram`

```
class mindspore.dataset.audio.InverseSpectrogram (length=None, n_fft=400, win_length=None,
                                                    hop_length=None, pad=0, window=WindowType.HANN,
                                                    normalized=False, center=True,
                                                    pad_mode=BorderType.REFLECT, onesided=True)
```

计算输入频谱的反向频谱，以恢复原始音频信号。

参数：

- **length** (int, 可选) - 波形的输出长度，必须是非负数。默认值：None，表示输出整个波形。
- **n_fft** (int, 可选) - FFT 的大小，创建 $n\_FFT/2+1$ 个频段，应该大于 0。默认值：400。
- **win_length** (int, 可选) - 窗口大小，应该大于 0。默认值：None，将被设为 *n_fft*。
- **hop_length** (int, 可选) - STFT 窗口之间的跳跃长度，应该大于 0。默认值：None，将被设为 *win_length* // 2。
- **pad** (int, 可选) - 信号两端的填充长度，不能小于 0。默认值：0。
- **window** (*WindowType*, 可选) - 作用于每一帧的窗口函数。默认值：WindowType.HANN。
- **normalized** (bool, 可选) - 是否在 stft 之后按幅度执行标准化。默认值：False。
- **center** (bool, 可选) - 是否同时在波形两端进行填充。默认值：True。

- **pad_mode** (*BorderType*, 可选) - 控制在 *center* 为 True 时使用的填充方法, 可为 *BorderType.REFLECT*、*BorderType.CONSTANT*、*BorderType.EDGE*、*BorderType.SYMMETRIC*。默认值: *BorderType.REFLECT*。
- **onesided** (bool, 可选) - 控制是否只返回一半波形, 以避免冗余。默认值: True。

异常:

- **TypeError** - 如果 *length* 的类型不为 int。
- **ValueError** - 如果 *length* 为负数。
- **TypeError** - 如果 *n_fft* 的类型不为 int。
- **ValueError** - 如果 *n_fft* 不为正数。
- **TypeError** - 如果 *win_length* 的类型不为 int。
- **ValueError** - 如果 *win_length* 不为正数。
- **TypeError** - 如果 *hop_length* 的类型不为 int。
- **ValueError** - 如果 *hop_length* 不为正数。
- **TypeError** - 如果 *pad* 的类型不为 int。
- **ValueError** - 如果 *pad* 为负数。
- **TypeError** - 如果 *window* 的类型不为 *mindspore.dataset.audio.WindowType*。
- **TypeError** - 如果 *normalized* 的类型不为 bool。
- **TypeError** - 如果 *center* 的类型不为 bool。
- **TypeError** - 如果 *pad_mode* 的类型不为 *mindspore.dataset.audio.BorderType*。
- **TypeError** - 如果 *onesided* 的类型不为 bool。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 400 // 2 + 1, 30, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.InverseSpectrogram(1, 400, 400, 200)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
-> "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(1,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([400 // 2 + 1, 30, 2]) # 1 sample
>>> output = audio.InverseSpectrogram(1, 400, 400, 200)(waveform)
>>> print(output.shape, output.dtype)
(1,) float64
```

教程样例：

- [音频变换样例库](#)

mindspore.dataset.audio.LFCC

class mindspore.dataset.audio.LFCC (*sample_rate=16000, n_filter=128, n_lfcc=40, f_min=0.0, f_max=None, dct_type=2, norm=NormMode.ORTHO, log_lf=False, speckwargs=None*)

计算音频信号的线性频率倒谱系数。

说明：待处理音频 shape 需为 <..., time>。

参数：

- **sample_rate** (int, 可选) - 音频信号的采样率。默认值：16000。
- **n_filter** (int, 可选) - 要应用的线性滤波器数量。默认值：128。
- **n_lfcc** (int, 可选) - 要保留的线性频率倒谱系数数。默认值：40。
- **f_min** (float, 可选) - 最小频率。默认值：0.0。
- **f_max** (float, 可选) - 最大频率。默认值：None，会被设置为 *sample_rate // 2*。
- **dct_type** (int, 可选) - 要使用的离散余弦变换的类型。该值只能为 2。默认值：2。
- **norm** (*NormMode*, 可选) - 要使用的标准化方法。默认值：NormMode.ORTHO。
- **log_lf** (bool, 可选) - 是否使用对数-线性频谱图而不是以分贝为刻度的频谱图。默认值：False。
- **speckwargs** (dict, 可选) - *Spectrogram* 接口的参数。默认值：None，会被设置为包含以下字段的字典
 - 'n_fft': 400
 - 'win_length': n_fft
 - 'hop_length': win_length // 2
 - 'pad': 0
 - 'window': WindowType.HANN
 - 'power': 2.0
 - 'normalized': False
 - 'center': True
 - 'pad_mode': BorderType.REFLECT
 - 'onesided': True

异常：

- **TypeError** - 如果 *sample_rate* 的类型不为 int。
- **TypeError** - 如果 *n_filter* 的类型不为 int。
- **TypeError** - 如果 *n_lfcc* 的类型不为 int。
- **TypeError** - 如果 *norm* 的类型不为 *mindspore.dataset.audio.NormMode*。
- **TypeError** - 如果 *log_lf* 的类型不为 bool。
- **TypeError** - 如果 *speckwargs* 的类型不为 dict。

- **ValueError** - 如果 `sample_rate` 为 0。
- **ValueError** - 如果 `n_lfcc` 小于 0。
- **ValueError** - 如果 `f_min` 大于 `f_max`。
- **ValueError** - 当 `f_max` 为 None 时, 如果 `f_min` 大于 `sample_rate // 2`。
- **ValueError** - 如果 `dct_type` 不为 2。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 10, 300])
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.LFCC()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(10, 40, 2) float32
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([10, 300]) # 1 sample
>>> output = audio.LFCC()(waveform)
>>> print(output.shape, output.dtype)
(10, 40, 2) float32
```

教程样例:

- 音频变换样例库

`mindspore.dataset.audio.LFilter`

class `mindspore.dataset.audio.LFilter` (`a_coeffs`, `b_coeffs`, `clamp=True`)

根据指定的差分方程施加 IIR 滤波器。

参数:

- **a_coeffs** (`Sequence[float]`) - 不同维度的差分方程分母系数。低维的延迟系数在前, 例如 `[a0, a1, a2, ...]`。维度必须与 `b_coeffs` 相同 (根据需要填充 0 值)。
- **b_coeffs** (`Sequence[float]`) - 不同维度的差分方程分子系数。低维的延迟系数在前, 例如 `[b0, b1, b2, ...]`。维度必须与 `a_coeffs` 相同 (根据需要填充 0 值)。
- **clamp** (`bool`, 可选) - 如果为 `True`, 将输出信号截断在 `[-1, 1]` 范围内。默认值: `True`。

异常:

- **TypeError** - 当 `a_coeffs` 的类型不为 `Sequence[float]`。
- **TypeError** - 当 `b_coeffs` 的类型不为 `Sequence[float]`。

- **ValueError** - 当 *a_coeffs* 与 *b_coeffs* 维度不同。
- **TypeError** - 当 *clamp* 的类型不为 bool。
- **RuntimeError** - 当输入音频的 shape 不为 $\langle \dots, \text{time} \rangle$ 。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.LFilter(a_coeffs=[0.1, 0.2, 0.3], b_coeffs=[0.3, 0.2, 0.1])]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16]) # 1 sample
>>> output = audio.LFilter(a_coeffs=[0.1, 0.2, 0.3], b_coeffs=[0.3, 0.2, 0.1])(waveform)
>>> print(output.shape, output.dtype)
(16,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.LowpassBiquad

class mindspore.dataset.audio.**LowpassBiquad** (*sample_rate*, *cutoff_freq*, *Q=0.707*)

给音频波形施加双极点低通滤波器。

低通滤波器允许低频信号通过，但减弱频率高于截止频率的信号。其系统函数为：

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1}$$

接口实现方式类似于 [SoX 库](#)。

说明：待处理音频 shape 需为 $\langle \dots, \text{time} \rangle$ 。

参数：

- **sample_rate** (int) - 采样频率（单位：Hz），不能为零。
- **cutoff_freq** (float) - 滤波器截止频率（单位：Hz）。
- **Q** (float, 可选) - 品质因子，能够反映带宽与采样频率和中心频率的关系，取值范围 (0, 1]。默认值：0.707。

异常:

- **TypeError** - 当 *sample_rate* 的类型不为 `int`。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *central_freq* 的类型不为 `float`。
- **TypeError** - 当 *Q* 的类型不为 `float`。
- **ValueError** - 当 *Q* 取值不在 (0, 1] 范围内。
- **RuntimeError** - 当输入音频的 `shape` 不为 `<..., time>`。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 10]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.LowpassBiquad(4000, 1500, 0.7)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(10,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([10]) # 1 sample
>>> output = audio.LowpassBiquad(4000, 1500, 0.7)(waveform)
>>> print(output.shape, output.dtype)
(10,) float64
```

教程样例:

- 音频变换样例库

`mindspore.dataset.audio.Magphase`

class `mindspore.dataset.audio.Magphase` (*power=1.0*)

将 `shape` 为 `(..., 2)` 的复值光谱图分离, 输出幅度和相位。

参数:

- **power** (`float`) - 范数的幂, 必须是非负的。默认值: 1.0。

异常:

- **RuntimeError** - 当输入音频的 `shape` 不为 `(..., 2)`。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Magphase()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"],
...                                              output_columns=["spect", "phase"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["spect"].shape, item["spect"].dtype)
...     break
(16,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 2]) # 1 sample
>>> output = audio.Magphase()(waveform)
>>> print(output[0].shape, output[0].dtype)
(16,) float64
```

教程样例:

- 音频变换样例库

`mindspore.dataset.audio.MaskAlongAxis`

class `mindspore.dataset.audio.MaskAlongAxis` (*mask_start, mask_width, mask_value, axis*)

对音频波形应用掩码。掩码的起始和长度由 [*mask_start, mask_start + mask_width*] 决定。

参数:

- **mask_start** (int) - 掩码的起始位置, 必须是非负的。
- **mask_width** (int) - 掩码的宽度, 必须是大于 0。
- **mask_value** (float) - 填充到掩码区间的值。
- **axis** (int) - 要应用掩码的轴 (1 表示频率, 2 表示时间)。

异常:

- **ValueError** - *mask_start* 参数值错误 (小于 0)。
- **ValueError** - *mask_width* 参数值错误 (小于 1)。
- **ValueError** - *axis* 参数类型错误或者值错误, 不属于 [1, 2]。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
```

(下页继续)

(续上页)

```

>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 20, 20]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.MaskAlongAxis(0, 10, 0.5, 1)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳"audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(20, 20) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([20, 20]) # 1 sample
>>> output = audio.MaskAlongAxis(0, 10, 0.5, 1)(waveform)
>>> print(output.shape, output.dtype)
(20, 20) float64

```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.MaskAlongAxisIID

class mindspore.dataset.audio.MaskAlongAxisIID(*mask_param, mask_value, axis*)

对音频波形沿 *axis* 轴应用掩码。掩码的起始和长度由 [*mask_start, mask_start + mask_width*) 决定，其中 *mask_width* 从 *uniform*[0, *mask_param*] 中采样，*mask_start* 从 *uniform*[0, *max_length - mask_width*] 中采样，*max_length* 是光谱图中特定轴的列数。

参数：

- **mask_param** (int) - 要屏蔽的列数，将从 [0, *mask_param*] 统一采样，必须是非负数。
- **mask_value** (float) - 填充到掩码区间的值。
- **axis** (int) - 要应用掩码的轴（1 表示频率，2 表示时间）。

异常：

- **TypeError** - 当 *mask_param* 的类型不为 int。
- **ValueError** - 当 *mask_param* 为负数。
- **TypeError** - 当 *mask_value* 的类型不为 float。
- **TypeError** - 当 *axis* 的类型不为 int。
- **ValueError** - 当 *axis* 取值不在 [1, 2] 范围内。
- **RuntimeError** - 当输入音频的 *shape* 不为 <..., freq, time>。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>

```

(下页继续)

(续上页)

```

>>> # Use the transform in dataset pipeline mode
>>> waveform= np.random.random([5, 20, 20]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.MaskAlongAxisIID(5, 0.5, 2)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳"audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(20, 20) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([20, 20]) # 1 sample
>>> output = audio.MaskAlongAxisIID(5, 0.5, 2)(waveform)
>>> print(output.shape, output.dtype)
(20, 20) float64

```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.MelScale`

class mindspore.dataset.audio.**MelScale** (*n_mels=128, sample_rate=16000, f_min=0.0, f_max=None, n_stft=201, norm=NormType.NONE, mel_type=MelType.HTK*)

将普通 STFT 转换为梅尔尺度的 STFT。

参数：

- **n_mels** (int, 可选) - 梅尔滤波器的数量。默认值：128。
- **sample_rate** (int, 可选) - 音频信号采样速率。默认值：16000（单位：Hz）。
- **f_min** (float, 可选) - 最小频率。默认值：0.0。
- **f_max** (float, 可选) - 最大频率。默认值：None，将设置为 `sample_rate//2`。
- **n_stft** (int, 可选) - STFT 中的频段数。默认值：201。
- **norm** (*NormType*, 可选) - 标准化方法，可以是 *NormType.SLANEY* 或 *NormType.NONE*。默认值：*NormType.NONE*，不使用标准化。若采用 *NormType.SLANEY*，则三角梅尔权重将被除以梅尔频带的宽度。
- **mel_type** (*MelType*, 可选) - 要使用的 Mel 比例，可以是 *MelType.SLAN* 或 *MelType.HTK*。默认值：*MelType.HTK*。

异常：

- **TypeError** - 如果 `n_mels` 的类型不为 int。
- **ValueError** - 如果 `n_mels` 不为正数。
- **TypeError** - 如果 `sample_rate` 的类型不为 int。
- **ValueError** - 如果 `sample_rate` 不为正数。
- **TypeError** - 如果 `f_min` 的类型不为 float。
- **ValueError** - 如果 `f_min` 大于等于 `f_max`。
- **TypeError** - 如果 `f_max` 的类型不为 float。

- **ValueError** - 如果 f_max 为负数。
- **TypeError** - 如果 n_stft 的类型不为 `int`。
- **ValueError** - 如果 n_stft 不为正数。
- **TypeError** - 如果 $norm$ 的类型不为 `mindspore.dataset.audio.NormType`。
- **TypeError** - 如果 mel_type 的类型不为 `mindspore.dataset.audio.MelType`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 201, 3]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.MelScale(200, 1500, 0.7)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(200, 3) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([201, 3]) # 1 sample
>>> output = audio.MelScale(200, 1500, 0.7)(waveform)
>>> print(output.shape, output.dtype)
(200, 3) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.MelSpectrogram`

```
class mindspore.dataset.audio.MelSpectrogram (sample_rate=16000, n_fft=400, win_length=None,
hop_length=None, f_min=0.0, f_max=None, pad=0,
n_mels=128, window=WindowType.HANN, power=2.0,
normalized=False, center=True,
pad_mode=BorderType.REFLECT, onesided=True,
norm=NormType.NONE, mel_scale=MelType.HTK)
```

计算原始音频信号的梅尔频谱。

参数：

- **sample_rate** (`int`, 可选) - 采样频率（单位：Hz），不能小于 0。默认值：16000。
- **n_fft** (`int`, 可选) - FFT 的大小，创建 $n\_fft // 2 + 1$ 个频段，应该大于 0 并且小于输入张量最后一维大小的两倍。默认值：400。
- **win_length** (`int`, 可选) - 窗口大小，应该大于 0 并且不能大于 n_fft 。默认值：None，会被设置为 n_fft 。

- **hop_length** (int, 可选) - STFT 窗口之间的跳跃长度, 应该大于 0。默认值: `None`, 会被设置为 `win_length // 2`。
- **f_min** (float, 可选) - 最小频率, 不能大于 `f_max`。默认值: `0.0`。
- **f_max** (float, 可选) - 最大频率, 不能小于 0。默认值: `None`, 会被设置为 `sample_rate // 2`。
- **pad** (int, 可选) - 信号两端的填充长度, 不能小于 0。默认值: `0`。
- **n_mels** (int, 可选) - 梅尔滤波器组的数量, 不能小于 0。默认值: `128`。
- **window** (`WindowType`, 可选) - 作用于每一帧的窗口函数。默认值: `WindowType.HANN`。
- **power** (float, 可选) - 幅值谱图的指数, 应该大于 0, 例如, 1 表示能量, 2 表示功率, 等等。默认值: `2.0`。
- **normalized** (bool, 可选) - 是否在 stft 之后按幅度执行标准化。默认值: `False`。
- **center** (bool, 可选) - 是否同时在波形两端进行填充。默认值: `True`。
- **pad_mode** (`BorderType`, 可选) - 控制在 `center` 为 `True` 时使用的填充方法, 可为 `BorderType.REFLECT`、`BorderType.CONSTANT`、`BorderType.EDGE`、`BorderType.SYMMETRIC`。默认值: `BorderType.REFLECT`。
- **onesided** (bool, 可选) - 控制是否只返回一半波形, 以避免冗余。默认值: `True`。
- **norm** (`NormType`, 可选) - 如果为 `'slaney'`, 则将三角形梅尔权重除以梅尔带的宽度 (区域归一化)。默认值: `NormType.NONE`, 不使用标准化。
- **mel_scale** (`MelType`, 可选) - 要使用的 Mel 比例, 可以是 `MelType.SLAN` 或 `MelType.HTK`。默认值: `MelType.HTK`。

异常:

- **TypeError** - 如果 `sample_rate` 的类型不为 `int`。
- **TypeError** - 如果 `n_fft` 的类型不为 `int`。
- **TypeError** - 如果 `n_mels` 的类型不为 `int`。
- **TypeError** - 如果 `f_min` 的类型不为 `float`。
- **TypeError** - 如果 `f_max` 的类型不为 `float`。
- **TypeError** - 如果 `window` 的类型不为 `mindspore.dataset.audio.WindowType`。
- **TypeError** - 如果 `norm` 的类型不为 `mindspore.dataset.audio.NormType`。
- **TypeError** - 如果 `mel_scale` 的类型不为 `mindspore.dataset.audio.MelType`。
- **TypeError** - 如果 `power` 的类型不为 `float`。
- **TypeError** - 如果 `normalized` 的类型不为 `bool`。
- **TypeError** - 如果 `center` 的类型不为 `bool`。
- **TypeError** - 如果 `pad_mode` 的类型不为 `mindspore.dataset.audio.BorderType`。
- **TypeError** - 如果 `onesided` 的类型不为 `bool`。
- **TypeError** - 如果 `pad` 的类型不为 `int`。
- **TypeError** - 如果 `win_length` 的类型不为 `int`。
- **TypeError** - 如果 `hop_length` 的类型不为 `int`。
- **ValueError** - 如果 `sample_rate` 为负数。
- **ValueError** - 如果 `n_fft` 不为正数。
- **ValueError** - 如果 `n_mels` 为负数。

- **ValueError** - 如果 f_min 大于 f_max 。
- **ValueError** - 如果 f_max 为负数。
- **ValueError** - 当 f_max 为 None 时, 如果 f_min 大于 $sample\_rate // 2$ 。
- **ValueError** - 如果 $power$ 不为正数。
- **ValueError** - 如果 pad 为负数。
- **ValueError** - 如果 win_length 不为正数。
- **ValueError** - 如果 hop_length 不为正数。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 32]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.MelSpectrogram(sample_rate=16000, n_fft=16, win_length=16, hop_
↳length=8, f_min=0.0,
...                                     f_max=5000.0, pad=0, n_mels=2, window=audio.
↳WindowType.HANN, power=2.0,
...                                     normalized=False, center=True, pad_mode=audio.
↳BorderType.REFLECT,
...                                     onesided=True, norm=audio.NormType.SLANEY, mel_
↳scale=audio.MelType.HTK)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↳"audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(2, 5) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([32]) # 1 sample
>>> output = audio.MelSpectrogram(sample_rate=16000, n_fft=16, win_length=16, hop_length=8,
↳f_min=0.0,
...                                     f_max=5000.0, pad=0, n_mels=2, window=audio.WindowType.
↳HANN, power=2.0,
...                                     normalized=False, center=True, pad_mode=audio.BorderType.
↳REFLECT,
...                                     onesided=True, norm=audio.NormType.SLANEY,
...                                     mel_scale=audio.MelType.HTK)(waveform)
>>> print(output.shape, output.dtype)
(2, 5) float64
```

教程样例:

- 音频变换样例库

mindspore.dataset.audio.MFCC

class mindspore.dataset.audio.**MFCC** (*sample_rate=16000, n_mfcc=40, dct_type=2, norm=NormMode.ORTHO, log_mels=False, melkwargs=None*)

计算音频信号的梅尔频率倒谱系数。

参数：

- **sample_rate** (int, 可选) - 采样频率（单位：Hz），不能小于零。默认值：16000。
- **n_mfcc** (int, 可选) - 要保留的梅尔频率倒谱系数数，不能小于零。默认值：40。
- **dct_type** (int, 可选) - 要使用的离散余弦变换类型（离散余弦变换），只能为 2。默认值：2。
- **norm** (*NormMode*, 可选) - 要使用的标准类型。默认值：NormMode.ORTHO。
- **log_mels** (bool, 可选) - 是否使用对数-梅尔频谱图而不是以分贝为刻度的频谱图。默认值：False。
- **melkwargs** (dict, 可选) - *MelSpectrogram* 接口的参数。默认值：None，会被设置为包含以下字段的字典
 - 'n_fft': 400
 - 'win_length': n_fft
 - 'hop_length': win_length // 2
 - 'f_min': 0.0
 - 'f_max': sample_rate // 2
 - 'pad': 0
 - 'window': WindowType.HANN
 - 'power': 2.0
 - 'normalized': False
 - 'center': True
 - 'pad_mode': BorderType.REFLECT
 - 'onesided': True
 - 'norm': NormType.NONE
 - 'mel_scale': MelType.HTK

异常：

- **TypeError** - 如果 *sample_rate* 的类型不为 int。
- **TypeError** - 如果 *log_mels* 的类型不为 bool。
- **TypeError** - 如果 *norm* 的类型不为 *mindspore.dataset.audio.NormMode*。
- **TypeError** - 如果 *n_mfcc* 的类型不为 int。
- **TypeError** - 如果 *melkwargs* 的类型不为 dict。
- **ValueError** - 如果 *sample_rate* 为负数。
- **ValueError** - 如果 *n_mfcc* 为负数。
- **ValueError** - 如果 *dct_type* 不为 2。

支持平台：

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 500]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.MFCC(4000, 128, 2)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(128, 3) float32
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([500]) # 1 sample
>>> output = audio.MFCC(4000, 128, 2)(waveform)
>>> print(output.shape, output.dtype)
(128, 3) float32
```

教程样例:

- 音频变换样例库

`mindspore.dataset.audio.MuLawDecoding`

class `mindspore.dataset.audio.MuLawDecoding` (*quantization_channels=256*)

解码 mu-law 编码的信号, 参考 [mu-law 算法](#)。

参数:

- **quantization_channels** (int, 可选) - 通道数, 必须为正数。默认值: 256。

异常:

- **TypeError** - 当 *quantization_channels* 的类型不为 int。
- **ValueError** - 当 *quantization_channels* 不为正数。
- **RuntimeError** - 当输入音频的 shape 不为 `<..., time>`。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 3, 4]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.MuLawDecoding()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
```

(下页继续)

(续上页)

```
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(3, 4) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([3, 4]) # 1 sample
>>> output = audio.MuLawDecoding()(waveform)
>>> print(output.shape, output.dtype)
(3, 4) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.MuLawEncoding`

class `mindspore.dataset.audio.MuLawEncoding` (`quantization_channels=256`)

基于 mu-law 压缩的信号编码。

参数：

- **quantization_channels** (int, 可选) - 通道数，必须为正数。默认值：256。

异常：

- **TypeError** - 当 `quantization_channels` 的类型不为 int。
- **ValueError** - 当 `quantization_channels` 不为正数。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 3, 4]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.MuLawEncoding()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
...     ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(3, 4) int32
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([3, 4]) # 1 sample
>>> output = audio.MuLawEncoding()(waveform)
>>> print(output.shape, output.dtype)
(3, 4) int32
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.Overdrive`

class `mindspore.dataset.audio.Overdrive` (*gain=20.0, color=20.0*)

给音频波形施加过载效果。

接口实现方式类似于 [SoX 库](#)。

参数：

- **gain** (float, 可选) - 期望提升（或衰减）的音频增益，单位为 dB，取值范围为 [0, 100]。默认值：20.0。
- **color** (float, 可选) - 控制过载输出中偶次谐波成份的量，取值范围为 [0, 100]。默认值：20.0。

异常：

- **TypeError** - 当 *gain* 的类型不为 float。
- **ValueError** - 当 *gain* 取值不在 [0, 100] 范围内。
- **TypeError** - 当 *color* 的类型不为 float。
- **ValueError** - 当 *color* 取值不在 [0, 100] 范围内。
- **RuntimeError** - 当输入音频的 shape 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 10]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Overdrive()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
...     ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(10,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([10]) # 1 sample
>>> output = audio.Overdrive()(waveform)
>>> print(output.shape, output.dtype)
(10,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Phaser

class mindspore.dataset.audio.**Phaser** (*sample_rate, gain_in=0.4, gain_out=0.74, delay_ms=3.0, decay=0.4, mod_speed=0.5, sinusoidal=True*)

给音频波形施加相位效果。

接口实现方式类似于 [SoX 库](#)。

参数：

- **sample_rate** (int) - 波形的采样率，例如 44100 (Hz)。
- **gain_in** (float, 可选) - 期望输入提升（或衰减）的增益，单位为 dB，取值范围为 [0.0, 1.0]。默认值：0.4。
- **gain_out** (float, 可选) - 期望输出提升（或衰减）的增益，单位为 dB，取值范围为 [0.0, 1e9]。默认值：0.74。
- **delay_ms** (float, 可选) - 期望的时延，单位为毫秒，取值范围为 [0.0, 5.0]。默认值：3.0。
- **decay** (float, 可选) - 期望的输入增益衰减系数，取值范围为 [0.0, 0.99]。默认值：0.4。
- **mod_speed** (float, 可选) - 调制速率，单位为 Hz，取值范围为 [0.1, 2.0]。默认值：0.5。
- **sinusoidal** (bool, 可选) - 如果为 True，将使用正弦调制（适用于多乐器场景）。如果为 False，则使用三角调制（使单乐器获得更明显的相位效果）。默认值：True。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **TypeError** - 当 *gain_in* 的类型不为 float。
- **ValueError** - 当 *gain_in* 取值不在 [0.0, 1.0] 范围内。
- **TypeError** - 当 *gain_out* 的类型不为 float。
- **ValueError** - 当 *gain_out* 取值不在 [0.0, 1e9] 范围内。
- **TypeError** - 当 *delay_ms* 的类型不为 float。
- **ValueError** - 当 *delay_ms* 取值不在 [0.0, 5.0] 范围内。
- **TypeError** - 当 *decay* 的类型不为 float。
- **ValueError** - 当 *decay* 取值不在 [0.0, 0.99] 范围内。
- **TypeError** - 当 *mod_speed* 的类型不为 float。
- **ValueError** - 当 *mod_speed* 取值不在 [0.1, 2.0] 范围内。
- **TypeError** - 当 *sinusoidal* 的类型不为 bool。
- **RuntimeError** - 当输入音频的 shape 不为 <..., time>。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 12]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Phaser(44100)]
```

(下页继续)

(续上页)

```

>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(12,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([12]) # 1 sample
>>> output = audio.Phaser(44100)(waveform)
>>> print(output.shape, output.dtype)
(12,) float64

```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.PhaseVocoder

class mindspore.dataset.audio.PhaseVocoder (*rate, phase_advance*)

对给定的 STFT 频谱，在不改变音高的情况下以一定比率进行加速。

参数：

- **rate** (float) - 加速比率。
- **phase_advance** (numpy.ndarray) - 每个频段的预期相位提前量，shape 为 <freq, 1>。

异常：

- **TypeError** - 当 *rate* 的类型不为 float。
- **ValueError** - 当 *rate* 不为正数。
- **TypeError** - 当 *phase_advance* 的类型不为 `numpy.ndarray`。
- **RuntimeError** - 当输入音频的 shape 不为 <···, freq, num_frame, complex=2>。

支持平台：

CPU

样例：

```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 44, 10, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.PhaseVocoder(rate=2, phase_advance=np.random.random([44, 1]))]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(44, 5, 2) float64
>>>

```

(下页继续)

(续上页)

```
>>> # Use the transform in eager mode
>>> waveform = np.random.random([44, 10, 2]) # 1 sample
>>> output = audio.PhaseVocoder(rate=2, phase_advance=np.random.random([44, 1]))(waveform)
>>> print(output.shape, output.dtype)
(44, 5, 2) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Resample

```
class mindspore.dataset.audio.Resample (orig_freq=16000, new_freq=16000,
                                         resample_method=ResampleMethod.SINC_INTERPOLATION,
                                         lowpass_filter_width=6, rolloff=0.99, beta=None)
```

将信号从一个频率重采样至另一个频率。支持指定重采样方法。

参数：

- **orig_freq** (float, 可选) - 信号的原始频率，必须为正。默认值：16000。
- **new_freq** (float, 可选) - 预期的输出频率，必须为正。默认值：16000。
- **resample_method** (*ResampleMethod*, 可选) - 使用的重采样方法，可为 *ResampleMethod.SINC_INTERPOLATION* 或 *ResampleMethod.KAISER_WINDOW*。默认值：*ResampleMethod.SINC_INTERPOLATION*。
- **lowpass_filter_width** (int, 可选) - 控制滤波器的带宽，数值越大表示带宽越宽，但效率越低，必须为正。默认值：6。
- **rolloff** (float, 可选) - 滤波器的滚降频率，是奈奎斯特公式的一部分。值越低越利于减少抗混叠，但同时也会减少一部分最高频率，取值范围为 (0, 1]。默认值：0.99。
- **beta** (float, 可选) - Kaiser 窗的形状参数。默认值：None，将使用 14.769656459379492。

异常：

- **TypeError** - 当 *orig_freq* 的类型不为 float。
- **ValueError** - 当 *orig_freq* 不为正数。
- **TypeError** - 当 *new_freq* 的类型不为 float。
- **ValueError** - 当 *new_freq* 不为正数。
- **TypeError** - 当 *resample_method* 的类型不为 *mindspore.dataset.audio.ResampleMethod*。
- **TypeError** - 当 *lowpass_filter_width* 的类型不为 int。
- **ValueError** - 当 *lowpass_filter_width* 不为正数。
- **TypeError** - 当 *rolloff* 的类型不为 float。
- **ValueError** - 当 *rolloff* 取值不在 (0, 1] 范围内。
- **RuntimeError** - 当输入音频的 shape 不为 <..., time>。

支持平台：

CPU

样例：


```

>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 30]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Resample(orig_freq=48000, new_freq=16000,
...                               resample_method=audio.ResampleMethod.SINC_INTERPOLATION,
...                               lowpass_filter_width=6, rolloff=0.99, beta=None)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16, 10) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 30]) # 1 sample
>>> output = audio.Resample(orig_freq=48000, new_freq=16000,
...                          resample_method=audio.ResampleMethod.SINC_INTERPOLATION,
...                          lowpass_filter_width=6, rolloff=0.99, beta=None)(waveform)
>>> print(output.shape, output.dtype)
(16, 10) float64

```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.PitchShift

class mindspore.dataset.audio.**PitchShift** (*sample_rate, n_steps, bins_per_octave=12, n_fft=512, win_length=None, hop_length=None, window=WindowType.HANN*)

将波形的音调移动 n_steps 步长。

参数：

- **sample_rate** (int) - 波形的采样频率（单位：Hz）。
- **n_steps** (int) - 移动波形的步长。
- **bins_per_octave** (int, 可选) - 每倍频程的步长。默认值：12。
- **n_fft** (int, 可选) - FFT 的大小，创建 $n\_fft // 2 + 1$ 个频段。默认值：512。
- **win_length** (int, 可选) - 窗口大小。默认值：None，将会设置为 n_fft 。
- **hop_length** (int, 可选) - STFT 窗口之间的跳跃长度。默认值：None，则将设置为 $win\_length // 4$ 。
- **window** (*WindowType*, 可选) - 作用于每一帧的窗口函数。默认值：WindowType.HANN。

异常：

- **TypeError** - 如果 *sample_rate* 的类型不为 int。
- **TypeError** - 如果 *n_steps* 的类型不为 int。
- **TypeError** - 如果 *bins_per_octave* 的类型不为 int。
- **TypeError** - 如果 *n_fft* 的类型不为 int。

- **TypeError** - 如果 `win_length` 的类型不为 `int`。
- **TypeError** - 如果 `hop_length` 的类型不为 `int`。
- **TypeError** - 如果 `window` 的类型不为 `mindspore.dataset.audio.WindowType`。
- **ValueError** - 如果 `sample_rate` 为负数。
- **ValueError** - 如果 `bins_per_octave` 为 0。
- **ValueError** - 如果 `n_fft` 为负数。
- **ValueError** - 如果 `win_length` 不是正数。
- **ValueError** - 如果 `hop_length` 不是正数。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 8, 30]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.PitchShift(sample_rate=16000, n_steps=4)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(8, 30) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([8, 30]) # 1 sample
>>> output = audio.PitchShift(sample_rate=16000, n_steps=4)(waveform)
>>> print(output.shape, output.dtype)
(8, 30) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.RiaaBiquad`

class `mindspore.dataset.audio.RiaaBiquad` (*sample_rate*)

对输入音频波形施加 RIAA 均衡。

接口实现方式类似于 `SoX` 库。

参数：

- **sample_rate** (`int`) - 波形的采样率，例如 44100 (Hz)，只能是 44100、48000、88200、96000 中的一个。

异常：

- **TypeError** - 当 `sample_rate` 的类型不为 `int`。
- **ValueError** - 当 `sample_rate` 不为 44100、48000、88200、96000 中的任何一个。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 24]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.RiaaBiquad(44100)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(24,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([24]) # 1 sample
>>> output = audio.RiaaBiquad(44100)(waveform)
>>> print(output.shape, output.dtype)
(24,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.SlidingWindowCmn`

class `mindspore.dataset.audio.SlidingWindowCmn` (*cmn_window=600, min_cmn_window=100, center=False, norm_vars=False*)

对每个话语应用滑动窗口倒谱均值（和可选方差）归一化。

参数：

- **cmn_window** (int, 可选) - 用于运行平均 CMN 计算的帧中窗口。默认值：600。
- **min_cmn_window** (int, 可选) - 解码开始时使用的最小 CMN 窗口（仅在开始时增加延迟）。仅在 *center* 为 *False* 时适用，在 *center* 为 *True* 时忽略。默认值：100。
- **center** (bool, 可选) - 如果为 *True*，则使用以当前帧为中心的窗口。如果为 *False*，则窗口在左侧。默认值：*False*。
- **norm_vars** (bool, 可选) - 如果为 *True*，则将方差规范化为 1。默认值：*False*。

异常：

- **TypeError** - 当 *cmn_window* 的类型不为 *int*。
- **ValueError** - 当 *cmn_window* 为负数。
- **TypeError** - 当 *min_cmn_window* 的类型不为 *int*。
- **ValueError** - 当 *min_cmn_window* 为负数。
- **TypeError** - 当 *center* 的类型不为 *bool*。
- **TypeError** - 当 *norm_vars* 的类型不为 *bool*。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 3]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.SlidingWindowCmn()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16, 3) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 3]) # 1 sample
>>> output = audio.SlidingWindowCmn()(waveform)
>>> print(output.shape, output.dtype)
(16, 3) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.SpectralCentroid`

class `mindspore.dataset.audio.SpectralCentroid` (*sample_rate*, *n_fft*=400, *win_length*=None, *hop_length*=None, *pad*=0, *window*=`WindowType.HANN`)

计算每个通道沿时间轴的频谱中心。

参数：

- **sample_rate** (int) - 音频信号的采样率，例如 44100 (Hz)。
- **n_fft** (int, 可选) - FFT 的大小，将创建 $n\_fft // 2 + 1$ 个频段。默认值：400。
- **win_length** (int, 可选) - 窗口大小。默认值：None，将使用 *n_fft*。
- **hop_length** (int, 可选) - STFT 窗口之间的跳跃长度。默认值：None，将使用 $win\_length // 2$ 。
- **pad** (int, 可选) - 信号两端的填充长度。默认值：0。
- **window** (`WindowType`, 可选) - 作用于每一帧的窗口函数，可为 `WindowType.BARTLETT`、`WindowType.BLACKMAN`、`WindowType.HAMMING`、`WindowType.HANN` 或 `WindowType.KAISER`。默认值：`WindowType.HANN`。

异常：

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 为负数。
- **TypeError** - 当 *n_fft* 的类型不为 int。
- **ValueError** - 当 *n_fft* 不为正数。

- **TypeError** - 当 `win_length` 的类型不为 `int`。
- **ValueError** - 当 `win_length` 不为正数。
- **ValueError** - 当 `win_length` 大于 `n_fft`。
- **TypeError** - 当 `hop_length` 的类型不为 `int`。
- **ValueError** - 当 `hop_length` 不为正数。
- **TypeError** - 当 `pad` 的类型不为 `int`。
- **ValueError** - 当 `pad` 为负数。
- **TypeError** - 当 `window` 的类型不为 `mindspore.dataset.audio.WindowType`。
- **RuntimeError** - 当输入音频的 `shape` 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 10, 20]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.SpectralCentroid(44100)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(10, 1, 1) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([10, 20]) # 1 sample
>>> output = audio.SpectralCentroid(44100)(waveform)
>>> print(output.shape, output.dtype)
(10, 1, 1) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.Spectrogram`

```
class mindspore.dataset.audio.Spectrogram (n_fft=400, win_length=None, hop_length=None, pad=0,
                                             window=WindowType.HANN, power=2.0, normalized=False,
                                             center=True, pad_mode=BorderType.REFLECT, onesided=True)
```

从音频信号创建其频谱。

参数：

- **n_fft** (`int`, 可选) - FFT 的大小，将创建 `n_fft // 2 + 1` 个频段。默认值：400。
- **win_length** (`int`, 可选) - 窗口大小。默认值：None，将使用 `n_fft`。

- **hop_length** (int, 可选) - STFT 窗口之间的跳跃长度。默认值: None , 将使用 `win_length//2` 。
- **pad** (int, 可选) - 信号两端的填充长度。默认值: 0 。
- **window** (`WindowType`, 可选) - 作用于每一帧的窗口函数, 可为 `WindowType.BARTLETT`、`WindowType.BLACKMAN`、`WindowType.HAMMING`、`WindowType.HANN` 或 `WindowType.KAISER`。当前, 在 macOS 上暂不支持 Kaiser 窗。默认值: `WindowType.HANN` 。
- **power** (float, 可选) - 幅度谱图的指数, 必须非负, 例如 1 代表能量谱, 2 代表功率谱等。默认值: 2.0 。
- **normalized** (bool, 可选) - 是否在 stft 之后按幅度执行标准化。默认值: False 。
- **center** (bool, 可选) - 是否同时在波形两端进行填充。默认值: True 。
- **pad_mode** (`BorderType`, 可选) - 控制在 `center` 为 True 时使用的填充方法, 可为 `BorderType.REFLECT`、`BorderType.CONSTANT`、`BorderType.EDGE`、`BorderType.SYMMETRIC`。默认值: `BorderType.REFLECT` 。
- **onesided** (bool, 可选) - 控制是否只返回一半波形, 以避免冗余。默认值: True 。

异常:

- **TypeError** - 当 `n_fft` 的类型不为 int。
- **ValueError** - 当 `n_fft` 不为正数。
- **TypeError** - 当 `win_length` 的类型不为 int。
- **ValueError** - 当 `win_length` 不为正数。
- **ValueError** - 当 `win_length` 大于 `n_fft` 。
- **TypeError** - 当 `hop_length` 的类型不为 int。
- **ValueError** - 当 `hop_length` 不为正数。
- **TypeError** - 当 `pad` 的类型不为 int。
- **ValueError** - 当 `pad` 为负数。
- **TypeError** - 当 `window` 的类型不为 `mindspore.dataset.audio.WindowType` 。
- **TypeError** - 当 `power` 的类型不为 float。
- **ValueError** - 当 `power` 为负数。
- **TypeError** - 当 `normalized` 的类型不为 bool。
- **TypeError** - 当 `center` 的类型不为 bool。
- **TypeError** - 当 `pad_mode` 的类型不为 `mindspore.dataset.audio.BorderType` 。
- **TypeError** - 当 `onesided` 的类型不为 bool。
- **RuntimeError** - 当输入音频的 shape 不为 `<..., time>`。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 10, 20]) # 5 samples
```

(下页继续)

(续上页)

```

>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Spectrogram()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(10, 201, 1) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([10, 20]) # 1 sample
>>> output = audio.Spectrogram()(waveform)
>>> print(output.shape, output.dtype)
(10, 201, 1) float64

```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.TimeMasking`

class `mindspore.dataset.audio.TimeMasking` (*iid_masks=False, time_mask_param=0, mask_start=0, mask_value=0.0*)

给音频波形施加时域掩码。

说明：待处理音频 shape 需为 `<..., freq, time>`。

参数：

- **iid_masks** (bool, 可选) - 是否施加随机掩码。默认值：False。
- **time_mask_param** (int, 可选) - 当 *iid_masks* 为 True 时，掩码长度将从 `[0, time_mask_param]` 中均匀采样；当 *iid_masks* 为 False 时，直接使用该值作为掩码的长度。取值范围为 `[0, time_length]`，其中 *time_length* 为音频波形在时域的长度。默认值：0。
- **mask_start** (int, 可选) - 添加掩码的起始位置，只有当 *iid_masks* 为 True 时，该值才会生效。取值范围为 `[0, time_length - time_mask_param]`，其中 *time_length* 为音频波形在时域的长度。默认值：0。
- **mask_value** (float, 可选) - 掩码填充值。默认值：0.0。

异常：

- **TypeError** - 当 *iid_masks* 的类型不为 bool。
- **TypeError** - 当 *time_mask_param* 的类型不为 int。
- **ValueError** - 当 *time_mask_param* 大于音频时域长度。
- **TypeError** - 当 *mask_start* 的类型不为 int。
- **ValueError** - 当 *mask_start* 为负数。
- **TypeError** - 当 *mask_value* 的类型不为 float。
- **ValueError** - 当 *mask_value* 为负数。
- **RuntimeError** - 当输入音频的 shape 不为 `<..., freq, time>`。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.TimeMasking(time_mask_param=1)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(16, 2) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 2]) # 1 sample
>>> output = audio.TimeMasking(time_mask_param=1)(waveform)
>>> print(output.shape, output.dtype)
(16, 2) float64
```

教程样例:

- 音频变换样例库

`mindspore.dataset.audio.TimeStretch`

class `mindspore.dataset.audio.TimeStretch` (*hop_length=None, n_freq=201, fixed_rate=None*)

以给定的比例拉伸音频短时傅里叶 (Short Time Fourier Transform, STFT) 频谱的时域, 但不改变音频的音高。

说明: 待处理音频 shape 需为 `<..., freq, time, complex=2>`。第零维代表实部, 第一维代表虚部。

参数:

- **hop_length** (int, 可选) - STFT 窗之间每跳的长度, 即连续帧之间的样本数。默认值: `None`, 表示取 `n_freq - 1`。
- **n_freq** (int, 可选) - STFT 中的滤波器组数。默认值: `201`。
- **fixed_rate** (float, 可选) - 频谱在时域加快或减缓的比例。默认值: `None`, 表示保持原始速率。

异常:

- **TypeError** - 当 `hop_length` 的类型不为 `int`。
- **ValueError** - 当 `hop_length` 不为正数。
- **TypeError** - 当 `n_freq` 的类型不为 `int`。
- **ValueError** - 当 `n_freq` 不为正数。
- **TypeError** - 当 `fixed_rate` 的类型不为 `float`。

- **ValueError** - 当 *fixed_rate* 不为正数。
- **RuntimeError** - 当输入音频的 *shape* 不为 $\langle \dots, \text{freq}, \text{num\_frame}, \text{complex}=2 \rangle$ 。

支持平台:

CPU

样例:

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 16, 8, 2]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.TimeStretch()]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(1, 16, 8, 2) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([16, 8, 2]) # 1 sample
>>> output = audio.TimeStretch()(waveform)
>>> print(output.shape, output.dtype)
(1, 16, 8, 2) float64
```

教程样例:

- 音频变换样例库

mindspore.dataset.audio.TrebleBiquad

class mindspore.dataset.audio.**TrebleBiquad**(*sample_rate*, *gain*, *central_freq*=3000, *Q*=0.707)

给音频波形施加高音音调控制效果。

接口实现方式类似于 SoX 库。

参数:

- **sample_rate** (int) - 采样频率 (单位: Hz), 不能为 0。
- **gain** (float) - 期望提升 (或衰减) 的音频增益 (单位: dB)。
- **central_freq** (float, 可选) - 中心频率 (单位: Hz)。默认值: 3000。
- **Q** (float, 可选) - 品质因子, 能够反映带宽与采样频率和中心频率的关系, 取值范围为 (0, 1], 默认值: 0.707。

异常:

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 的数值为 0。
- **TypeError** - 当 *gain* 的类型不为 float。
- **TypeError** - 当 *central_freq* 的类型不为 float。

- **TypeError** - 当 Q 的类型不为 float。
- **ValueError** - 当 Q 取值不在 (0, 1] 范围内。
- **RuntimeError** - 当输入音频的 shape 不为 $\langle \dots, \text{time} \rangle$ 。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 20]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.TrebleBiquad(44100, 200.0)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(20,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([20]) # 1 sample
>>> output = audio.TrebleBiquad(44100, 200.0)(waveform)
>>> print(output.shape, output.dtype)
(20,) float64
```

教程样例：

- 音频变换样例库

mindspore.dataset.audio.Vad

```
class mindspore.dataset.audio.Vad(sample_rate, trigger_level=7.0, trigger_time=0.25, search_time=1.0,
                                allowed_gap=0.25, pre_trigger_time=0.0, boot_time=0.35, noise_up_time=0.1,
                                noise_down_time=0.01, noise_reduction_amount=1.35, measure_freq=20.0,
                                measure_duration=None, measure_smooth_time=0.4, hp_filter_freq=50.0,
                                lp_filter_freq=6000.0, hp_lifter_freq=150.0, lp_lifter_freq=2000.0)
```

语音活动检测器。

试图修剪去除语音记录末尾沉默或安静的背景声音。

接口实现方式类似于 [SoX 库](#)。

参数：

- **sample_rate** (int) - 音频信号的采样率。
- **trigger_level** (float, 可选) - 触发音频活动检测的测量级别。默认值：7.0。
- **trigger_time** (float, 可选) - 时间常数（以秒为单位），用于帮助忽略短促的爆破音。默认值：0.25。
- **search_time** (float, 可选) - 在检测到的触发点之前，搜索更安静/更短促爆破音的时长（以秒为单位）。默认值：1.0。

- **allowed_gap** (float, 可选) - 在检测到的触发点之前, 允许的更安静/更短促爆破音之间的时间间隔 (以秒为单位)。默认值: 0.25。
- **pre_trigger_time** (float, 可选) - 在触发点和任意找到的更安静/更短促的爆破音之前, 要保留的音频时长 (以秒为单位)。默认值: 0.0。
- **boot_time** (float, 可选) - 初始噪声的评估时间。默认值: 0.35。
- **noise_up_time** (float, 可选) - 用于自适应噪声评估的时间常量, 指定噪声等级上升时的时间。默认值: 0.1。
- **noise_down_time** (float, 可选) - 用于自适应噪声评估的时间常量, 指定噪声等级下降时的时间。默认值: 0.01。
- **noise_reduction_amount** (float, 可选) - 检测算法中使用的降噪量。默认值: 1.35。
- **measure_freq** (float, 可选) - 算法处理/检测的频率。默认值: 20.0。
- **measure_duration** (float, 可选) - 检测持续的时间。默认值: None, 将使用两倍测量周期的时长。
- **measure_smooth_time** (float, 可选) - 用于平滑频谱测量的时间常数。默认值: 0.4。
- **hp_filter_freq** (float, 可选) - 施加到检测器算法输入上的高通滤波器的截止频率。默认值: 50.0。
- **lp_filter_freq** (float, 可选) - 施加到检测器算法输入上的低通滤波器的截止频率。默认值: 6000.0。
- **hp_lifter_freq** (float, 可选) - 施加到检测器算法输入上的高通提升器的截止频率。默认值: 150.0。
- **lp_lifter_freq** (float, 可选) - 施加到检测器算法输入上的低通提升器的截止频率。默认值: 2000.0。

异常:

- **TypeError** - 当 *sample_rate* 的类型不为 int。
- **ValueError** - 当 *sample_rate* 不为正数。
- **TypeError** - 当 *trigger_level* 的类型不为 float。
- **TypeError** - 当 *trigger_time* 的类型不为 float。
- **ValueError** - 当 *trigger_time* 为负数。
- **TypeError** - 当 *search_time* 的类型不为 float。
- **ValueError** - 当 *search_time* 为负数。
- **TypeError** - 当 *allowed_gap* 的类型不为 float。
- **ValueError** - 当 *allowed_gap* 为负数。
- **TypeError** - 当 *pre_trigger_time* 的类型不为 float。
- **ValueError** - 当 *pre_trigger_time* 为负数。
- **TypeError** - 当 *boot_time* 的类型不为 float。
- **ValueError** - 当 *boot_time* 为负数。
- **TypeError** - 当 *noise_up_time* 的类型不为 float。
- **ValueError** - 当 *noise_up_time* 为负数。
- **TypeError** - 当 *noise_down_time* 的类型不为 float。
- **ValueError** - 当 *noise_down_time* 为负数。
- **ValueError** - 当 *noise_up_time* 小于 *noise_down_time*。
- **TypeError** - 当 *noise_reduction_amount* 的类型不为 float。
- **ValueError** - 当 *noise_reduction_amount* 为负数。

- **TypeError** - 当 `measure_freq` 的类型不为 `float`。
- **ValueError** - 当 `measure_freq` 不为正数。
- **TypeError** - 当 `measure_duration` 的类型不为 `float`。
- **ValueError** - 当 `measure_duration` 为负数。
- **TypeError** - 当 `measure_smooth_time` 的类型不为 `float`。
- **ValueError** - 当 `measure_smooth_time` 为负数。
- **TypeError** - 当 `hp_filter_freq` 的类型不为 `float`。
- **ValueError** - 当 `hp_filter_freq` 不为正数。
- **TypeError** - 当 `lp_filter_freq` 的类型不为 `float`。
- **ValueError** - 当 `lp_filter_freq` 不为正数。
- **TypeError** - 当 `hp_lifter_freq` 的类型不为 `float`。
- **ValueError** - 当 `hp_lifter_freq` 不为正数。
- **TypeError** - 当 `lp_lifter_freq` 的类型不为 `float`。
- **ValueError** - 当 `lp_lifter_freq` 不为正数。
- **RuntimeError** - 当输入音频的 `shape` 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 1000]) # 5 samples
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Vad(sample_rate=600)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↪ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(660,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([1000]) # 1 sample
>>> output = audio.Vad(sample_rate=600)(waveform)
>>> print(output.shape, output.dtype)
(660,) float64
```

教程样例：

- 音频变换样例库

`mindspore.dataset.audio.Vol`

class `mindspore.dataset.audio.Vol` (*gain*, *gain_type*=`GainType.AMPLITUDE`)

调整波形的音量。

参数：

- **gain** (float) - 提升（或衰减）的增益。如果 *gain_type* 为 `GainType.AMPLITUDE`，应为一个非负的幅度比。如果 *gain_type* 为 `GainType.POWER`，应为一个功率（电压的平方）。如果 *gain_type* 为 `GainType.DB`，应以分贝为单位。
- **gain_type** (`GainType`, 可选) - 增益的类型，可为 `GainType.AMPLITUDE`、`GainType.POWER` 或 `GainType.DB`。默认值：`GainType.AMPLITUDE`。

异常：

- **TypeError** - 当 *gain* 的类型不为 `float`。
- **TypeError** - 当 *gain_type* 的类型不为 `mindspore.dataset.audio.GainType`。
- **ValueError** - 当 *gain_type* 为 `GainType.AMPLITUDE` 时，*gain* 为负数。
- **ValueError** - 当 *gain_type* 为 `GainType.POWER` 时，*gain* 不为正数。
- **RuntimeError** - 当输入音频的 `shape` 不为 `<..., time>`。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> import mindspore.dataset as ds
>>> import mindspore.dataset.audio as audio
>>>
>>> # Use the transform in dataset pipeline mode
>>> waveform = np.random.random([5, 30]) # 5 sample
>>> numpy_slices_dataset = ds.NumpySlicesDataset(data=waveform, column_names=["audio"])
>>> transforms = [audio.Vol(gain=10, gain_type=audio.GainType.DB)]
>>> numpy_slices_dataset = numpy_slices_dataset.map(operations=transforms, input_columns=[
    ↳ "audio"])
>>> for item in numpy_slices_dataset.create_dict_iterator(num_epochs=1, output_numpy=True):
...     print(item["audio"].shape, item["audio"].dtype)
...     break
(30,) float64
>>>
>>> # Use the transform in eager mode
>>> waveform = np.random.random([30]) # 1 sample
>>> output = audio.Vol(gain=10, gain_type=audio.GainType.DB)(waveform)
>>> print(output.shape, output.dtype)
(30,) float64
```

教程样例：

- 音频变换样例库

13.4.3 工具

<code>mindspore.dataset.audio.BorderType</code>	填充模式。
<code>mindspore.dataset.audio.DensityFunction</code>	密度函数类型。
<code>mindspore.dataset.audio.FadeShape</code>	淡入淡出形状。
<code>mindspore.dataset.audio.GainType</code>	增益类型。
<code>mindspore.dataset.audio.Interpolation</code>	音频波形的插值模式。
<code>mindspore.dataset.audio.MelType</code>	梅尔标度实现类型。
<code>mindspore.dataset.audio.Modulation</code>	调制类型。
<code>mindspore.dataset.audio.NormMode</code>	标准化模式。
<code>mindspore.dataset.audio.NormType</code>	标准化类型。
<code>mindspore.dataset.audio.ResampleMethod</code>	重采样方法。
<code>mindspore.dataset.audio.ScaleType</code>	音频标度枚举类。
<code>mindspore.dataset.audio.WindowType</code>	窗函数类型。
<code>mindspore.dataset.audio.create_dct</code>	创建一个 <code>shape</code> 为 <code>(n_mels, n_mfcc)</code> 的 DCT 变换矩阵，并根据范数进行标准化。
<code>mindspore.dataset.audio.linear_fbanks</code>	创建一个线性三角滤波器组。
<code>mindspore.dataset.audio.melscale_fbanks</code>	创建频率变换矩阵。

mindspore.dataset.audio.BorderType

class mindspore.dataset.audio.BorderType

填充模式。

可选的枚举值包括：BorderType.CONSTANT、BorderType.EDGE、BorderType.REFLECT 和 BorderType.SYMMETRIC。

- **BorderType.CONSTANT** - 使用常量值填充。
- **BorderType.EDGE** - 使用各边的边界像素值填充。
- **BorderType.REFLECT** - 以各边的边界为轴进行镜像填充，忽略边界像素值。例如，向 [1, 2, 3, 4] 的两边分别填充 2 个元素，结果为 [3, 2, 1, 2, 3, 4, 3, 2]。
- **BorderType.SYMMETRIC** - 以各边的边界为轴进行对称填充，包括边界像素值。例如，向 [1, 2, 3, 4] 的两边分别填充 2 个元素，结果为 [2, 1, 1, 2, 3, 4, 4, 3]。

说明：该类派生自 `str` 以支持 JSON 可序列化。

mindspore.dataset.audio.DensityFunction

class mindspore.dataset.audio.DensityFunction

密度函数类型。

可选的枚举值包括：DensityFunction.TPDF、DensityFunction.RPDF 和 DensityFunction.GPDF。

- **DensityFunction.TPDF**：三角概率密度函数。
- **DensityFunction.RPDF**：矩形概率密度函数。
- **DensityFunction.GPDF**：高斯概率密度函数。

mindspore.dataset.audio.FadeShape

class mindspore.dataset.audio.FadeShape

淡入淡出形状。

可能的枚举值包括: FadeShape.QUARTER_SINE、FadeShape.HALF_SINE、FadeShape.LINEAR、FadeShape.LOGARITHMIC、FadeShape.EXPONENTIAL。

- **FadeShape.QUARTER_SINE**: 表示淡入淡出形状为四分之一正弦模式。
- **FadeShape.HALF_SINE**: 表示淡入形状为半正弦模式。
- **FadeShape.LINEAR**: 表示淡入淡出形状为线性模式。
- **FadeShape.LOGARITHMIC**: 表示淡入淡出形状为对数模式。
- **FadeShape.EXPONENTIAL**: 表示淡入淡出形状为指数模式。

mindspore.dataset.audio.GainType

class mindspore.dataset.audio.GainType

增益类型。

可选的枚举值包括: GainType.AMPLITUDE、GainType.POWER 和 GainType.DB。

- **GainType.AMPLITUDE** - 表示输入增益类型为振幅。
- **GainType.POWER** - 表示输入增益类型为功率。
- **GainType.DB** - 表示输入增益类型为分贝。

mindspore.dataset.audio.Interpolation

class mindspore.dataset.audio.Interpolation

音频波形的插值模式。

可选的枚举值包括: Interpolation.LINEAR 和 Interpolation.QUADRATIC。

- **Interpolation.LINEAR** - 插值模式为线性。
- **Interpolation.QUADRATIC** - 插值模式为二次型。

mindspore.dataset.audio.MelType

class mindspore.dataset.audio.MelType

梅尔标度实现类型。

可选的枚举值包括: MelType.HTK、MelType.SLANEY。

- **MelType.HTK** - 隐马尔可夫工具包 (HTK) 实现, 参考 [HTK](#)。
- **MelType.SLANEY** - MATLAB 听觉工具包的 Slaney 实现, 参考 [Auditory Toolbox](#)。

mindspore.dataset.audio.Modulation

class mindspore.dataset.audio.**Modulation**
调制类型。

可选的枚举值包括：Modulation.SINUSOIDAL、Modulation.TRIANGULAR。

- **Modulation.SINUSOIDAL** - 表示输入调制类型为正弦。
- **Modulation.TRIANGULAR** - 表示输入调制类型为三角形。

mindspore.dataset.audio.NormMode

class mindspore.dataset.audio.**NormMode**
标准化模式。

可选的枚举值包括：NormMode.ORTHO、NormMode.NONE。

- **NormMode.ORTHO** - 使用正交标准化的 DCT 基。
- **NormMode.NONE** - 不使用标准化。

mindspore.dataset.audio.NormType

class mindspore.dataset.audio.**NormType**
标准化类型。

可选的枚举值包括：NormType.SLANEY、NormType.NONE。

- **NormType.SLANEY** - 使用面积标准化。
- **NormType.None** - 不使用标准化。

mindspore.dataset.audio.ResampleMethod

class mindspore.dataset.audio.**ResampleMethod**
重采样方法。

可选的枚举值包括：ResampleMethod.SINC_INTERPOLATION、ResampleMethod.KAISER_WINDOW。

- **ResampleMethod.SINC_INTERPOLATION** - 香农插值或 sinc 插值公式。
- **ResampleMethod.KAISER_WINDOW** - 使用 kaiser 窗口重新采样。

mindspore.dataset.audio.ScaleType

class mindspore.dataset.audio.**ScaleType**
音频标度枚举类。

可选枚举值为：ScaleType.MAGNITUDE 和 ScaleType.POWER。

- **ScaleType.MAGNITUDE** - 表示输入音频的标度为振幅。
- **ScaleType.POWER** - 表示输入音频的标度为功率。

mindspore.dataset.audio.WindowType

class mindspore.dataset.audio.WindowType
窗函数类型。

可选的枚举值包括: WindowType.BARTLETT、WindowType.BLACKMAN、WindowType.HAMMING、WindowType.HANN、WindowType.KAISER。

- **WindowType.BARTLETT** - Bartlett 窗函数。
- **WindowType.BLACKMAN** - Blackman 窗函数。
- **WindowType.HAMMING** - Hamming 窗函数。
- **WindowType.HANN** - Hann 窗函数。
- **WindowType.KAISER** - Kaiser 窗函数。当前, 不支持在 macOS 上使用。

mindspore.dataset.audio.create_dct

mindspore.dataset.audio.create_dct(*n_mfcc*, *n_mels*, *norm*=NormMode.NONE)
创建一个 shape 为 (*n_mels*, *n_mfcc*) 的 DCT 变换矩阵, 并根据范数进行标准化。

参数:

- **n_mfcc** (int) - 要保留 mfc 系数的数量, 该值必须大于 0。
- **n_mels** (int) - mel 滤波器的数量, 该值必须大于 0。
- **norm** (*NormMode*, 可选) - 标准化模式, 可以是 NormMode.NONE 或 NormMode.ORTHO。默认值: NormMode.NONE。

返回:

numpy.ndarray, shape 为 (*n_mels*, *n_mfcc*) 的 DCT 转换矩阵。

异常:

- **TypeError** - 如果 *n_mfcc* 的类型不为 int。
- **ValueError** - 如果 *n_mfcc* 不为正数。
- **TypeError** - 如果 *n_mels* 的类型不为 int。
- **ValueError** - 如果 *n_mels* 不为正数。
- **TypeError** - 如果 *norm* 的类型不为 *mindspore.dataset.audio.NormType*。

支持平台:

CPU

样例:

```
>>> from mindspore.dataset.audio import create_dct, NormMode
>>>
>>> dct = create_dct(100, 200, NormMode.NONE)
```

mindspore.dataset.audio.linear_fbanks

`mindspore.dataset.audio.linear_fbanks(n_freqs, f_min, f_max, n_filter, sample_rate)`
创建一个线性三角滤波器组。

参数：

- **n_freqs** (int) - 要突出显示/应用的频率数。
- **f_min** (float) - 最小频率，单位赫兹。
- **f_max** (float) - 最大频率，单位赫兹。
- **n_filter** (int) - 线性三角滤波器数目。
- **sample_rate** (int) - 音频波形的采样率。

返回：

`numpy.ndarray`，线性三角滤波器组。

异常：

- **TypeError** - 如果 `n_freqs` 的类型不为 `int`。
- **ValueError** - 如果 `n_freqs` 为负数。
- **TypeError** - 如果 `f_min` 的类型不为 `float`。
- **ValueError** - 如果 `f_min` 为负数。
- **TypeError** - 如果 `f_max` 的类型不为 `float`。
- **ValueError** - 如果 `f_max` 为负数。
- **ValueError** - 如果 `f_min` 大于 `f_max`。
- **TypeError** - 如果 `n_filter` 的类型不为 `int`。
- **ValueError** - 如果 `n_filter` 不为正数。
- **TypeError** - 如果 `sample_rate` 的类型不为 `int`。
- **ValueError** - 如果 `sample_rate` 不为正数。

支持平台：

CPU

样例：

```
>>> from mindspore.dataset.audio import linear_fbanks
>>>
>>> fbanks = linear_fbanks(n_freqs=4096, f_min=0, f_max=8000, n_filter=40, sample_rate=16000)
```

mindspore.dataset.audio.melscale_fbanks

`mindspore.dataset.audio.melscale_fbanks` (*n_freqs*, *f_min*, *f_max*, *n_mels*, *sample_rate*, *norm=NormType.NONE*, *mel_type=MelType.HTK*)

创建频率变换矩阵。

参数：

- **n_freqs** (int) - 要加强或应用的频率数。
- **f_min** (float) - 最小频率，单位为 Hz。
- **f_max** (float) - 最大频率，单位为 Hz。
- **n_mels** (int) - 梅尔滤波器组数。
- **sample_rate** (int) - 音频波形的采样频率。
- **norm** (*NormType*, 可选) - 标准化方法，可以是 *NormType.NONE* 或 *NormType.SLANEY*。默认值：*NormType.NONE*。
- **mel_type** (*MelType*, 可选) - 使用的标度，可以是 *MelType.HTK* 或 *MelType.SLANEY*。默认值：*MelType.HTK*。

返回：

`numpy.ndarray`，频率变换矩阵，`shape` 为 (*n_freqs*, *n_mels*)。

异常：

- **TypeError** - 如果 *n_freqs* 的类型不为 `int`。
- **ValueError** - 如果 *n_freqs* 为负数。
- **TypeError** - 如果 *f_min* 的类型不为 `float`。
- **ValueError** - 如果 *f_min* 大于 *f_max*。
- **TypeError** - 如果 *f_max* 的类型不为 `float`。
- **ValueError** - 如果 *f_max* 为负数。
- **TypeError** - 如果 *n_mels* 的类型不为 `int`。
- **ValueError** - 如果 *n_mels* 不为正数。
- **TypeError** - 如果 *sample_rate* 的类型不为 `int`。
- **ValueError** - 如果 *sample_rate* 不为正数。
- **TypeError** - 如果 *norm* 的类型不为 `mindspore.dataset.audio.NormType`。
- **TypeError** - 如果 *mel_type* 的类型不为 `mindspore.dataset.audio.MelType`。

支持平台：

CPU

样例：

```
>>> from mindspore.dataset.audio import melscale_fbanks
>>>
>>> fbanks = melscale_fbanks(n_freqs=4096, f_min=0, f_max=8000, n_mels=40, sample_rate=16000)
```


MindRecord 是 MindSpore 提供的高效数据存储/读取模块，此模块提供了一些方法帮助用户将不同公开数据集转换为 MindRecord 格式，也提供了一些方法对 MindRecord 数据文件进行读取、写入、检索等。



MindSpore 格式数据可以更加方便地保存和加载数据，其目标是归一化用户的数据集，并针对不同数据场景进行了性能优化。使用 MindRecord 数据格式可以减少磁盘 IO、网络 IO 开销，从而获得更好的数据加载体验。

用户可以使用 `mindspore.mindrecord.FileWriter` 生成 MindRecord 格式数据文件，并使用 `mindspore.dataset.MindDataset` 加载 MindRecord 格式数据集。

用户还可以将其他格式数据集转换为 MindRecord 格式数据集，详见 [MindRecord 格式转换](#)。同时，MindRecord 还支持配置文件加密、解密和完整性校验，以保证 MindRecord 格式数据集的安全。

class `mindspore.mindrecord.FileWriter` (*file_name*, *shard_num=1*, *overwrite=False*)

将用户自定义的数据转为 MindRecord 格式数据集的类。

说明：生成 MindRecord 文件后，如果修改文件名，可能会导致读取文件失败。

参数：

- **file_name** (str) - 转换生成的 MindRecord 文件路径。
- **shard_num** (int, 可选) - 生成 MindRecord 的文件个数。取值范围为 [1, 1000]。默认值：1。
- **overwrite** (bool, 可选) - 当指定目录存在同名文件时是否覆盖写。默认值：False。

异常：

- **ParamValueError** - *file_name* 或 *shard_num* 无效。

样例:

```
>>> from mindspore.mindrecord import FileWriter
>>>
>>> writer = FileWriter(file_name="test.mindrecord", shard_num=1, overwrite=True)
>>> schema_json = {"file_name": {"type": "string"}, "label": {"type": "int32"}, "data": {
↪ "type": "bytes"}}
>>> writer.add_schema(schema_json, "test_schema")
>>> indexes = ["file_name", "label"]
>>> writer.add_index(indexes)
>>> for i in range(10):
...     data = [{"file_name": str(i) + ".jpg", "label": i,
...             "data": b"\x10c\xb3w\xa8\xee$o&q\x8c\x8e(\xa2\x90\x90\x96\xbc\xb1\x1e\
↪xd4QER\x13?\xff"}]
...     writer.write_raw_data(data)
>>> writer.commit()
```

add_index (*index_fields*)

指定 schema 中的字段作为索引来加速 MindRecord 文件的读取。schema 可以通过 *add_schema* 来添加。

说明:

- 索引字段应为 **Primitive** 类型, 例如 *int*、*float*、*str*。
- 如果不调用该函数, 则默认将 schema 中所有的 **Primitive** 类型的字段设置为索引。请参考 *mindspore.mindrecord.FileWriter* 类的样例代码。

参数:

- **index_fields** (list[str]) - schema 中的字段。

异常:

- **ParamTypeError** - 索引字段无效。
- **MRMDefineIndexError** - 索引字段不是 **Primitive** 类型。
- **MRMAddIndexError** - 无法添加索引字段。
- **MRMGetMetaError** - 未设置 schema 或无法获取 schema。

add_schema (*content*, *desc=None*)

增加描述用户自定义数据的 schema。

说明: 请参考 *mindspore.mindrecord.FileWriter* 类的样例代码。

表 1: MindRecord 支持的数据类型

数据类型	数据维度	说明
int32	/	整数
int64	/	整数
float32	/	浮点数
float64	/	浮点数
string	/	字符串
bytes	/	二进制
int32	[-1] / [-1, 32, 32] / [3, 224, 224]	numpy.ndarray 类型
int64	[-1] / [-1, 32, 32] / [3, 224, 224]	numpy.ndarray 类型
float32	[-1] / [-1, 32, 32] / [3, 224, 224]	numpy.ndarray 类型
float64	[-1] / [-1, 32, 32] / [3, 224, 224]	numpy.ndarray 类型

参数：

- **content** (dict) - schema 内容的字典。
- **desc** (str, 可选) - schema 的描述。默认值：None。

异常：

- **MRMInvalidSchemaError** - schema 无效。
- **MRMBuildSchemaError** - 构建 schema 失败。
- **MRMAddSchemaError** - 添加 schema 失败。

样例：

```
>>> # Examples of available schemas
>>> schema1 = {"file_name": {"type": "string"}, "label": {"type": "int32"}, "data": {
    ↪ "type": "bytes"}}
>>> schema2 = {"input_ids": {"type": "int32", "shape": [-1]},
...           "input_masks": {"type": "int32", "shape": [-1]}}
```

commit()

将内存中的数据同步到磁盘，并生成相应的数据库文件。

说明：请参考 `mindspore.mindrecord.FileWriter` 类的样例代码。

异常：

- **MRMOpenError** - 打开 MindRecord 文件失败。
- **MRMSetHeaderError** - 设置 MindRecord 文件的 header 失败。
- **MRMIndexGeneratorError** - 创建索引 Generator 失败。
- **MRMGenerateIndexError** - 写入数据库失败。
- **MRMCommitError** - 数据同步到磁盘失败。
- **RuntimeError** - 并行写失败。

classmethod open_for_append (*file_name*)

打开 MindRecord 文件, 准备追加数据。

参数:

- **file_name** (str) - MindRecord 格式的数据集文件的路径。

返回:

FileWriter, MindRecord 文件的写对象。

异常:

- **ParamValueError** - *file_name* 无效。
- **FileNameError** - MindRecord 文件路径中包含无效字符。
- **MRMOpenError** - 打开 MindRecord 文件失败。
- **MRMOpenForAppendError** - 打开 MindRecord 文件追加数据失败。

样例:

```
>>> from mindspore.mindrecord import FileWriter
>>>
>>> data = [{"file_name": "0.jpg", "label": 0,
...         "data": b"\x10c\xb3w\xa8\xee$o&<q\x8c\x8e(\xa2\x90\x90\x96\xbc\xb1\x1e\
↳xd4QER\x13?\xff"}]
>>> writer = FileWriter(file_name="test.mindrecord", shard_num=1, overwrite=True)
>>> schema_json = {"file_name": {"type": "string"}, "label": {"type": "int32"}, "data": {
↳"type": "bytes"}}
>>> writer.add_schema(schema_json, "test_schema")
>>> writer.write_raw_data(data)
>>> writer.commit()
>>>
>>> write_append = FileWriter.open_for_append("test.mindrecord")
>>> for i in range(9):
...     data = [{"file_name": str(i+1) + ".jpg", "label": i,
...             "data": b"\x10c\xb3w\xa8\xee$o&<q\x8c\x8e(\xa2\x90\x90\x96\xbc\xb1\x1e\
↳xd4QER\x13?\xff"}]
...     write_append.write_raw_data(data)
>>> write_append.commit()
```

set_header_size (*header_size*)

设置 MindRecord 文件的 header, 其中包含 shard 信息、schema 信息、page 的元信息等。header 越大, MindRecord 文件可以存储更多的元信息。如果 header 大于默认大小 (16MB), 需要调用本函数来设置合适的大小。

参数:

- **header_size** (int) - header 大小, 单位: 字节, 可设置范围为 16*1024(16KB) 到 128*1024*1024(128MB)。

异常:

- **MRMInvalidHeaderSizeError** - 设置 header 大小失败。

样例:

```
>>> from mindspore.mindrecord import FileWriter
>>> writer = FileWriter(file_name="test.mindrecord", shard_num=1)
>>> writer.set_header_size(1 << 25) # 32MB
```


set_page_size (*page_size*)

设置存储数据的 page 大小, page 分为两种类型: raw page 和 blob page。page 越大, page 可以存储更多的数据。如果单个样本大于默认大小 (32MB), 需要调用本函数来设置合适的大小。

参数:

- **page_size** (int) - page 大小, 单位: 字节, 可设置范围为 32*1024(32KB) 到 256*1024*1024(256MB)。

异常:

- **MRMInvalidPageSizeError** - 设置 page 大小失败。

样例:

```
>>> from mindspore.mindrecord import FileWriter
>>> writer = FileWriter(file_name="test.mindrecord", shard_num=1)
>>> writer.set_page_size(1 << 26) # 64MB
```

write_raw_data (*raw_data*, *parallel_writer=False*)

根据 schema 校验用户自定义数据后, 将数据转换为一系列连续的 MindRecord 格式的数据集文件。

说明: 请参考 `mindspore.mindrecord.FileWriter` 类的样例代码。

参数:

- **raw_data** (list[dict]) - 用户自定义数据的列表。
- **parallel_writer** (bool, 可选) - 如果为 True, 则并行写入用户自定义数据。默认值: False。

异常:

- **ParamTypeError** - 索引字段无效。
- **MRMOpenError** - 打开 MindRecord 文件失败。
- **MRMValidateDataError** - 数据校验失败。
- **MRMSetHeaderError** - 设置 MindRecord 文件的 header 失败。
- **MRMWriteDatasetError** - 写入 MindRecord 格式的数据集失败。
- **TypeError** - 参数 *parallel_writer* 不是 bool 类型。

class mindspore.mindrecord.**FileReader** (*file_name*, *num_consumer=4*, *columns=None*, *operator=None*)

读取 MindRecord 格式数据集。

说明:

- 如果 *file_name* 是文件路径的字符串, 则会尝试加载同一批转换生成的所有 MindRecord 文件, 如果缺少其中某个 MindRecord 文件, 则会引发异常。
- 如果 *file_name* 是文件路径组成的列表, 则只加载列表中指定的 MindRecord 文件。
- 参数 *operator* 未发挥任何作用, 会逐步废弃。

参数:

- **file_name** (str, list[str]) - MindRecord 格式的数据集文件路径或文件路径组成的列表。
- **num_consumer** (int, 可选) - 加载数据的并发数。默认值: 4。不应小于 1 或大于处理器的核数。

- **columns** (list[str], 可选) - MindRecord 中待读取数据列的列表。默认值: None , 读取所有的数据列。
- **operator** (int, 可选) - 保留参数。默认值: None 。

异常:

- **ParamValueError** - *file_name* 、 *num_consumer* 或 *columns* 无效。

样例:

```
>>> from mindspore.mindrecord import FileReader
>>>
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> reader = FileReader(file_name=mindrecord_file)
>>>
>>> # create iterator for mindrecord and get saved data
>>> for _, item in enumerate(reader.get_next()):
...     ori_data = item
>>> reader.close()
```

close()

停止数据集加载并且关闭文件句柄。

说明: 请参考 *mindspore.mindrecord.FileReader* 类的样例代码。

get_next()

按列名一次返回下一批的数据。

说明: 请参考 *mindspore.mindrecord.FileReader* 类的样例代码。

返回:

dict, 下一批数据, 键值与数据列名相同。

异常:

- **MRMUnsupportedSchemaError** - 当 schema 无效。

len()

返回当前 MindRecord 文件的样本个数。

返回:

int, 样本个数。

样例:

```
>>> from mindspore.mindrecord import FileReader
>>>
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> reader = FileReader(file_name=mindrecord_file)
>>> length = reader.len()
>>> reader.close()
```

schema()

返回当前 MindRecord 文件的 Schema 信息。

返回:

dict, Schema 信息。

样例:

```
>>> from mindspore.mindrecord import FileReader
>>>
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> reader = FileReader(file_name=mindrecord_file)
>>> schema = reader.schema()
>>> reader.close()
```

class mindspore.mindrecord.MindPage (file_name, num_consumer=4)

以分页方式读取 MindRecord 文件的类。

参数:

- **file_name** (Union[str, list[str]]) - MindRecord 格式的数据集文件或文件列表。
- **num_consumer** (int, 可选) - 加载数据的并发数。默认值: 4。不应小于 1 或大于处理器的核数。

异常:

- **ParamValueError** - file_name 不是 str 类型或 list[str] 类型。
- **ParamValueError** - num_consumer 不是 int 类型。

样例:

```
>>> from mindspore.mindrecord import MindPage
>>>
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> mind_page = MindPage(mindrecord_file)
>>>
>>> # get all the index fields
>>> fields = mind_page.candidate_fields
>>>
>>> # set the field to be retrieved
>>> mind_page.category_field = "file_name"
>>>
>>> # get all the group info
>>> info = mind_page.read_category_info()
>>>
>>> # get the row by id which is from category info
>>> row_by_id = mind_page.read_at_page_by_id(0, 0, 1)
>>>
>>> # get the row by name which is from category info
>>> row_by_name = mind_page.read_at_page_by_name("8.jpg", 0, 1)
```

property candidate_fields

返回用于数据分组的候选 category 字段。

说明: 请参考 `mindspore.mindrecord.MindPage` 类的样例代码。

返回:

list[str], 候选 category 字段。

property category_field

设置或者返回用于数据分组的 category 字段。

说明: 请参考`mindspore.mindrecord.MindPage` 类的样例代码。

返回:

`list[str]`, `category` 字段。

read_at_page_by_id (*category_id*, *page*, *num_row*)

以分页方式按 `category ID` 进行查询。

说明: 请参考`mindspore.mindrecord.MindPage` 类的样例代码。

参数:

- **category_id** (int) - category ID, 参考 `read_category_info` 函数的返回值。
- **page** (int) - 分页的索引。
- **num_row** (int) - 每个分页的行数。

返回:

`list[dict]`, 根据 `category ID` 查询的数据。

异常:

- **ParamValueError** - 参数无效。
- **MRMFetchDataError** - 无法按 `category ID` 获取数据。
- **MRMUnsupportedSchemaError** - schema 无效。

read_at_page_by_name (*category_name*, *page*, *num_row*)

以分页方式按 `category` 字段进行查询。

说明: 请参考`mindspore.mindrecord.MindPage` 类的样例代码。

参数:

- **category_name** (str) - category 字段对应的字符, 参考 `read_category_info` 函数的返回值。
- **page** (int) - 分页的索引。
- **num_row** (int) - 每个分页的行数。

返回:

`list[dict]`, 根据 `category` 字段查询的数据。

read_category_info ()

当数据按指定的 `category` 字段进行分组时, 返回 `category` 信息。

返回结果类似如下, 其中 `key` 表示索引字段名, `categories` 代表针对索引的统计信息。

```
{ "categories": [ { "count": 1, "id": 0, "name": "0.jpg" },
                  { "count": 1, "id": 1, "name": "1.jpg" },
                  { "count": 1, "id": 2, "name": "2.jpg" },
                  { "count": 1, "id": 3, "name": "3.jpg" } ],
  "key": "file_name_0" }
```

说明: 请参考 `mindspore.mindrecord.MindPage` 类的样例代码。

返回:

`str`, 分组信息的描述。

异常:

- **MRMReadCategoryInfoError** - 读取 category 信息失败。

class `mindspore.mindrecord.Cifar10ToMR` (*source*, *destination*)

将 CIFAR-10 数据集 (需要是 Python 版本的, 名字类似: `cifar-10-python.tar.gz`) 转换为 MindRecord 格式数据集。

参数:

- **source** (`str`) - 待转换的 CIFAR-10 数据集文件所在目录的路径。
- **destination** (`str`) - 转换生成的 MindRecord 文件路径, 需提前创建目录并且目录下不能存在同名文件。

异常:

- **ValueError** - *source* 或 *destination* 无效。

样例:

```
>>> from mindspore.mindrecord import Cifar10ToMR
>>>
>>> cifar10_dir = "/path/to/cifar10"
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> cifar10_to_mr = Cifar10ToMR(cifar10_dir, mindrecord_file)
>>> cifar10_to_mr.transform()
```

transform (*fields=None*)

执行从 CIFAR-10 数据集到 MindRecord 格式数据集的转换。

说明: 请参考 `mindspore.mindrecord.Cifar10ToMR` 类的样例代码。

参数:

- **fields** (`list[str]`, 可选) - 索引字段的列表。默认值: `None`。索引字段的设置请参考函数 `mindspore.mindrecord.FileWriter.add_index()`。

异常:

- **ParamTypeError** - 设置 MindRecord 索引字段失败。
- **MRMOpenError** - 新建 MindRecord 文件失败。
- **MRMValidateDataError** - 原始数据集数据异常。
- **MRMSetHeaderError** - 设置 MindRecord 文件头失败。
- **MRMWriteDatasetError** - 创建 MindRecord 索引失败。

- **ValueError** - 参数 *fields* 不合法。

class mindspore.mindrecord.Cifar100ToMR(*source, destination*)

将 CIFAR-100 数据集（需要是 Python 版本的，名字类似：cifar-100-python.tar.gz）转换为 MindRecord 格式数据集。

参数：

- **source** (str) - 待转换的 CIFAR-100 数据集文件所在目录的路径。
- **destination** (str) - 转换生成的 MindRecord 文件路径，需提前创建目录并且目录下不能存在同名文件。

异常：

- **ValueError** - 参数 *source* 或 *destination* 无效。

样例：

```
>>> from mindspore.mindrecord import Cifar100ToMR
>>>
>>> cifar100_dir = "/path/to/cifar100"
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> cifar100_to_mr = Cifar100ToMR(cifar100_dir, mindrecord_file)
>>> cifar100_to_mr.transform()
```

transform (*fields=None*)

执行从 CIFAR-100 数据集到 MindRecord 格式数据集的转换。

说明：请参考 `mindspore.mindrecord.Cifar100ToMR` 类的样例代码。

参数：

- **fields** (list[str], 可选) - 索引字段的列表，例如 ['fine_label', 'coarse_label']。默认值：None。索引字段的设置请参考函数 `mindspore.mindrecord.FileWriter.add_index()`。

异常：

- **ParamTypeError** - 设置 MindRecord 索引字段失败。
- **MRMOpenError** - 新建 MindRecord 文件失败。
- **MRMValidateDataError** - 原始数据集数据异常。
- **MRMSetHeaderError** - 设置 MindRecord 文件头失败。
- **MRMWriteDatasetError** - 创建 MindRecord 索引失败。
- **ValueError** - 参数 *fields* 不合法。

class mindspore.mindrecord.CsvToMR(*source, destination, columns_list=None, partition_number=1*)

将 CSV 格式数据集转换为 MindRecord 格式的数据集。

参数：

- **source** (str) - 待转换的 CSV 文件路径。
- **destination** (str) - 转换生成的 MindRecord 文件路径，需提前创建目录并且目录下不能存在同名文件。
- **columns_list** (list[str], 可选) - CSV 中待读取数据列的列表。默认值：None，读取所有的数据列。
- **partition_number** (int, 可选) - 生成 MindRecord 的文件个数。默认值：1。

异常：

- **ValueError** - 参数 *source*、*destination*、*partition_number* 无效。
- **RuntimeError** - 参数 *columns_list* 无效。

样例:

```
>>> from mindspore.mindrecord import CsvToMR
>>>
>>> csv_file = "/path/to/csv/file"
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> csv_to_mr = CsvToMR(csv_file, mindrecord_file)
>>> csv_to_mr.transform()
```

transform()

执行从 CSV 格式数据集到 MindRecord 格式数据集的转换。

说明: 请参考 `mindspore.mindrecord.CsvToMR` 类的样例代码。

异常:

- **ParamTypeError** - 设置 MindRecord 索引字段失败。
- **MRMOpenError** - 新建 MindRecord 文件失败。
- **MRMValidateDataError** - 原始数据集数据异常。
- **MRMSetHeaderError** - 设置 MindRecord 文件头失败。
- **MRMWriteDatasetError** - 创建 MindRecord 索引失败。
- **IOError** - 参数 *source* 不存在。
- **ValueError** - CSV 文件首行为列名, 每个字段不能以数字开头。

class mindspore.mindrecord.**ImageNetToMR** (*map_file*, *image_dir*, *destination*, *partition_number*=1)

将 ImageNet 数据集转换为 MindRecord 格式数据集。

参数:

- **map_file** (str) - 标签映射文件的路径。该文件可通过命令: `ls -l [image_dir] | grep -vE " 总用量 |total|\." | awk -F " " '{print $9, NR-1;}' > [file_path]` 生成, 其中 *image_dir* 为 ImageNet 数据集的目录路径, *file_path* 为生成的 *map_file* 文件。*map_file* 文件内容示例如下:

```
n01440764 0
n01443537 1
n01484850 2
n01491361 3
...
n15075141 999
```

- **image_dir** (str) - ImageNet 数据集的目录路径, 目录中包含类似 n01440764、n01443537、n01484850 和 n15075141 的子目录。
- **destination** (str) - 转换生成的 MindRecord 文件路径, 需提前创建目录并且目录下不能存在同名文件。
- **partition_number** (int, 可选) - 生成 MindRecord 的文件个数。默认值: 1。

异常:

- **ValueError** - 参数 *map_file*、*image_dir* 或 *destination* 无效。

样例:

```
>>> from mindspore.mindrecord import ImageNetToMR
>>>
>>> map_file = "/path/to/imagenet/map_file"
>>> imagenet_dir = "/path/to/imagenet/train"
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> imagenet_to_mr = ImageNetToMR(map_file, imagenet_dir, mindrecord_file, 8)
>>> imagenet_to_mr.transform()
```

transform()

执行从 ImageNet 数据集到 MindRecord 格式数据集的转换。

说明: 请参考 `mindspore.mindrecord.ImageNetToMR` 类的样例代码。

异常:

- **ParamTypeError** - 设置 MindRecord 索引字段失败。
- **MRMOpenError** - 新建 MindRecord 文件失败。
- **MRMValidateDataError** - 原始数据集数据异常。
- **MRMSetHeaderError** - 设置 MindRecord 文件头失败。
- **MRMWriteDatasetError** - 创建 MindRecord 索引失败。

class mindspore.mindrecord.**MnistToMR** (*source, destination, partition_number=1*)

将 MNIST 数据集转换为 MindRecord 格式数据集。

参数:

- **source** (str) - 数据集目录路径, 其包含 t10k-images-idx3-ubyte.gz、train-images-idx3-ubyte.gz、t10k-labels-idx1-ubyte.gz 和 train-labels-idx1-ubyte.gz 数据集文件。
- **destination** (str) - 转换生成的 MindRecord 文件路径, 需提前创建目录并且目录下不能存在同名文件。
- **partition_number** (int, 可选) - 生成 MindRecord 的文件个数。默认值: 1。

异常:

- **ValueError** - 参数 *source*、*destination*、*partition_number* 无效。

样例:

```
>>> from mindspore.mindrecord import MnistToMR
>>>
>>> mnist_dir = "/path/to/mnist"
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> mnist_to_mr = MnistToMR(mnist_dir, mindrecord_file)
>>> mnist_to_mr.transform()
```

transform()

执行从 MNIST 数据集到 MindRecord 格式数据集的转换。

说明: 请参考 `mindspore.mindrecord.MnistToMR` 类的样例代码。

异常:

- **ParamTypeError** - 设置 MindRecord 索引字段失败。

- **MRMOpenError** - 新建 MindRecord 文件失败。
- **MRMValidateDataError** - 原始数据集数据异常。
- **MRMSetHeaderError** - 设置 MindRecord 文件头失败。
- **MRMWriteDatasetError** - 创建 MindRecord 索引失败。

class mindspore.mindrecord.TFRecordToMR(*source, destination, feature_dict, bytes_fields=None*)

将 TFRecord 格式数据集转换为 MindRecord 格式数据集。

参数：

- **source** (str) - 待转换的 TFRecord 文件路径。
- **destination** (str) - 转换生成的 MindRecord 文件路径，需提前创建目录并且目录下不能存在同名文件。
- **feature_dict** (dict[str, FixedLenFeature]) - TFRecord 的 feature 类别的字典，当前支持 **FixedLenFeature** 类型。
- **bytes_fields** (list[str], 可选) - *feature_dict* 中的字节字段，可以为字节类型的图像字段。默认值：None，表示没有诸如图像的二进制字段。

异常：

- **ValueError** - 无效参数。
- **Exception** - 找不到 TensorFlow 模块或其版本不正确。

样例：

```
>>> from mindspore.mindrecord import TFRecordToMR
>>> import tensorflow as tf
>>>
>>> tfrecord_file = "/path/to/tfrecord/file"
>>> mindrecord_file = "/path/to/mindrecord/file"
>>> feature_dict = {"file_name": tf.io.FixedLenFeature([], tf.string),
...                 "image_bytes": tf.io.FixedLenFeature([], tf.string),
...                 "int64_scalar": tf.io.FixedLenFeature([], tf.int64),
...                 "float_scalar": tf.io.FixedLenFeature([], tf.float32),
...                 "int64_list": tf.io.FixedLenFeature([6], tf.int64),
...                 "float_list": tf.io.FixedLenFeature([7], tf.float32)}
>>> tfrecord_to_mr = TFRecordToMR(tfrecord_file, mindrecord_file, feature_dict, ["image_bytes"])
>>> tfrecord_to_mr.transform()
```

transform()

执行从 TFRecord 格式数据集到 MindRecord 格式数据集的转换。

说明：请参考 `mindspore.mindrecord.TFRecordToMR` 类的样例代码。

异常：

- **ParamTypeError** - 设置 MindRecord 索引字段失败。
- **MRMOpenError** - 新建 MindRecord 文件失败。
- **MRMValidateDataError** - 原始数据集数据异常。
- **MRMSetHeaderError** - 设置 MindRecord 文件头失败。
- **MRMWriteDatasetError** - 创建 MindRecord 索引失败。

`mindspore.mindrecord.set_enc_key(enc_key)`

设置 MindRecord 数据格式的加密密钥。

参数:

- **enc_key** (str) - 用于加密的密钥, 有效长度为 16、24 或者 32。其中: None 代表不启用加密。

异常:

- **ValueError** - 参数 `enc_key` 不是字符串或者长度错误。

样例:

```
>>> from mindspore.mindrecord import set_enc_key
>>>
>>> set_enc_key("0123456789012345")
```

`mindspore.mindrecord.set_enc_mode(enc_mode='AES-GCM')`

设置 MindRecord 数据格式的加密算法。

参数:

- **enc_mode** (Union[str, function], 可选) - 指定加密模式, 当设置 `enc_key` 时启用。支持的加密选项有: "AES-GCM" 和用户自定义加密算法。默认值: "AES-GCM"。如果是自定义加密, 用户需要自己保证加密正确性及加密算法的安全性, 并且在出错时报异常。

异常:

- **ValueError** - 参数 `enc_mode` 无效或者自定义加密不能被调用执行。

样例:

```
>>> from mindspore.mindrecord import set_enc_mode
>>>
>>> set_enc_mode("AES-GCM")
```

`mindspore.mindrecord.set_dec_mode(dec_mode='AES-GCM')`

设置 MindRecord 数据格式的解密算法。

如果使用的是内置 `enc_mode`, 且没有指定 `dec_mode`, 则使用 `enc_mode` 指定的加密算法进行解密。如果使用自定义加密函数, 则必须在读取时指定自定义解密函数。

参数:

- **dec_mode** (Union[str, function], 可选) - 指定解密模式, 当设置 `enc_key` 时启用。支持的解密选项有: "AES-GCM" 和用户自定义解密算法。默认值: "AES-GCM"。其中: None 代表不设置解密算法。如果是自定义解密, 用户需要自己保证解密正确性, 并且在出错时报异常。

异常:

- **ValueError** - 参数 `dec_mode` 无效或者自定义解密不能被调用执行。

样例:

```
>>> from mindspore.mindrecord import set_dec_mode
>>>
>>> set_dec_mode("AES-GCM")
```

MindSpore NumPy 工具包提供了一系列类 NumPy 接口。用户可以使用类 NumPy 语法在 MindSpore 上进行模型的搭建。

MindSpore Numpy 具有四大功能模块：Array 生成、Array 操作、逻辑运算和数学运算。

在 API 示例中，常用的模块导入方法如下：

```
import mindspore.numpy as np
```

说明：MindSpore numpy 通过组装底层算子来提供与 numpy 一致的编程体验接口，方便开发人员使用和代码移植。相比于 MindSpore 的 function 和 ops 接口，与原始 numpy 的接口格式及行为一致性更好，以便于用户理解和使用。注意：由于兼容 numpy 的考虑，部分接口的性能可能弱于 function 和 ops 接口。使用者可以按需选择不同类型的接口。

15.1 Array 生成

生成类算子用来生成和构建具有指定数值、类型和形状的数组 (Tensor)。

构建数组代码示例：

```
import mindspore.numpy as np
import mindspore.ops as ops
input_x = np.array([1, 2, 3], np.float32)
print("input_x =", input_x)
print("type of input_x =", ops.typeof(input_x))
```

运行结果如下：

```
input_x = [1. 2. 3.]
type of input_x = Tensor[Float32]
```

除了使用上述方法来创建外，也可以通过以下几种方式创建。

- 生成具有相同元素的数组

生成具有相同元素的数组代码示例：

```
input_x = np.full((2, 3), 6, np.float32)
print(input_x)
```

运行结果如下：

```
[[6. 6. 6.]
 [6. 6. 6.]]
```

生成指定形状的全 1 数组，示例：

```
input_x = np.ones((2, 3), np.float32)
print(input_x)
```

运行结果如下：

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

- 生成具有某个范围内的数值的数组

生成指定范围内的等差数组代码示例：

```
input_x = np.arange(0, 5, 1)
print(input_x)
```

运行结果如下：

```
[0 1 2 3 4]
```

- 生成特殊类型的数组

生成给定对角线处下方元素为 1，上方元素为 0 的矩阵，示例：

```
input_x = np.tri(3, 3, 1)
print(input_x)
```

运行结果如下：

```
[[1. 1. 0.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

生成对角线为 1，其他元素为 0 的二维矩阵，示例：

```
input_x = np.eye(2, 2)
print(input_x)
```

运行结果如下：

```
[[1. 0.]
 [0. 1.]]
```

接口名	概述	支持平台

[下页继续](#)

表 1 - 续上页

<code>mindspore.numpy.arange</code>	返回给定区间内均匀间隔的值。	Ascend GPU CPU
<code>mindspore.numpy.array</code>	该函数接受一个类似数组的对象创建 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.asarray</code>	该函数将一个类似数组的对象转换为 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.asfarray</code>	类似于 <code>asarray</code> ，将输入转换为 float tensor 。	Ascend GPU CPU
<code>mindspore.numpy.bartlett</code>	用于生成 Bartlett 窗口。	Ascend GPU CPU
<code>mindspore.numpy.blackman</code>	用于生成 Blackman 窗口。	Ascend GPU CPU
<code>mindspore.numpy.copy</code>	返回给定对象的 Tensor 副本。	Ascend GPU CPU
<code>mindspore.numpy.diag</code>	用于提取或构造对角线数组。	Ascend GPU CPU
<code>mindspore.numpy.diag_indices</code>	返回一个可以访问数组的主对角线的索引数组。	Ascend GPU CPU
<code>mindspore.numpy.diagflat</code>	返回一个二维数组，其数组输入作为新输出数组的对角线。	Ascend GPU CPU
<code>mindspore.numpy.diagonal</code>	返回数组指定的对角线。	Ascend GPU CPU
<code>mindspore.numpy.empty</code>	返回一个给定 shape 和类型的新数组，而不进行初始化。	Ascend GPU CPU
<code>mindspore.numpy.empty_like</code>	返回一个 shape 和类型与给定数组相同的新数组。	Ascend GPU CPU
<code>mindspore.numpy.eye</code>	返回一个对角线上值为 1，其他位置为 0 的二维 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.full</code>	返回一个给定 shape 、类型，并用 <code>fill_value</code> 填充的新数组。	Ascend GPU CPU
<code>mindspore.numpy.full_like</code>	返回一个与给定数组具有相同 shape 和类型的完整数组。	Ascend GPU CPU
<code>mindspore.numpy.geomspace</code>	返回在对数刻度（几何级数）上均匀间隔的数字。	Ascend GPU CPU
<code>mindspore.numpy.hamming</code>	返回一个 Hamming 窗口函数。	Ascend GPU CPU
<code>mindspore.numpy.hanning</code>	返回一个 Hanning 窗口函数。	Ascend GPU CPU
<code>mindspore.numpy.histogram_bin_edges</code>	计算 histogram 函数需要使用的 bins 的边界值。	Ascend GPU CPU
<code>mindspore.numpy.identity</code>	返回单位数组。	Ascend GPU CPU
<code>mindspore.numpy.indices</code>	返回一个表示网格索引的数组。	Ascend GPU CPU
<code>mindspore.numpy.ix_</code>	从多个序列构建一个开放式网格，用于索引的坐标数组。	Ascend GPU CPU
<code>mindspore.numpy.linspace</code>	返回给定区间内均匀间隔的值。	Ascend GPU CPU
<code>mindspore.numpy.logspace</code>	返回在对数刻度上均匀间隔的值。	Ascend GPU CPU
<code>mindspore.numpy.meshgrid</code>	返回由坐标向量生成的坐标矩阵。	Ascend GPU CPU
<code>mindspore.numpy.mgrid</code>	返回一个密集矩阵 NdGrid 实例，其中 <code>sparse=False</code> 。	Ascend GPU CPU
<code>mindspore.numpy.ogrid</code>	返回一个稀疏矩阵 NdGrid 实例，其中 <code>sparse=True</code> 。	Ascend GPU CPU
<code>mindspore.numpy.ones</code>	返回一个给定 shape 和类型的新 Tensor ，其中所有元素用 1 来填充。	Ascend GPU CPU
<code>mindspore.numpy.ones_like</code>	返回一个与给定数组 <code>a</code> 具有相同 shape 和类型的 Tensor ，其中所有元素用 1 来填充。	Ascend GPU CPU
<code>mindspore.numpy.pad</code>	对矩阵进行填充。	Ascend GPU CPU
<code>mindspore.numpy.rand</code>	返回一个给定 shape 和类型的新 Tensor ，其中所有元素以区间 <code>[0, 1)</code> 上均匀分布的随机数来填充。	Ascend GPU CPU
<code>mindspore.numpy.randint</code>	返回从 <code>minval</code> （包括）到 <code>maxval</code> （不包括）的随机整数。	Ascend GPU CPU

下页继续

表 1 - 续上页

<code>mindspore.numpy.randn</code>	返回一个给定 shape 和类型的新 Tensor ，并填充来自标准正态分布中的一个（或多个）样本。	Ascend GPU CPU
<code>mindspore.numpy.trace</code>	返回张量沿对角线的元素之和。	Ascend GPU CPU
<code>mindspore.numpy.tri</code>	返回一个 Tensor ，在给定的对角线处及以下元素值为 1，在其他位置为 0。	Ascend GPU CPU
<code>mindspore.numpy.tril</code>	返回张量的下三角部分。	Ascend GPU CPU
<code>mindspore.numpy.tril_indices</code>	返回 shape 为 (n, m) 数组的下三角的索引。	Ascend GPU CPU
<code>mindspore.numpy.tril_indices_from</code>	返回 arr 下三角的索引。	Ascend GPU CPU
<code>mindspore.numpy.triu</code>	返回张量的上三角部分。	Ascend GPU CPU
<code>mindspore.numpy.triu_indices</code>	返回 shape 为 (n, m) 数组的上三角的索引。	Ascend GPU CPU
<code>mindspore.numpy.triu_indices_from</code>	返回 arr 上三角的索引。	Ascend GPU CPU
<code>mindspore.numpy.vander</code>	生成一个范德蒙德矩阵。	Ascend GPU CPU
<code>mindspore.numpy.zeros</code>	返回一个给定 shape 和类型的新 Tensor ，其中所有元素以 0 来填充。	Ascend GPU CPU
<code>mindspore.numpy.zeros_like</code>	返回一个与给定数组 a 具有相同 shape 和类型的 Tensor ，其中所有元素用 0 来填充。	Ascend GPU CPU

15.1.1 mindspore.numpy.arange

`mindspore.numpy.arange` (*start*, *stop=None*, *step=None*, *dtype=None*)

返回给定区间内均匀间隔的值。

参数：

- **start** (Union[int, float]) - 区间的起始值，区间包含该值。如果希望以位置参数处理入参 *stop*，那么 *start* 是必选参数；否则，*start* 是可选参数，默认值：0。
- **stop** (Union[int, float], 可选) - 用于指定生成的数组中的终止值，不包括该值本身（除非在某些情况下 *step* 不是整数并且浮点数的舍入会影响输出的数组长度。）。
- **step** (Union[int, float], 可选) - 用于指定数组中连续的两个值之间的间距。如果未指定 *step* 参数，数组中的元素间距默认值：1。
- **dtype** (Union[mindspore.dtype, str], 可选) - 如果指定了该参数，则生成数组的数据类型将采用给定的数据类型。如果未指定，则生成数组的数据类型将根据提供的参数自动推断出来。默认值：None。

返回：

Tensor，给定区间范围内以均匀间隔生成的值。

异常：

- **TypeError(PyNative Mode)** - 如果输入的参数类型没有按上述内容指定，或者没有按照上述内容给定的顺序输入参数。
- **RuntimeError(Graph Mode)** - 在 Pynative 模式中导致 **TypeError** 的输入将在 Graph 模式中导致 **RuntimeError**。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.arange(0, 5, 1))
[0 1 2 3 4]
>>> print(np.arange(3))
```

(下页继续)

(续上页)

```
[0 1 2]
>>> print(np.arange(start=0, stop=3))
[0 1 2]
>>> print(np.arange(0, stop=3, step=0.5))
[0.  0.5 1.  1.5 2.  2.5]
```

15.1.2 mindspore.numpy.array

`mindspore.numpy.array(obj, dtype=None, copy=True, ndmin=0)`

该函数接受一个类似数组的对象创建 Tensor。

参数：

- **obj** (Union[int, float, bool, list, tuple]) - 可以转换为 Tensor 的任何形式的输入数据。包含 int, float, bool, Tensor, list, tuple。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor 数据类型可以是 np.int32 或 int32。如果 dtype 为 None, 则将从 obj 推断出新 Tensor 的数据类型。默认值: None。
- **copy** (bool) - 如果为 True, 输入的对象会被拷贝; 如果为 False, 只有在需要时对象才会被拷贝, 默认值: True。
- **ndmin** (int) - 用于指定输出的 Tensor 应具有的最小维度数, 将根据需要对其 shape 进行预处理。默认值: 0。

返回：

Tensor, 具有指定数据类型。

异常：

- **TypeError** - 如果没有按照上述内容给定的数据类型输入参数。
- **ValueError** - 如果输入 obj 在不同的维度上具有不同的大小。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.array([1, 2, 3]))
[1 2 3]
```

15.1.3 mindspore.numpy.asarray

`mindspore.numpy.asarray(a, dtype=None)`

该函数将一个类似数组的对象转换为 Tensor。

参数：

- **a** (Union[int, float, bool, list, tuple, Tensor]) - 可以转换为 Tensor 的任何形式的输入数据。包含 int, float, bool, Tensor, list, tuple。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor 的数据类型, 可以是 np.int32 或 int32。如果 dtype 为 None, 则将从 a 推断出新 Tensor 的数据类型。默认值: None。

返回：

Tensor, 具有指定数据类型。

异常:

- **TypeError** - 如果没有按照上述内容给定的数据类型输入参数。
- **ValueError** - 如果输入 *a* 在不同的维度上具有不同的大小。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.asarray([1,2,3]))
[1 2 3]
```

15.1.4 mindspore.numpy.asfarray

`mindspore.numpy.asfarray(a, dtype=mstype.float32)`

类似于 `asarray`, 将输入转换为 `float tensor`。如果 `dtype` 定义的是一个 `non-float` 格式, 则该函数将返回 `float32` 格式的 `Tensor`。

参数:

- **a** (`Union[int, float, bool, list, tuple, Tensor]`) - 可以转换为 `Tensor` 的任何形式的输入数据。包含 `int`, `float`, `bool`, `Tensor`, `list`, `tuple`。
- **dtype** (`Union[mindspore.dtype, str]`, 可选) - 指定 `Tensor` 的数据类型, 可以是 `np.int32` 或 `int32`。如果 `dtype` 为 `None`, 则将从 `a` 推断出新 `Tensor` 的数据类型。默认值: `mstype.float32`。

返回:

`Tensor`, 具有指定的 `float` 数据类型。

异常:

- **TypeError** - 如果没有按照上述内容给定的数据类型输入参数。
- **ValueError** - 如果输入 *a* 在不同的维度上具有不同的大小。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.asfarray([1,2,3]))
[1. 2. 3.]
```


15.1.5 mindspore.numpy.bartlett

`mindspore.numpy.bartlett(M)`

用于生成 Bartlett 窗口。Bartlett 窗口是一种特殊的三角形窗口函数，它在信号处理和频谱分析中经常使用。Bartlett 窗口在时域上是对称的，幅度逐渐减小。

参数：

- **M (int)** - 输出窗口的长度。如果为零或小于零，则返回一个空数组。

返回：

Tensor，生成一个长度为 M 的 Bartlett 窗口，其中它的第一个值和最后一个值为 0，中间部分为尖峰，即三角形形状（仅当样本数为奇数时才会出现值 1）。

异常：

- **TypeError** - 如果输入 M 不是一个 int 型数据。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.bartlett(12))
[0.          0.18181819 0.36363637 0.5454545  0.72727275 0.9090909
 0.9090909  0.72727275 0.5454545  0.36363637 0.18181819 0.]
```

15.1.6 mindspore.numpy.blackman

`mindspore.numpy.blackman(M)`

用于生成 Blackman 窗口。Blackman 窗口使用前三项的余弦和计算得到，其设计是为了尽可能减少泄露。它接近于最优，仅次于 Kaiser 接口。

参数：

- **M (int)** - 输出窗口的长度。如果为零或小于零，则返回一个空数组。

返回：

Tensor，生成一个长度为 M 且最大值归一化为 1 的 Blackman 窗口（仅当样本数为奇数时才会出现值 1）。

异常：

- **TypeError** - 如果输入 M 不是一个 int 型数据。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.blackman(12))
[-1.4901161e-08  3.2606430e-02  1.5990365e-01  4.1439798e-01
 7.3604518e-01  9.6704674e-01  9.6704674e-01  7.3604518e-01
 4.1439798e-01  1.5990365e-01  3.2606430e-02 -1.4901161e-08]
```

15.1.7 mindspore.numpy.copy

`mindspore.numpy.copy(a)`

返回给定对象的 Tensor 副本。

参数:

- **a** (Union[int, float, bool, list, tuple, Tensor]) - 可以转换为 Tensor 的任何形式的输入数据。包含 int, float, bool, Tensor, list, tuple。

返回:

Tensor, 有与 a 相同的数据。

异常:

- **TypeError** - 如果没有按照上述内容给定的数据类型输入参数。
- **ValueError** - 如果输入 a 在不同的维度上具有不同的大小。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.ones((2,2))
>>> print(np.copy(x))
[[1. 1.]
 [1. 1.]]
```

15.1.8 mindspore.numpy.diag

`mindspore.numpy.diag(v, k=0)`

用于提取或构造对角线数组。

参数:

- **v** (Tensor) - 如果 v 是二维数组, 则返回其第 k 个对角线的副本。如果 v 是一维数组, 则返回一个 v 作为第 k 个对角线的二维数组。
- **k** (int, 可选) - 对角线的位置, $k > 0$ 即为对主对角线上方, $k < 0$ 即为主对角线下方。

返回:

Tensor, 返回该数组的对角线元素或者构造一个新的二维数组, 其中指定的一维数组为对角线元素。

异常:

- **ValueError** - 如果 v 不是一维或者二维数组。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.arange(9).reshape((3,3))
>>> print(x)
[[0 1 2]
```

(下页继续)

(续上页)

```
[3 4 5]
[6 7 8]]
>>> output = np.diag(x)
>>> print(output)
[0 4 8]
>>> output = np.diag(x, k=1)
>>> print(output)
[1 5]
>>> output = np.diag(x, k=-1)
>>> print(output)
[3 7]
```

15.1.9 mindspore.numpy.diag_indices

`mindspore.numpy.diag_indices(n, ndim=2)`

返回一个可以访问数组的主对角线的索引数组。该函数返回一个索引的元组，索引指向数组 `a` 的主对角线，数组 `a` 满足 `a.ndim >= 2` 与 `shape` 为 `(n, n, ..., n)`。如果满足 `a.ndim = 2`，该接口与 `diagonal` 功能一致；如果 `a.ndim > 2`，则返回访问 `a[i, i, ..., i]` 的索引集合，其中 `i = [0..n-1]`。

参数：

- **n** (int) - 可使用返回索引的数组沿每个维度的大小。
- **ndim** (int, 可选) - 维数。

返回：

元素为 `Tensor` 的 `Tuple`。

异常：

- **TypeError** - 如果没有按照上述内容给定的数据类型输入参数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.diag_indices(5, 3)
>>> print(output)
(Tensor(shape=[5], dtype=Int32, value= [0, 1, 2, 3, 4]),
 Tensor(shape=[5], dtype=Int32, value= [0, 1, 2, 3, 4]),
 Tensor(shape=[5], dtype=Int32, value= [0, 1, 2, 3, 4]))
```

15.1.10 mindspore.numpy.diagflat

`mindspore.numpy.diagflat(v, k=0)`

返回一个二维数组，其数组输入作为新输出数组的对角线。

说明：在 GPU 上，支持的数据类型是 `np.float16` 和 `np.float32`。

参数：

- **v** (Tensor) - 输入数据，将其平坦化并设置为输出的第 k 个对角线。
- **k** (int, 可选) - 需要的对角线；默认值：0 即主对角线， $k > 0$ 意味着对角线高于主对角线，反之亦然。

返回：

2-D Tensor。

异常：

- **TypeError** - 如果输入不是一个 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.diagflat(np.asarray([[1,2], [3,4]]))
>>> print(output)
[[1 0 0 0]
 [0 2 0 0]
 [0 0 3 0]
 [0 0 0 4]]
>>> output = np.diagflat(np.asarray([1,2]), 1)
>>> print(output)
[[0 1 0]
 [0 0 2]
 [0 0 0]]
```

15.1.11 mindspore.numpy.diagonal

`mindspore.numpy.diagonal(a, offset=0, axis1=0, axis2=1)`

返回数组指定的对角线。如果 a 是二维，则返回具有给定 `offset` 的 a 的对角线，形如 $a[i, i+offset]$ 的元素的集合。如果 a 具有两个以上的维度，则使用 `axis1` 和 `axis2` 指定的轴来确定返回对角线的二维子数组。可以通过移除 `axis1` 和 `axis2` 并在右边附加等于结果对角线大小的索引来确定结果数组的 `shape`。

参数：

- **a** (Tensor) - 输入数据，用于从中获取对角线的数组。
- **offset** (int, 可选) - 对角线与主对角线的偏移量。可以是正的或负的。默认主对角线。
- **axis1** (int, 可选) - 二维子数组的第一个轴。默认为第一个轴 (0)。
- **axis2** (int, 可选) - 二维子数组的第二个轴。默认为第二个轴。

返回：

Tensor，如果传递的数组 a 是二维数组，则返回包含对角线元素的一维数组。对于三维数组或更多维数组，将删除 `axis1` 和 `axis2` 指定的轴，并将两个参数中较小的一个添加到最后一个轴。

异常：

- **ValueError** - 如果输入的数据少于两个维度。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore.numpy as np
>>> a = np.arange(4).reshape(2,2).astype(np.float32)
>>> print(a)
[[0. 1.]
 [2. 3.]]
>>> output = np.diagonal(a)
>>> print(output)
[0. 3.]
>>> output = np.diagonal(a, 1)
>>> print(output)
[1.]
>>> a = np.arange(8).reshape(2, 2, 2).astype(np.float32)
>>> print(a)
[[[0. 1.]
 [2. 3.]]
 [[4. 5.]
 [6. 7.]]]
>>> output = np.diagonal(a, 0, 0, 1)
>>> print(output)
[[0. 6.]
 [1. 7.]]

```

15.1.12 mindspore.numpy.empty

`mindspore.numpy.empty(shape, dtype=mstype.float32)`

返回一个给定 `shape` 和类型的新数组，而不进行初始化。

说明：不支持原生 Numpy 接口的 `order` 参数，不支持对象数组。

参数：

- **shape** (`Union[int, tuple(int)]`) - 空数组的 `shape`，例如：(2, 3) 或 2。
- **dtype** (`mindspore.dtype`, 可选) - 数组所需输出的数据类型，例如 `mstype.int8`。默认值：`mstype.float32`。

返回：

Tensor，给定 `shape`、数据类型的未初始化（任意）数据的数组。

异常：

- **TypeError** - 如果输入的 `shape` 或数据类型无效。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore.numpy as np
>>> output = np.empty((2, 3))
>>> print(output)
[[0. 0. 0.]
 [0. 0. 0.]]

```

15.1.13 mindspore.numpy.empty_like

`mindspore.numpy.empty_like (prototype, dtype=None, shape=None)`

返回一个 `shape` 和类型与给定数组相同的新数组。

说明：输入数组在整个维度上必须具有相同的大小。如果 `prototype` 不是 `Tensor`，且未设置 `dtype`，则 `dtype` 默认为 `float32`。

参数：

- **prototype** (`Union[Tensor, list, tuple]`) - 原数组，其中生成数组的 `shape`、类型默认和原数组相同。
- **dtype** (`mindspore.dtype`, 可选) - 覆盖结果的数据类型。
- **shape** (`int`, `ints` 的序列, 可选) - 覆盖结果的 `shape`。

返回：

`Tensor`，给定 `shape` 和类型与 `prototype` 相同的未初始化（任意）数据的数组。

异常：

- **ValueError** - 如果 `prototype` 不是 `Tensor`、`List` 或 `Tuple`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((4,1,2))
>>> output = np.empty_like(a)
>>> print(output)
[[[0. 0.]]
 [[0. 0.]]
 [[0. 0.]]
 [[0. 0.]]]
```

15.1.14 mindspore.numpy.eye

`mindspore.numpy.eye (N, M=None, k=0, dtype=mstype.float32)`

返回一个对角线上值为 1，其他位置为 0 的二维 `Tensor`。

参数：

- **N** (`int`) - 输出二维 `Tensor` 的行数。输入值必须大于 0。
- **M** (`int`, 可选) - 输出二维 `Tensor` 的列数。默认值：None，如果输入为 None，则默认值：N，输入值必须大于 0。
- **k** (`int`, 可选) - 对角线的索引。默认值：0，即主对角线。正值表示上对角线，负值表示下对角线。
- **dtype** (`Union[mindspore.dtype, str]`, 可选) - 指定的 `Tensor dtype`。默认值：`mstype.float32`。

返回：

`Tensor`，`shape` 为 (N, M)。其中，除第 `k` 个对角线值等于 1 外，其余所有元素都等于 0。

异常：

- **TypeError** - 如果输入参数非给定的数据类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.eye(2, 2))
[[1. 0.]
 [0. 1.]]
```

15.1.15 mindspore.numpy.full

`mindspore.numpy.full(shape, fill_value, dtype=None)`

返回一个给定 `shape`、类型，并用 `fill_value` 填充的新数组。

参数：

- **shape** (`Union[int, tuple(int), list(int)]`) - 指定的 Tensor 的 `shape`，例如：(2, 3) 或 2。
- **fill_value** (`Union[int, float, bool, list, tuple]`) - 填充值，可以为标量或数组。
- **dtype** (`Union[mindspore.dtype, str]`, 可选) - 指定的 Tensor `dtype`。如果 `dtype` 为 `None`，则将从 `fill_value` 推断出新 Tensor 的数据类型。默认值：`None`。

返回：

Tensor，具有给定的 `shape`、类型，并用 `fill_value` 填充。

异常：

- **TypeError** - 如果输入参数非给定的数据类型。
- **ValueError** - 如果 `shape` 的元素数目小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.full((2,2), True))
[[True True]
 [True True]]
```

15.1.16 mindspore.numpy.full_like

`mindspore.numpy.full_like(a, fill_value, dtype=None, shape=None)`

返回一个与给定数组具有相同 `shape` 和类型的完整数组。

说明：

- 输入数组在一个维度上的大小必须相同。
- 如果 `a` 不是 Tensor，`dtype` 会默认为 `float32`。

参数：

- **a** (Union[Tensor, list, tuple]) - 原数组，返回数组的 **shape** 和数据类型与 **a** 相同。
- **fill_value** (scalar) - 填充值。
- **dtype** (mindspore.dtype, 可选) - 覆盖结果的数据类型。
- **shape** (int, ints 的序列, 可选) - 覆盖结果的 **shape**。

返回：

Tensor，与 **a** 的 **shape**、类型相同，并用 **fill_value** 填充。

异常：

- **ValueError** - 如果 **a** 的数据类型不是 Tensor、List 或 Tuple。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((4,1,2))
>>> output = np.full_like(a, 0.5)
>>> print(output)
[[[0.5 0.5]]
 [[0.5 0.5]]
 [[0.5 0.5]]
 [[0.5 0.5]]]
```

15.1.17 mindspore.numpy.geomspace

mindspore.numpy.geomspace (*start, stop, num=50, endpoint=True, dtype=None, axis=0*)

返回在对数刻度（几何级数）上均匀间隔的数字。这和 *mindspore.numpy.logspace()* 类似，但直接指定了端点。每个输出样本都是前一个样本的常数倍。

参数：

- **start** (Union[int, list(int), tuple(int), tensor]) - 序列的起始值。
- **stop** (Union[int, list(int), tuple(int), tensor]) - 当 **endpoint** 为 **True** 时，为序列的最终值；当 **endpoint** 为 **False** 时，在对数空间内的区间上均匀间隔 **num + 1** 个值，返回除最后一个值外（长度为 **num** 的序列）的其他值。
- **num** (int, 可选) - 生成的样本数。默认值 50。
- **endpoint** (bool, 可选) - 如果为 **True**，**stop** 是最后一个样本。否则，它不包括在内。默认值：True。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor dtype，可以是 np.float32 或 float32。如果 dtype 为 None，则将从其他输入参数推断出数据类型。默认值：None。
- **axis** (int, 可选) - 结果中用于存储样本的轴。仅当 **start** 或 **stop** 为类似数组对象时才用到。默认值：0，默认情况下的采样将沿着在开始处插入的新轴。使用 -1 在末尾获取一个轴。

返回：

Tensor，对数刻度上均匀间隔的样本。

异常：

- **TypeError** - 如果输入参数非给定的数据类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.geomspace(1, 256, num=9)
>>> print(output)
[ 1.  2.  4.  8. 16. 32. 64. 128. 256.]
>>> output = np.geomspace(1, 256, num=8, endpoint=False)
>>> print(output)
[ 1.  2.  4.  8. 16. 32. 64. 128.]
```

15.1.18 mindspore.numpy.hamming

`mindspore.numpy.hamming(M)`

返回一个 Hamming 窗口函数。Hamming 窗口函数是一种基于加权余弦的锥度函数。

参数：

- **M (int)** - 输出窗口的长度。如果为零或小于零，则返回一个空数组。

返回：

Tensor，生成一个长度为 M 且最大值归一化为 1 的 Hamming 窗口（仅当样本数为奇数时才会出现值 1）。

异常：

- **TypeError** - 如果 M 不是 int 型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.hamming(12))
[0.08000001 0.15302339 0.34890914 0.6054648  0.841236  0.9813669
 0.9813668 0.8412359 0.6054647 0.34890908 0.15302327 0.08000001]
```

15.1.19 mindspore.numpy.hanning

`mindspore.numpy.hanning(M)`

返回一个 Hanning 窗口函数。Hanning 窗口函数是一种基于加权余弦的锥度函数。

参数：

- **M (int)** - 输出窗口的长度。如果为零或小于零，则返回一个空数组。

返回：

Tensor，生成一个长度为 M 且最大值归一化为 1 的 Hanning 窗口（仅当样本数为奇数时才会出现值 1）。

异常：

- **TypeError** - 如果 M 不是 int 型。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.hanning(12))
[0.          0.07937324 0.29229254 0.5711574  0.8274304  0.9797465
 0.97974646 0.82743025 0.5711573  0.29229245 0.07937312 0.          ]
```

15.1.20 mindspore.numpy.histogram_bin_edges

`mindspore.numpy.histogram_bin_edges(a, bins=10, range=None, weights=None)`

计算 histogram 函数需要使用的 bins 的边界值。

说明: bins 不支持 string 类型。

参数:

- **a** (Union[int, float, bool, list, tuple, Tensor]) - 输入数据。直方图是在展平的数组上计算的。
- **bins** (Union[int, tuple, list, Tensor], 可选) - 指定分组的数量或具体边界。如果 bins 是一个 int 型数据, 则它给定了分组数 (默认为 10 个), 分组是在给定范围的等宽分组。如果 bins 是序列, 则它将定义每一个分组的边界 (包括最右边), 此时分组宽度可以是不等的。
- **range** ((float, float), 可选) - 指定边界的最小值和最大值。如果未提供, 则默认范围为 `(a.min(), a.max())`。超出范围的值将被忽略。范围的第一个元素必须小于或等于第二个。
- **weights** (Union[int, float, bool, list, tuple, Tensor], 可选) - 与 a shape 相同的权重数组。其中的每个权重值赋给相应的 a 中的值。目前暂未被任何分组估计算法使用, 未来可能被使用。默认值: None。

返回:

Tensor, 传递到 histogram 函数中的边界值。

异常:

- **TypeError** - 如果 bins 是一个数组且不是一维的。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> arr = np.array([0, 0, 0, 1, 2, 3, 3, 4, 5])
>>> print(np.histogram_bin_edges(arr, bins=2))
[0.  2.5 5. ]
```

15.1.21 mindspore.numpy.identity

`mindspore.numpy.identity(n, dtype=mstype.float32)`

返回单位数组。主对角线上全为 1。

参数：

- **n** (int) - 设置的输出数组的行数和列数，必须大于 0。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor dtype，默认值：mstype.float32。

返回：

Tensor，shape 为 (n, n)，除主对角线上值为 1，其余所有元素都等于 0。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.identity(2))
[[1. 0.]
 [0. 1.]]
```

15.1.22 mindspore.numpy.indices

`mindspore.numpy.indices(dimensions, dtype=mstype.int32, sparse=False)`

返回一个表示网格索引的数组。计算一个数组，其中子数组包含仅沿相应轴变化的索引值 0, 1, ...。

参数：

- **dimensions** (Union[list(int), tuple]) - 网格的 shape。
- **dtype** (mindspore.dtype, 可选) - 指定结果的数据类型。
- **sparse** (boolean, 可选) - 返回网格的稀疏表示而非密集表示，默认值：False。

返回：

Tensor，或元素为 Tensor 的 Tuple。如果 sparse 为 False，则返回一个网格索引数组：grid.shape = (len(dimensions),) + tuple(dimensions)。如果 sparse 为 True，则返回数组的 Tuple，其中 grid[i].shape = (1, ..., 1, dimensions[i], 1, ..., 1)，dimensions[i] 位于第 i 个位置。

异常：

- **TypeError** - 如果输入的 dimensions 不是 Tuple 或 List。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> grid = np.indices((2, 3))
>>> print(grid)
[Tensor(shape=[2, 3], dtype=Int32, value=
```

(下页继续)

(续上页)

```
[[0, 0, 0],
 [1, 1, 1]], Tensor(shape=[2, 3], dtype=Int32, value=
 [[0, 1, 2],
 [0, 1, 2]])]
```

15.1.23 mindspore.numpy.ix_

`mindspore.numpy.ix_(*args)`

从多个序列构建一个开放式网格，用于索引的坐标数组。它接受 N 个一维整数数组作为输入，并返回 N 个输出，每个输出具有 N 个维度，使得除了一个维度外，所有维度的 `shape` 都是 1，并且具有非单位形状值的维度在所有 N 个维度上循环。表示用于索引多维数组的坐标。使用 `ix_` 可以快速构建索引数组，对叉乘进行索引。`a[np.ix_([1,3],[2,5])]` 返回数组 `[[a[1,2] a[1,5]], [a[3,2] a[3,5]]]`。

说明：不支持布尔掩码。

参数：

- ***args** (Tensor) - 一维数组序列。

返回：

元素为 `Tensor` 的 `Tuple`。 N 个序列，每个序列有 N 个维度，其中 N 是输入序列的数量。这些序列共同构成一个开放式网格。

异常：

- **TypeError** - 如果输入的数据类型不是 `Tensor`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> ixgrid = np.ix_(np.array([0, 1]), np.array([2, 4]))
>>> print(ixgrid)
(Tensor(shape=[2, 1], dtype=Int32, value=
 [[0],
 [1]]), Tensor(shape=[1, 2], dtype=Int32, value=
 [[2, 4]]))
```

15.1.24 mindspore.numpy.linspace

`mindspore.numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)`

返回给定区间内均匀间隔的值。

参数：

- **start** (Union[int, list(int), tuple(int), tensor]) - 序列的起始值。
- **stop** (Union[int, list(int), tuple(int), tensor]) - 当 `endpoint` 为 `True` 时，为序列的最终值；当 `endpoint` 为 `False` 时，在给定区间上均匀间隔 `num + 1` 个值，返回除最后一个值外（长度为 `num` 的序列）的其他值。

- **num** (int, 可选) - 要生成的等间隔样例数量, 默认值: 50。
- **endpoint** (bool, 可选) - 序列中是否包含 stop 值, 默认值: True。
- **retstep** (bool, 可选) - 如果为 True, 则返回包含样例之间的 step (samples, step), 其中 step 是生成数值的间隔。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor dtype。如果 dtype 为 None, 则将从其他输入参数推断出新 Tensor 的数据类型。默认值: None。
- **axis** (int, 可选) - 结果中用于存储样本的轴。仅当 start 或 stop 为类似数组对象时才用到。默认值: 0, 默认情况下的采样将沿着在开始处插入的新轴。使用 -1 在末尾获取一个轴。

返回:

Tensor, 给定区间范围内以均匀间隔生成 num 个值。如果 endpoint 为 True, 数值范围为: $[start, stop]$; 反之则为: $[start, stop)$ 。Step, 样本之间间距的大小, 仅当 retstep 为 True 时返回。

异常:

- **TypeError** - 如果输入参数为非给定的数据类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.linspace(0, 5, 6))
[0. 1. 2. 3. 4. 5.]
```

15.1.25 mindspore.numpy.logspace

`mindspore.numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)`

返回在对数刻度上均匀间隔的值。在线性空间中, 序列从 $base ** start$ 开始, 以 $base ** stop$ 结束。

参数:

- **start** (Union[int, list(int), tuple(int), tensor]) - $base ** start$ 是序列的起始值。
- **stop** (Union[int, list(int), tuple(int), tensor]) - 当 endpoint 为 True 时, $base ** stop$ 是序列的最终值; 当 endpoint 为 False 时, 在对数空间内的区间上均匀间隔 $num + 1$ 个值, 返回除最后一个值外 (长度为 num 的序列) 的其他值。
- **num** (int, 可选) - 要生成的等间隔样例数量, 默认值: 50。
- **endpoint** (bool, 可选) - 序列中是否包含 stop 值, 默认值: True。
- **base** (Union[int, float], 可选) - 对数的底数。 $\ln(samples)/\ln(base)$ (或 $\log_{base}(samples)$) 元素之间的步长是均匀的, 默认值: 10。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor dtype。如果 dtype 为 None, 则将从其他输入参数推断出新 Tensor 的数据类型。默认值: None。
- **axis** (int, 可选) - 结果中用于存储样本的轴。仅当 start 或 stop 为类似数组对象时才用到。默认值: 0, 默认情况下的采样将沿着在开始处插入的新轴。使用 -1 在末尾获取一个轴。

返回:

Tensor, 对数刻度上均匀间隔的样本。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.logspace(0, 5, 6, base=2.0))
[ 1.  2.  4.  8. 16. 32.]
```

15.1.26 mindspore.numpy.meshgrid

`mindspore.numpy.meshgrid(*xi, sparse=False, indexing='xy')`

返回由坐标向量生成的坐标矩阵。根据一维坐标数组 $x_1, x_2, \dots, x_n$ ，生成用于对 N 维标量/向量场在 N 维网格上进行矢量化评估的 N 维坐标数组。

说明：不支持 Numpy 参数副本，并且始终只返回一个副本。

参数：

- ***xi** (Tensor) - 表示网格点坐标的一维数组。
- **indexing** ('xy', 'ij', 可选) - 输出数组的笛卡尔 ('xy', 默认值) 或矩阵 ('ij') 的索引。在输入长度为 M 和 N 的二维情况下，'xy' 索引的输出 shape 为 (N, M) ，'ij' 索引 shape 为 (M, N) 。在输入长度为 M 、 N 和 P 的三维情况下，'xy' 索引的输出 shape 为 (N, M, P) ，'ij' 索引的输入 shape 为 (M, N, P) 。
- **sparse** (bool, 可选) - 如果为 True 则返回稀疏矩阵以节省内存，默认值：False。

返回：

元素为 Tensor 的 Tuple，对于长度为 $N_i = \text{len}(x_i)$ 的向量 $x_1, x_2, \dots, x_n$ ，如果 `indexing='ij'`，则返回 shape 为 $(N_1, N_2, N_3, \dots, N_n)$ 的数组，或者如果 `indexing='xy'`，则返回 shape 为 $(N_2, N_1, N_3, \dots, N_n)$ 的数组，其中 x_i 的元素沿着第一维填充 x_1 矩阵，第二维度填充 x_2 矩阵，依此类推。

异常：

- **TypeError** - 如果输入不是 Tensor，或者 `sparse` 不是布尔值，或者 `indexing` 不是 'xy' 或 'ij'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.linspace(0, 1, 3)
>>> y = np.linspace(0, 1, 2)
>>> xv, yv = np.meshgrid(x, y)
>>> print(xv)
[[0.  0.5  1. ]
 [0.  0.5  1. ]]
>>> print(yv)
[[0.  0.  0.]
 [1.  1.  1.]]
>>> xv, yv = np.meshgrid(x, y, sparse=True)
```

(下页继续)

(续上页)

```
>>> print(xv)
[[0.  0.5  1.  ]]
>>> print(yv)
[[0.]
 [1.]]
```

15.1.27 mindspore.numpy.mgrid

`mindspore.numpy.mgrid()`

返回一个密集矩阵 NdGrid 实例，其中 `sparse=False`。输出数组的维度和数量等于索引维度的数量。如果 `step` 不是一个复数，则 `stop` 不包括在内。然而，如果 `step` 是复数（例如 `5j`），那么它的整数部分被解释为指定要在 `start` 和 `stop` 之间创建的点的数量，其中 `stop` 包括在内。

说明：在 `graph` 模式下不受支持。与 `Numpy` 不同，如果 `step` 是一个带有实数分量的复数，则 `step` 被处理为等效于 `int(abs(step))`。

返回：

元素为 `Tensor` 或 `Tensor` 的 `Tuple`，返回 `meshgrid` 网格点坐标数组。

异常：

- **TypeError** - 如果切片索引不是整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.mgrid[0:5, 0:5]
>>> print(output)
[[[0 0 0 0 0]
 [1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]]
 [[0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]
 [0 1 2 3 4]]]
>>> output = mgrid[-1:1:5j]
>>> print(output)
[-1.  -0.5  0.   0.5  1. ]
```

15.1.28 mindspore.numpy.ogrid

`mindspore.numpy.ogrid()`

返回一个稀疏矩阵 NdGrid 实例，其中 `sparse=True`。输出数组的维度和数量等于索引维度的数量。如果 `step` 不是一个复数，则 `stop` 不包括在内。然而，如果 `step` 是复数（例如 `5j`），那么它的整数部分被解释为指定要在 `start` 和 `stop` 之间创建的点的数量，其中 `stop` 包括在内。

说明：在 `graph` 模式下不受支持。与 `Numpy` 不同，如果 `step` 是一个带有实数分量的复数，则 `step` 被处理为等效于 `int(abs(step))`。

异常：

- **TypeError** - 如果切片索引不是整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.ogrid[0:5,0:5]
>>> print(output)
[Tensor(shape=[5, 1], dtype=Int32, value=
[[0],
 [1],
 [2],
 [3],
 [4]]), Tensor(shape=[1, 5], dtype=Int32, value=
[[0, 1, 2, 3, 4]])]
>>> output = ogrid[-1:1:5j]
>>> print(output)
[-1.  -0.5  0.   0.5  1. ]
```

15.1.29 mindspore.numpy.ones

`mindspore.numpy.ones(shape, dtype=mstype.float32)`

返回一个给定 `shape` 和类型的新 `Tensor`，其中所有元素用 1 来填充。

参数：

- **shape** (`Union[int, tuple, list]`) - 指定 `Tensor` 的 `shape`。
- **dtype** (`Union[mindspore.dtype, str]`, 可选) - 指定的 `Tensor dtype`。默认值：`mstype.float32`。

返回：

`Tensor`，给定 `shape` 和 `dtype`，其中所有元素都为 1。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 `shape` 的元素项值 < 0 。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.ones((2,2)))
[[1. 1.]
 [1. 1.]]
```

15.1.30 mindspore.numpy.ones_like

`mindspore.numpy.ones_like(a, dtype=None, shape=None)`

返回一个与给定数组 `a` 具有相同 `shape` 和类型的 `Tensor`，其中所有元素用 1 来填充。

说明：输入数组在同一维度上必须具有相同的 `size`。如果 `a` 不是 `Tensor`，且未设置 `dtype`，那么默认情况下 `dtype` 是 `float32`。

参数:

- **a** (`Union[Tensor, list, tuple]`) - 原数组，返回数组的 `shape` 和数据类型与 `a` 相同。
- **dtype** (`mindspore.dtype`, 可选) - 覆盖结果的数据类型。
- **shape** (`int`, `ints` 的序列, 可选) - 覆盖结果的 `shape`。

返回:

`Tensor`，与给定 `a` 的 `shape` 和数据类型相同，其中所有元素都为 1。

异常:

- **ValueError** - 如果 `a` 不是 `Tensor`、`List` 或 `Tuple`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.ones((4,1,2))
>>> output = np.ones_like(a)
>>> print(output)
[[[1. 1.]]
 [[1. 1.]]
 [[1. 1.]]
 [[1. 1.]]]
```

15.1.31 mindspore.numpy.pad

`mindspore.numpy.pad(arr, pad_width, mode='constant', stat_length=None, constant_values=0, end_values=0, reflect_type='even', **kwargs)`

对矩阵进行填充。

说明：目前，不支持 `median` 模式。`reflect` 和 `symmetric` 模式只支持 GPU 端。

参数：

- **arr** (Union[list, tuple, Tensor]) - 要填充的矩阵。
- **pad_width** (Union[int, tuple, list]) - 每个轴的边缘需要填充的值的数目。((before_1, after_1), ..., (before_N, after_N)) 为每个轴创建唯一的填充宽度。((before, after),) 为每个轴产生前后相同的 pad。(pad,) 或 int 是将所有轴设置为 before = after = pad width 的快捷设置。
- **mode** (string, 可选) - 填充的方式。
选择以下之一：
 - **constant** (默认值): 连续填充相同的值。
 - **edge**: 用 arr 的边缘值填充。
 - **linear_ramp**: 用边缘递减的方式填充。
 - **maximum**: 最大值填充。
 - **mean**: 均值填充。
 - **median**: 中位数填充。
 - **minimum**: 最小值填充。
 - **reflect**: 以边缘值为轴对称填充。
 - **symmetric**: 沿着边缘值对称填充。
 - **wrap**: 用原数组后面的值填充前面，前面的值填充后面。
 - **empty**: 用未定义值填充。
 - **<function>**: 用自定义函数填充。该自定义函数应修改并返回一个新的一维 Tensor。定义如下：
padding_func(tensor, iaxis_pad_width, iaxis, kwargs)。
- **stat_length** (Union[tuple, list, int], 可选) - 选择 'maximum', 'mean', 'median', 'minimum' 模式时，用于计算统计值的每个轴边缘的值的数量。可以为每个轴指定不同的统计长度，如 ((before_1, after_1), ..., (before_N, after_N))，也可以为所有轴设置唯一的统计长度。((before, after),) 为所有轴指定相等的前后统计长度，(stat_length,) 或 int 是将所有轴设置为 before = after = statistic length 的快捷设置。默认值：None。
- **constant_values** (Union[tuple, list, int], 可选) - 选择 'constant' 模式时，为每个轴设置填充值的常数值。((before_1, after_1), ..., (before_N, after_N)) 可以为每个轴定义唯一的填充常量。((before, after),) 为所有轴设置前后相同的填充常量。(constant,) 或 constant 是将所有轴设置为 before = after = constant 的快捷设置。默认值：0。
- **end_values** (Union[tuple, list, int], 可选) - 选择 'linear_ramp' 模式时，定义 linear_ramp 的结束值，这些值将构成填充 arr 的边缘。((before_1, after_1), ..., (before_N, after_N)) 可为每个轴设置唯一的结束值，((before, after),) 为所有轴设置前后相同的结束值。(constant,) 或 constant 是将所有轴设置为 before = after = constant 的快捷设置。默认值：0。

- **reflect_type** (string, 可选) - 选择'reflect', 'symmetric' 模式时, 默认为'even' 样式, 即围绕边缘值未改变的反射。对于'odd' 样式, 数组的扩展部分是通过从两倍的边缘值中减去反射值来创建的。
- **kwargs** (anytype, 可选) - 选择 <function> 模式时, 自定义函数使用的关键字参数。

返回:

填充后的 Tensor, 其秩与 arr 相同, shape 根据 pad_width 增加。

异常:

- **TypeError** - 如果输入的参数类型没有按上述内容指定。
- **ValueError** - 如果 mode 的输入无法被识别, 或如果 pad_width, stat_length, constant_values, end_values 无法被广播到 (arr.ndim, 2) 或关键字参数得到了意外的输入。
- **NotImplementedError** - 如果 mode 选择为'median' 或 function。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> tensor = np.array([1., 2., 3., 4., 5.])
>>> print(np.pad(tensor, (3, 4)))
[0. 0. 0. 1. 2. 3. 4. 5. 0. 0. 0. 0.]
>>> print(np.pad(tensor, (3, 4), mode="wrap"))
[3. 4. 5. 1. 2. 3. 4. 5. 1. 2. 3. 4.]
>>> print(np.pad(tensor, (3, 4), mode="linear_ramp", end_values=(10, 10)))
[10.    7.    4.    1.    2.    3.    4.    5.    6.25  7.5   8.75 10. ]
```

15.1.32 mindspore.numpy.rand

`mindspore.numpy.rand(*shape, dtype=mstype.float32)`

返回一个给定 shape 和类型的新 Tensor, 其中所有元素以区间 [0, 1) 上均匀分布的随机数来填充。

参数:

- **shape** (Union[int, tuple(int), list(int)]) - 指定新 Tensor 的 shape, 例如 (2, 3) 或 2。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定 Tensor 的数据类型。默认值: mstype.float32。

返回:

Tensor, 给定 shape 和类型, 其中所有元素都为区间 [0, 1) 上均匀分布的随机数。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 dtype 不是 float 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> from mindspore import set_seed
>>> set_seed(1)
>>> print(np.rand((2,3)))
[[4.1702199e-01 9.9718481e-01 7.2032452e-01]
 [9.3255734e-01 1.1438108e-04 1.2812445e-01]]
```

15.1.33 mindspore.numpy.randint

`mindspore.numpy.randint` (*minval*, *maxval=None*, *shape=None*, *dtype=mstype.int32*)

返回从 *minval*（包括）到 *maxval*（不包括）的随机整数。从“半开”区间范围 [*minval*, *maxval*) 内的指定数据类型的离散均匀分布中返回随机整数。如果 *maxval* 为 *None*（默认值），则取值范围为 [0, *minval*)，此时 *minval* 必须大于 0。

参数：

- **minval** (Union[int]) - 间隔的起始值（包括此值）。当 *maxval* 为 *None* 时，*minval* 必须大于 0。当 *maxval* 不为 *None* 时，*minval* 必须小于 *maxval*。
- **maxval** (Union[int], 可选) - 间隔的结束值（不包括此值）。
- **shape** (Union[int, tuple(int)]) - 指定的新 Tensor 的 *shape*，例如 (2,3) 或 2。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor 数据类型，必须是 int 型数据。默认值：mstype.int32。

返回：

Tensor，给定 *shape* 和类型，其中所有元素都为 *minval*（包括）到 *maxval*（不包括）之间的随机整数。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果输入参数的值不符合上述规定。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> from mindspore import set_seed
>>> set_seed(1)
>>> print(np.randint(1, 10, (2,3)))
[[4 9 7]
 [9 1 2]]
```

15.1.34 mindspore.numpy.randn

`mindspore.numpy.randn(*shape, dtype=mstype.float32)`

返回一个给定 `shape` 和类型的新 Tensor，并填充来自标准正态分布中的一个（或多个）样本。

参数：

- **shape** (Union[int, tuple(int), list(int)]) - 指定新 Tensor 的 shape，例如 (2, 3) 或 2。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的 Tensor 数据类型，必须是 float 型数据。默认值：mstype.float32。

返回：

Tensor，给定 shape 和类型，并填充来自“标准正态”分布中的一个（或多个）样本。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 dtype 不是 float 类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> from mindspore import set_seed
>>> set_seed(1)
>>> print(np.randn((2, 3)))
[[ 0.30639967 -0.42438635 -0.20454668]
 [-0.4287376   1.3054721   0.64747655]]
```

15.1.35 mindspore.numpy.trace

`mindspore.numpy.trace(a, offset=0, axis1=0, axis2=1, dtype=None)`

返回张量沿对角线的元素之和。如果 `a` 是二维数组，则返回沿给定偏移量的对角线的元素之和，即所有 `i` 对应的元素 `a[i, i+offset]` 的和。如果 `a` 有超过两个维度，则使用 `axis1` 和 `axis2` 指定的轴来确定返回的二维子数组的迹。结果数组的 `shape` 与移除 `axis1` 和 `axis2` 后的 `a` 相同。

说明：在 GPU 上，支持的数据类型为 `np.float16` 和 `np.float32`。在 CPU 上，支持的数据类型为 `np.float16`、`np.float32` 和 `np.float64`。

参数：

- **a** (Tensor) - 输入的需要计算的矩阵。
- **offset** (int, 可选) - 对角线相对于主对角线的偏移量。可以为正或负。默认是主对角线。
- **axis1** (int, 可选) - 用于二维子数组的第一个轴，从中提取对角线。默认是第一个轴 (0)。
- **axis2** (int, 可选) - 用于二维子数组的第二个轴，从中提取对角线。默认是第二个轴。
- **dtype** (mindspore.dtype, 可选) - 指定 Tensor 的数据类型，若不为 None 则重写 Tensor 的数据类型。默认值：None。

返回：

Tensor, 沿对角线的元素之和。如果传递的数组 *a* 是二维数组, 则返回主对角线元素的总和。如果 *a* 有更高的维度, 则返回沿对角线的和组成的数组。

异常:

- **ValueError** - 如果输入 Tensor 的维度小于二维。
- **ValueError** - 如果 *axis1* 或 *axis2* 不在 $[-\text{dims}, \text{dims})$ 范围内, 其中 *dims* 的维度为 *a*。
- **ValueError** - 如果 *axis1* 和 *axis2* 指定的轴相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.numpy import trace
>>> x = Tensor(np.eye(3, dtype=np.float32))
>>> print(trace(x))
3.0
```

15.1.36 mindspore.numpy.tri

`mindspore.numpy.tri(N, M=None, k=0, dtype=mstype.float32)`

返回一个 Tensor, 在给定的对角线处及以下元素值为 1, 在其他位置为 0。

参数:

- **N** (int) - 输入数组的行数。
- **M** (int, 可选) - 输入数组的列数。默认情况下 *M* 等于 *N*。
- **k** (int, 可选) - 对角线的偏移量: $k = 0$ 即为主对角线, $k < 0$ 即对角线向下偏移, $k > 0$ 即对角线向上偏移。默认值: 0。
- **dtype** (mindspore.dtype, 可选) - 指定的 Tensor 数据类型。默认值: `mstype.float32`。

返回:

Tensor, shape 为 (N, M) , 其中他的下三角区域填充为 1, 其余位置填充为 0; 用公式表达就是当 $j \leq i + k$ 时, $T[i, j] = 1$, 否则为 0。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.tri(3, 3, 1)
>>> print(output)
[[1. 1. 0.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

15.1.37 mindspore.numpy.tril

`mindspore.numpy.tril(m, k=0)`

返回张量的下三角部分。返回 `Tensor` 的副本，其中第 `k` 个对角线以上的元素为 0。

参数：

- **m** (`Union[Tensor, list, tuple]`) - 输入的 `Tensor`。返回的 `Tensor` 与 `m` 的 `shape`、数据类型都相同。
- **k** (`int`, 可选) - 对角线的偏移量：`k = 0` 即为主对角线，`k < 0` 即对角线向下偏移，`k > 0` 即对角线向上偏移。默认值：0。

返回：

`Tensor`，`m` 的下三角部分，`shape` 和数据类型与 `m` 相同。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果输入的 `m` 的秩 `< 1`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.tril(np.ones((3, 3)))
>>> print(output)
[[1. 0. 0.]
 [1. 1. 0.]
 [1. 1. 1.]]
```

15.1.38 mindspore.numpy.tril_indices

`mindspore.numpy.tril_indices(n, k=0, m=None)`

返回 `shape` 为 `(n, m)` 数组的下三角的索引。

参数：

- **n** (`int`) - 指定返回索引适用的数组大小。
- **k** (`int`, 可选) - 对角线的偏移量。默认值：0。
- **m** (`int`, 可选) - 指定返回索引适用的数组的列维度，默认情况下 `m` 等于 `n`。

返回：

下三角形的索引。返回的 `Tuple` 包含两个 `Tensor`，每个 `Tensor` 分别对应其在一个维度上的索引。

异常：

- **TypeError** - 如果 `n`，`k`，`m` 不是数字。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.tril_indices(3))
(Tensor(shape=[6], dtype=Int32, value= [0, 1, 1, 2, 2, 2]),
 Tensor(shape=[6], dtype=Int32, value= [0, 0, 1, 0, 1, 2]))
```

15.1.39 mindspore.numpy.tril_indices_from

`mindspore.numpy.tril_indices_from(arr, k=0)`

返回 `arr` 下三角的索引。

参数：

- **arr** (Union[`Tensor`, list, tuple]) - 二维数组。
- **k** (int, 可选) - 对角线的偏移量，默认值：0。

返回：

`tril_indices_from`，由两个 `Tensor` 组成的 `Tuple`，`arr` 的下三角形的索引。

异常：

- **TypeError** - 如果 `arr` 不能转换为一个 `Tensor`，或者 `k` 不是一个数字。
- **ValueError** - 如果 `arr` 不能转换为一个二维的 `Tensor`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> tensor = np.ones((3,3))
>>> print(np.tril_indices_from(tensor))
(Tensor(shape=[6], dtype=Int32, value= [0, 1, 1, 2, 2, 2]),
 Tensor(shape=[6], dtype=Int32, value= [0, 0, 1, 0, 1, 2]))
```

15.1.40 mindspore.numpy.triu

`mindspore.numpy.triu(m, k=0)`

返回张量的上三角部分。返回 `Tensor` 的副本，其中第 `k` 个对角线以下的元素为 0。

参数：

- **m** (Union[`Tensor`, list, tuple]) - 输入的 `Tensor`。返回的 `Tensor` 与 `m` 的 `shape`、数据类型都相同。
- **k** (int, 可选) - 对角线的偏移量： $k = 0$ 即为主对角线， $k < 0$ 即对角线向下偏移， $k > 0$ 即对角线向上偏移。默认值：0。

返回：

`Tensor`，`m` 的上三角部分，`shape` 和数据类型与 `m` 相同。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果输入的 `m` 的秩 < 1 。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.triu(np.ones((3, 3)))
>>> print(output)
[[1. 1. 1.]
 [0. 1. 1.]
 [0. 0. 1.]]
```

15.1.41 mindspore.numpy.triu_indices

`mindspore.numpy.triu_indices(n, k=0, m=None)`

返回 shape 为 (n, m) 数组的上三角的索引。

参数:

- **n** (int) - 指定返回索引适用的数组大小。
- **k** (int, 可选) - 对角线的偏移量。默认值: 0。
- **m** (int, 可选) - 指定返回索引适用的数组的列维度, 默认情况下 m 等于 n。

返回:

上三角形的索引。返回的 Tuple 包含两个 Tensor, 每个 Tensor 分别对应其在一个维度上的索引。

异常:

- **TypeError** - 如果 n, k, m 不是数字。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.triu_indices(3))
(Tensor(shape=[6], dtype=Int32, value= [0, 0, 0, 1, 1, 2]),
 Tensor(shape=[6], dtype=Int32, value= [0, 1, 2, 1, 2, 2]))
```

15.1.42 mindspore.numpy.triu_indices_from

`mindspore.numpy.triu_indices_from(arr, k=0)`

返回 arr 上三角的索引。

参数:

- **arr** (Union[Tensor, list, tuple]) - 二维数组。
- **k** (int, 可选) - 对角线的偏移量, 默认值: 0。

返回:

`triu_indices_from`, 由两个 Tensor 组成的 Tuple, arr 的上三角形的索引。

异常:

- **TypeError** - 如果 arr 不能转换为一个 Tensor, 或者 k 不是一个数字。

- **ValueError** - 如果 `arr` 不能转换为一个二维的 `Tensor`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> tensor = np.ones((3,3))
>>> print(np.triu_indices_from(tensor))
(Tensor(shape=[6], dtype=Int32, value= [0, 0, 0, 1, 1, 2]),
 Tensor(shape=[6], dtype=Int32, value= [0, 1, 2, 1, 2, 2]))
```

15.1.43 mindspore.numpy.vander

`mindspore.numpy.vander(x, N=None, increasing=False)`

生成一个范德蒙德矩阵。输出矩阵的列是输入向量的幂。幂的顺序由 `increasing boolean` 型参数决定。具体而言，当 `increasing` 为 `False` 时，第 `i` 列输出是按元素递增的输入向量，其幂为 $N - i - 1$ 。这样的每一个每行都有几何级数的矩阵被称为范德蒙德矩阵。

参数：

- **x** (`Union[list, tuple, Tensor]`) - 输入的一维数组。
- **N** (`int`, 可选) - 输出结果的列数。如果未指定 `N`，则返回一个 $N = \text{len}(x)$ 的方阵。
- **increasing** (`bool`, 可选) - 列的幂次顺序。如果为 `True`，则幂次从左到右递增，如果为 `False`，则幂次反向，默认值：`False`。

返回：

`Tensor`，范德蒙德矩阵，如果 `increasing` 为 `False`，则第一列为 $x^{(N-1)}$ ，第二列为 $x^{(N-2)}$ ，依此类推。如果 `increasing` 为 `True`，则列为 $x^0, x^1, \dots, x^{(N-1)}$ 。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果输入的 `x` 不是一维，或 `N` 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.vander([1., 2., 3., 4., 5.]))
[[ 1.  1.  1.  1.  1.]
 [16.  8.  4.  2.  1.]
 [81. 27.  9.  3.  1.]
 [256. 64. 16.  4.  1.]
 [625. 125. 25.  5.  1.]]
```

15.1.44 mindspore.numpy.zeros

`mindspore.numpy.zeros(shape, dtype=mstype.float32)`

返回一个给定 shape 和类型的新 Tensor，其中所有元素以 0 来填充。

参数：

- **shape** (Union[int, tuple, list]) - 指定的新 Tensor 的 shape。
- **dtype** (Union[mindspore.dtype, str], 可选) - 指定的新的 Tensor 数据类型。默认值：mstype.float32。

返回：

Tensor，给定 shape 和 dtype，其中所有元素都为 0。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 shape 的元素项值 < 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.zeros((2,2)))
[[0. 0.]
 [0. 0.]]
```

15.1.45 mindspore.numpy.zeros_like

`mindspore.numpy.zeros_like(a, dtype=None, shape=None)`

返回一个与给定数组 a 具有相同 shape 和类型的 Tensor，其中所有元素用 0 来填充。

说明：输入数组在同一维度上必须具有相同的 size。如果 a 不是 Tensor，且未设置 dtype，那么默认情况下 dtype 是 float32。

参数：

- **a** (Union[Tensor, list, tuple]) - 原数组，返回数组的 shape 和数据类型与 a 相同。
- **dtype** (mindspore.dtype, 可选) - 覆盖结果的数据类型。
- **shape** (int, ints 的序列, 可选) - 覆盖结果的 shape。

返回：

Tensor，与给定 a 的 shape 和数据类型相同，其中所有元素都为 0。

异常：

- **ValueError** - 如果 a 不是 Tensor、List 或 Tuple。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((4,1,2))
>>> output = np.zeros_like(a)
>>> print(output)
[[[0. 0.]]
 [[0. 0.]]
 [[0. 0.]]
 [[0. 0.]]]
```

15.2 Array 操作

操作类算子主要进行数组的维度变换，分割和拼接等。

- 数组维度变换

矩阵转置，代码示例：

```
input_x = np.arange(10).reshape(5, 2)
output = np.transpose(input_x)
print(output)
```

运行结果如下：

```
[[0 2 4 6 8]
 [1 3 5 7 9]]
```

交换指定轴，代码示例：

```
input_x = np.ones((1, 2, 3))
output = np.swapaxes(input_x, 0, 1)
print(output.shape)
```

运行结果如下：

```
(2, 1, 3)
```

- 数组分割

将输入数组平均切分为多个数组，代码示例：

```
input_x = np.arange(9)
output = np.split(input_x, 3)
print(output)
```

运行结果如下：

```
(Tensor(shape=[3], dtype=Int32, value= [0, 1, 2]), Tensor(shape=[3], dtype=Int32, value= [3, 4, 5]), Tensor(shape=[3], dtype=Int32, value= [6, 7, 8]))
```

- 数组拼接

将两个数组按照指定轴进行拼接，代码示例：

```
input_x = np.arange(0, 5)
input_y = np.arange(10, 15)
```

(下页继续)

(续上页)

```
output = np.concatenate((input_x, input_y), axis=0)
print(output)
```

运行结果如下:

```
[ 0  1  2  3  4 10 11 12 13 14]
```

接口名	概述	支持平台
<code>mindspore.numpy.append</code>	将值添加到 Tensor 的末尾。	Ascend GPU CPU
<code>mindspore.numpy.apply_along_axis</code>	在指定轴的一维切片上调用给定函数。	Ascend GPU CPU
<code>mindspore.numpy.apply_over_axes</code>	在多个轴上重复应用 <code>func</code> 函数。	Ascend GPU CPU
<code>mindspore.numpy.argwhere</code>	返回 Tensor 中非零元素的索引, 并按元素分组。	Ascend GPU CPU
<code>mindspore.numpy.array_split</code>	将一个 Tensor 切分为多个 Sub-Tensors 。	Ascend GPU CPU
<code>mindspore.numpy.array_str</code>	返回数组中数据的字符串表示形式。	Ascend GPU CPU
<code>mindspore.numpy.atleast_1d</code>	将输入转换为至少一维的数组。	Ascend GPU CPU
<code>mindspore.numpy.atleast_2d</code>	将输入重新调整为至少二维的数组。	Ascend GPU CPU
<code>mindspore.numpy.atleast_3d</code>	将输入重新调整为至少三维的数组。	Ascend GPU CPU
<code>mindspore.numpy.broadcast_arrays</code>	将任意数量的数组广播到共同的 shape 。	Ascend GPU CPU
<code>mindspore.numpy.broadcast_to</code>	将数组广播到新 shape 。	Ascend GPU CPU
<code>mindspore.numpy.choose</code>	从索引数组和要选择的数组列表构造一个新数组。	Ascend GPU CPU
<code>mindspore.numpy.column_stack</code>	将一维 Tensor 作为列堆叠为二维 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.concatenate</code>	沿现有轴连接一系列 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.dsplits</code>	沿第三轴 (深度) 将 Tensor 分割为多个 sub-tensor 。	Ascend GPU CPU
<code>mindspore.numpy.dstack</code>	按顺序在深度方向 (沿第三轴) 堆叠 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.expand_dims</code>	扩展 Tensor 的 shape 。	Ascend GPU CPU
<code>mindspore.numpy.flip</code>	沿给定轴反转数组中的元素顺序。	GPU CPU
<code>mindspore.numpy.fliplr</code>	在左右方向上翻转每行中的元素。	GPU CPU
<code>mindspore.numpy.flipud</code>	在上下方向上翻转每列中的元素。	GPU CPU
<code>mindspore.numpy.hsplit</code>	水平 (按列) 将 Tensor 分割为多个 sub-tensor 。	Ascend GPU CPU
<code>mindspore.numpy.hstack</code>	按顺序水平堆叠 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.intersect1d</code>	查找两个 Tensor 的交集。	Ascend GPU CPU
<code>mindspore.numpy.moveaxis</code>	将数组的轴移动到新位置。	Ascend GPU CPU
<code>mindspore.numpy.pieceswise</code>	执行分段定义的函数。	Ascend GPU CPU
<code>mindspore.numpy.ravel</code>	返回一个连续的展平 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.repeat</code>	重复数组的元素。	Ascend GPU CPU
<code>mindspore.numpy.reshape</code>	在不改变数据的情况下重塑一个 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.roll</code>	将 Tensor 沿给定的轴进行滚动。	Ascend GPU CPU
<code>mindspore.numpy.rollaxis</code>	将指定轴向后滚动, 直到它位于给定的位置。	Ascend GPU CPU
<code>mindspore.numpy.rot90</code>	将 Tensor 在指定的轴平面内旋转 90 度。	GPU
<code>mindspore.numpy.select</code>	根据条件从 <code>choicelist</code> 中的元素中返回数组。	Ascend GPU CPU
<code>mindspore.numpy.setdiff1d</code>	计算两个 Tensor 的差集。	Ascend GPU CPU
<code>mindspore.numpy.size</code>	返回沿给定轴的元素数量。	Ascend GPU CPU
<code>mindspore.numpy.split</code>	将一个 Tensor 沿指定轴分割为多个 sub-tensor 。	Ascend GPU CPU
<code>mindspore.numpy.squeeze</code>	返回删除指定 <code>axis</code> 中大小为 1 的维度后的 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.stack</code>	沿新轴连接一系列数组。	Ascend GPU CPU

下页继续

表 2 - 续上页

<code>mindspore.numpy.swapaxes</code>	交换 Tensor 的两个轴。	Ascend GPU CPU
<code>mindspore.numpy.take</code>	从数组中沿指定轴提取元素。	Ascend GPU CPU
<code>mindspore.numpy.take_along_axis</code>	根据一维索引和数据切片从输入数组中提取值。	Ascend GPU CPU
<code>mindspore.numpy.tile</code>	通过重复 <code>a</code> 指定次数构造一个数组, 次数由 <code>reps</code> 给出。	Ascend GPU CPU
<code>mindspore.numpy.transpose</code>	反转或交换 Tensor 的轴; 返回修改后的 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.unique</code>	返回去重后的 Tensor 元素。	Ascend GPU CPU
<code>mindspore.numpy.unravel_index</code>	将一维索引或一维索引数组转换为坐标数组的 tuple 。	Ascend GPU CPU
<code>mindspore.numpy.vsplit</code>	垂直 (按行) 将 Tensor 分割为多个 sub-tensor 。	Ascend GPU CPU
<code>mindspore.numpy.vstack</code>	按顺序垂直堆叠 Tensor 。	Ascend GPU CPU
<code>mindspore.numpy.where</code>	根据 <code>condition</code> 从 <code>x</code> 或 <code>y</code> 中选择元素。	Ascend GPU CPU

15.2.1 mindspore.numpy.append

`mindspore.numpy.append(arr, values, axis=None)`

将值添加到 **Tensor** 的末尾。

参数:

- **arr** (**Tensor**) - 需要被添加 `values` 的数组。
- **values** (**Tensor**) - 添加到数组 `arr` 中的值。该输入的形状必须正确 (与 `arr` 的形状相同, 不包括 `axis`)。如果未指定 `axis`, 则值可以是任何形状, 并且在使用前将被展平。
- **axis** (`int`, 可选) - 所要附加的 `values` 的轴。如果未给定 `axis`, 则 `arr` 和 `values` 在使用前都会被展平, 默认值: `None`。

返回:

Tensor, 添加值到指定 `axis` 上的 **Tensor**。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果输入的 `axis` 超过了 `arr.ndim`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.ones((2, 3))
>>> b = np.ones((2, 1))
>>> print(np.append(a, b, axis=1).shape)
(2, 4)
```

15.2.2 mindspore.numpy.apply_along_axis

`mindspore.numpy.apply_along_axis(func1d, axis, arr, *args, **kwargs)`

在指定轴的一维切片上调用给定函数。执行 `func1d(a, *args, **kwargs)`，其中 `func1d` 在一维数组上运算，`a` 是 `arr` 沿着指定 `axis` 的一维切片。

参数：

- **func1d** (function) - Maps (M,) -> (Nj...)。该函数仅接受输入为一维数组，应用于沿指定轴的 `arr` 的一维切片。
- **axis** (int) - 指定 `arr` 所需切片的轴。
- **arr** (Tensor) - 输入的数组，且输入包含数组 shape (Ni..., M, Nk...)。
- **args** (any) - `func1d` 的附加参数。
- **kwargs** (any) - `func1d` 的附加命名参数。

返回：

Tensor, shape 为 (Ni..., Nj..., Nk...)，除了 `axis` 那一维，它的 `shape` 与 `arr` 的 `shape` 相同。`axis` 那一维被替换为 `func1d` 的返回值 `shape`。因此，如果 `func1d` 返回标量，则输出的维度将比 `arr` 少一个。

异常：

- **ValueError** - 如果 `axis` 超出索引范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> b = np.array([[1,2,3], [4,5,6], [7,8,9]])
>>> print(np.apply_along_axis(np.diag, -1, b))
[[[1 0 0]
 [0 2 0]
 [0 0 3]]
 [[4 0 0]
 [0 5 0]
 [0 0 6]]
 [[7 0 0]
 [0 8 0]
 [0 0 9]]]
```

15.2.3 mindspore.numpy.apply_over_axes

`mindspore.numpy.apply_over_axes(func, a, axes)`

在多个轴上重复应用 `func` 函数。`res = func(a, axis)`，其中 `axis` 是 `axes` 的第一个元素。函数返回的结果 `res` 具有与 `a` 相同的维度或少一个维度。如果 `res` 比 `a` 少一个维度，则在 `axes` 之前插入一个维度。然后将 `res` 作为 `func` 的第一个参数，对 `axes` 中的每个轴重复调用 `func`。

参数：

- **func** (function) - 该函数必须有两个参数 `a`，`axis`。
- **a** (Union[int, float, bool, list, tuple, Tensor]) - 输入的 Tensor。
- **axes** (Union[int, list, tuple]) - 应用 `func` 函数的轴。输入元素必须是整数

返回：

Tensor，维度与 a 的相同，但 shape 可以不同。这取决于 func 函数是否改变输出的 shape。

异常：

- **TypeError** - 如果输入的 a 不是类似数组的对象，或者 axes 参数的类型不是 int 型或 int 型的序列。
- **ValueError** - 如果 axes 中的轴重复或超出索引范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.arange(10).reshape(2, 5).astype('float32')
>>> print(x)
[[0. 1. 2. 3. 4.]
 [5. 6. 7. 8. 9.]]
>>> print(np.apply_over_axes(np.sum, x, axes=0))
[[ 5.  7.  9. 11. 13.]]
```

15.2.4 mindspore.numpy.argmaxwhere

`mindspore.numpy.argmaxwhere(a)`

返回 Tensor 中非零元素的索引，并按元素分组。

参数：

- **a** (Union[list, tuple, Tensor]) - 输入的 Tensor。

返回：

Tensor，非零元素的索引。索引按元素分组。Tensor 的 shape 为 $(N, a.ndim)$ 其中 N 为非零元素个数。

异常：

- **TypeError** - 如果输入的 a 不是类似数组的对象。
- **ValueError** - 如果 a 的维数为 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.array([[1, 0], [-5, 0]])
>>> np.argmaxwhere(x)
Tensor(shape=[2, 3], dtype=Int64, value=[[0, 0, 0], [0, 1, 0]])
```


15.2.5 mindspore.numpy.array_split

`mindspore.numpy.array_split(x, indices_or_sections, axis=0)`

将一个 Tensor 切分为多个 Sub-Tensors。

说明：目前，`array_split` 仅支持 CPU 上的 `mindspore.float32`。

`np.split` 和 `np.array_split` 之间的唯一区别是，`np.array_split` 允许 `indices_or_sections` 是一个不用等分 `axis` 的整数。对于长度为 `l` 的 Tensor，其应当被分割成 `n` 个部分，返回的子数组中一部分子数组 `shape` 为 `l//n+1`，剩余子数组 `shape` 为 `l//n`。

参数：

- **x** (Tensor) - 待切分的 Tensor。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 如果是整数 `N`，Tensor 将沿指定的轴被拆分成 `N` 个子张量。如果是 `tuple(int)`、`list(int)` 或排序后的整数序列，则这些条目表示在指定轴上的拆分位置。例如，给定 `[2, 3]` 在 `axis = 0` 上拆分，则结果将是三个 Sub-Tensors，为 `x[:2]`，`x[2:3]`，`x[3:]`。如果某个索引超出了指定轴的维度，相应地将返回一个空子数组。
- **axis** (int) - 要沿其切分的轴。默认值：0。

返回：

Sub-Tensors 列表。

异常：

- **TypeError** - 如果参数 `indices_or_sections` 不是整数、`tuple(int)` 或 `list(int)`，或者参数 `axis` 不是整数。
- **ValueError** - 如果参数 `axis` 超出 `[-x.ndim, x.ndim)` 的范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input_x = np.arange(9).astype("float32")
>>> output = np.array_split(input_x, 4)
>>> print(output)
(Tensor(shape=[3], dtype=Float32,
  value= [ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00]),
Tensor(shape=[2], dtype=Float32,
  value= [ 3.00000000e+00,  4.00000000e+00]),
Tensor(shape=[2], dtype=Float32,
  value= [ 5.00000000e+00,  6.00000000e+00]),
Tensor(shape=[2], dtype=Float32,
  value= [ 7.00000000e+00,  8.00000000e+00]))
```

15.2.6 mindspore.numpy.array_str

`mindspore.numpy.array_str(a)`

返回数组中数据的字符串表示形式。数组中的数据以单个字符串的形式返回。此函数类似于 `array_repr`，不同之处在于 `array_repr` 还返回数组的类型及其数据类型的信息。

说明：不支持 Numpy 的 `max_line_width`、`precision` 和 `suppress_small` 参数。在图模式下不支持该函数。

参数：

- **a** (Tensor) - 输入数据。

返回：

字符串。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.arange(5)
>>> np.array_str(x)
'[0 1 2 3 4]'
```

15.2.7 mindspore.numpy.atleast_1d

`mindspore.numpy.atleast_1d(*arys)`

将输入转换为至少一维的数组。标量输入将被转换为一维数组，而高维输入则保持不变。

说明：在图模式下，返回的不是 Tensor 列表，而是 Tensor 的 tuple。

参数：

- ***arys** (Tensor) - 一个或多个输入 Tensor。

返回：

Tensor 或 Tensor 列表，每个 Tensor 满足 `a.ndim >= 1`。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((2, 3))
>>> b = np.ones(())
>>> c = np.ones(5)
>>> output = np.atleast_1d(a, b, c)
>>> print(output)
[Tensor(shape=[2, 3], dtype=Float32, value=
[[1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
 [1.00000000e+00, 1.00000000e+00, 1.00000000e+00]]),
 Tensor(shape=[1], dtype=Float32, value= [1.00000000e+00]),
 Tensor(shape=[5], dtype=Float32,
value= [1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
1.00000000e+00, 1.00000000e+00])]
```

15.2.8 mindspore.numpy.atleast_2d

`mindspore.numpy.atleast_2d(*args)`

将输入重新调整为至少二维的数组。

说明：在图模式下，返回的不是 Tensor 列表，而是 Tensor 的 tuple。

参数：

- ***args** (Tensor) - 一个或多个输入 Tensor。

返回：

Tensor 或 Tensor 列表，每个 Tensor 满足 $a.ndim \geq 2$ 。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((2, 3))
>>> b = np.ones(())
>>> c = np.ones(5)
>>> output = np.atleast_2d(a, b, c)
>>> print(output)
[Tensor(shape=[2, 3], dtype=Float32, value=
[[1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
 [1.00000000e+00, 1.00000000e+00, 1.00000000e+00]]),
 Tensor(shape=[1, 1], dtype=Float32, value= [[1.00000000e+00]]),
 Tensor(shape=[1, 5], dtype=Float32,
value= [[1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
1.00000000e+00, 1.00000000e+00]])]
```

15.2.9 mindspore.numpy.atleast_3d

`mindspore.numpy.atleast_3d(*arys)`

将输入重新调整为至少三维的数组。

说明：在图模式下，返回的不是 Tensor 列表，而是 Tensor 的 tuple。

参数：

- ***arys** (Tensor) - 一个或多个输入 Tensor。

返回：

Tensor 或 Tensor 列表，每个 Tensor 满足 $a.ndim \geq 3$ 。例如，一个 shape 为 (N,) 的一维数组会变为 shape 为 (1, N, 1) 的 Tensor，一个 shape 为 (M, N) 的二维数组会变为 shape 为 (M, N, 1) 的 Tensor。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((2, 3))
>>> b = np.ones(())
>>> c = np.ones(5)
>>> output = np.atleast_3d(a, b, c)
>>> print(output)
[Tensor(shape=[2, 3, 1], dtype=Float32, value=
[[[1.00000000e+00], [1.00000000e+00], [1.00000000e+00]],
 [[1.00000000e+00], [1.00000000e+00], [1.00000000e+00]]]),
 Tensor(shape=[1, 1, 1], dtype=Float32, value= [[1.00000000e+00]]),
 Tensor(shape=[1, 5, 1], dtype=Float32,
 value= [[1.00000000e+00], [1.00000000e+00], [1.00000000e+00],
 [1.00000000e+00], [1.00000000e+00]])]
```

15.2.10 mindspore.numpy.broadcast_arrays

`mindspore.numpy.broadcast_arrays(*args)`

将任意数量的数组广播到共同的 shape。

说明：不支持 Numpy 的 `subok` 参数。在图模式下，返回的不是 Tensor 列表，而是 Tensor 的 tuple。

参数：

- ***args** (Tensor) - 要进行广播的数组。

返回：

Tensor 列表。

异常：

- **ValueError** - 如果数组不能被广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.array([[1, 2, 3]])
>>> y = np.array([[4], [5]])
>>> output = np.broadcast_arrays(x, y)
>>> print(output)
[Tensor(shape=[2, 3], dtype=Int32, value=
[[1, 2, 3],
[1, 2, 3]]), Tensor(shape=[2, 3], dtype=Int32, value=
[[4, 4, 4],
[5, 5, 5]])]
```

15.2.11 mindspore.numpy.broadcast_to

`mindspore.numpy.broadcast_to(array, shape)`

将数组广播到新 shape。

参数：

- **array** (Tensor) - 要广播的数组。
- **shape** (tuple) - 目标数组的 shape。

返回：

Tensor，原始数组被广播到指定的 shape。

异常：

- **ValueError** - 如果数组不能被广播到指定 shape。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.array([1, 2, 3])
>>> output = np.broadcast_to(x, (3, 3))
>>> print(output)
[[1 2 3]
 [1 2 3]
 [1 2 3]]
```

15.2.12 mindspore.numpy.choose

`mindspore.numpy.choose(a, choices, mode='clip')`

从索引数组和要选择的数组列表构造一个新数组。给定一个包含整数的“索引”数组 `a` 和一个由 `n` 个数组 (`choices`) 组成的序列, `a` 和每个选择数组首先会根据需要广播为相同 `shape` 的数组; 我们称这些数组为 `Ba` 和 `Bchoices[i]`, $i=0, \dots, n-1$, 并且对于每个 i , 必然有 `Ba.shape == Bchoices[i].shape`。然后, 按照以下方式创建一个新数组, `shape` 为 `Ba.shape` :

- 如果 `mode='raise'` (默认值), 那么首先, `a` (因此 `Ba`) 的每个元素必须在 $[0, n-1]$ 范围内; 假设 i (在该范围内) 是 `Ba` 中 $(j_0, j_1, \dots, j_m)$ 位置的索引——那么新数组中相同位置的索引就是 `Bchoices[i]` 在该位置的索引;
- 如果 `mode='wrap'`, 则 `a` (因此 `Ba`) 中的索引可以是任何 (带符号) 整数; 使用模运算将超出 $[0, n-1]$ 范围的索引映射回该范围内; 然后像上述一样构造新数组;
- 如果 `mode='clip'`, 则 `a` (因此 `Ba`) 中的索引可以是任何 (带符号) 整数; 负索引被映射为 0; 大于 $n-1$ 的索引被映射为 $n-1$; 然后像上述一样构造新数组。

说明: 不支持 Numpy 的 `out` 参数。不支持 `mode = 'raise'`, 默认 `mode` 为 `'clip'`。

参数:

- **a** (int array) - 该数组必须包含 $[0, n-1]$ 范围内的索引, 其中 n 是选择数组的数量, 除非使用 `mode=wrap` 或 `mode=clip`, 在这些情况下, 任何索引都可以。
- **choices** (数组的序列) - 选择数组。 `a` 和所有 `choices` 必须可以广播为相同的 `shape`。如果 `choices` 本身是一个数组, 则其最外层维度 (即 `choices.shape[0]` 对应的维度) 定义为 “sequence”。
- **mode** ('raise', 'wrap', 'clip', 可选) - 指定如何处理超出 $[0, n-1]$ 范围的索引:
 - 'raise' - 抛出错误;
 - 'wrap' - 循环;
 - 'clip' - 限制在范围内。'clip' 模式意味着所有过大的索引都将被替换为该轴最后一个元素的索引。请注意, 这将禁止使用负数进行索引。

返回:

Tensor, 合并的结果。

异常:

- **ValueError** - 如果 `a` 与任一 `choices` 不能被广播。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> choices = [[0, 1, 2, 3], [10, 11, 12, 13], [20, 21, 22, 23], [30, 31, 32, 33]]
>>> print(np.choose([2, 3, 1, 0], choices))
[20 31 12  3]
>>> print(np.choose([2, 4, 1, 0], choices, mode='clip'))
[20 31 12  3]
>>> print(np.choose([2, 4, 1, 0], choices, mode='wrap'))
[20  1 12  3]
>>> a = [[1, 0, 1], [0, 1, 0], [1, 0, 1]]
```

(下页继续)

(续上页)

```
>>> choices = [-10, 10]
>>> print(np.choose(a, choices))
[[ 10 -10  10]
 [-10  10 -10]
 [ 10 -10  10]]
```

15.2.13 mindspore.numpy.column_stack

`mindspore.numpy.column_stack(tup)`

将一维 Tensor 作为列堆叠为二维 Tensor。二维 Tensor 保持原样堆叠，如同 `np.hstack`。

参数：

- **tup** (Union[Tensor, tuple, list]) - 一组一维或二维 Tensor。它们必须具有相同的 shape，除了要连接的轴外。

返回：

二维 Tensor，由给定的 Tensor 堆叠而成。

异常：

- **TypeError** - 如果 `tup` 不是 Tensor、list 或 tuple。
- **ValueError** - 如果 `tup` 为空。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.array([1, 2, 3]).astype('int32')
>>> x2 = np.array([4, 5, 6]).astype('int32')
>>> output = np.column_stack((x1, x2))
>>> print(output)
[[1 4]
 [2 5]
 [3 6]]
```

15.2.14 mindspore.numpy.concatenate

`mindspore.numpy.concatenate(arrays, axis=0)`

沿现有轴连接一系列 Tensor。

说明：为了匹配 Numpy 的行为，当 `axis >= 32` 时不会引发值错误，`axis` 将被视为 `None`。

参数：

- **arrays** (Union[Tensor, tuple(Tensor), list(Tensor)]) - 要连接的一个 Tensor 或 Tensor 列表。
- **axis** (Union[None, int], 可选) - 沿该轴连接 Tensor。如果 `axis` 为 `None`，则在使用前会先将 Tensor 展平。默认值：0。

返回：

从一个 Tensor 或 Tensor 列表中连接后的 Tensor。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 `axis` 不在范围 `[-ndim, ndim - 1]` 内，且小于 32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.ones((1, 2, 3))
>>> x2 = np.ones((1, 2, 1))
>>> x = np.concatenate((x1, x2), axis=-1)
>>> print(x.shape)
(1, 2, 4)
```

15.2.15 mindspore.numpy.dsplitt

`mindspore.numpy.dsplitt(x, indices_or_sections)`

沿第三轴（深度）将 Tensor 分割为多个 sub-tensor。此操作等同于使用 `axis = 2`（默认值）进行分割，不论数组的维度如何，始终沿第三轴分割。

参数：

- **x** (Tensor) - 待分割的 Tensor。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 如果是整数 N ，Tensor 将沿轴分割为 N 个相等的 sub-tensor。如果是 tuple(int)、list(int) 或排序后的整数，则指示沿轴的分割位置。例如，对于 `axis = 0`，`[2, 3]` 将产生三个 sub-tensor: `x[: 2]`、`x[2 : 3]` 和 `x[3 :]`。如果索引超出轴上数组的维度，则相应地返回空子数组。

返回：

sub-tensor 列表。

异常：

- **TypeError** - 如果参数 `indices_or_sections` 不是整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input_x = np.arange(6).reshape((1, 2, 3)).astype('float32')
>>> output = np.dsplitt(input_x, 3)
>>> print(output)
(Tensor(shape=[1, 2, 1], dtype=Float32,
value=[[[ 0.00000000e+00],
[ 3.00000000e+00]]]),
Tensor(shape=[1, 2, 1], dtype=Float32,
value=[[[ 1.00000000e+00],
[ 4.00000000e+00]]]),
Tensor(shape=[1, 2, 1], dtype=Float32,
```

(下页继续)

(续上页)

```
value=[[ [ 2.00000000e+00],
         [ 5.00000000e+00]]])
```

15.2.16 mindspore.numpy.dstack

`mindspore.numpy.dstack(tup)`

按顺序在深度方向（沿第三轴）堆叠 Tensor。此操作等同于沿第三轴进行连接。对于一维 Tensor (N)，应将其 shape 变为 $(1, N, 1)$ 。对于二维 Tensor (M, N)，应将其 shape 变为 $(M, N, 1)$ 后再进行连接。

参数：

- **tup** (Union[Tensor, tuple, list]) - 一组 Tensor。Tensor 必须在除第三轴之外的所有轴上具有相同的 shape。一维或二维 Tensor 必须具有相同的 shape。

返回：

堆叠后的 Tensor，由给定的 Tensor 堆叠而成。

异常：

- **TypeError** - 如果 tup 不是 Tensor、list 或 tuple。
- **ValueError** - 如果 tup 为空。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.array([1, 2, 3]).astype('float32')
>>> x2 = np.array([4, 5, 6]).astype('float32')
>>> output = np.dstack((x1, x2))
>>> print(output)
[[[1. 4.]
  [2. 5.]
  [3. 6.]]]
```

15.2.17 mindspore.numpy.expand_dims

`mindspore.numpy.expand_dims(a, axis)`

扩展 Tensor 的 shape。在扩展后的 Tensor 的 shape 中插入一个新轴，该轴将出现在指定的位置。

参数：

- **a** (Tensor) - 输入 Tensor 数组。
- **axis** (Union[int, list(int), tuple(int)]) - 新轴在扩展轴中的位置。

返回：

Tensor，指定轴位置的维度数增加后的 Tensor。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 axis 超出了 `a.ndim`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.ones((2,2))
>>> x = np.expand_dims(x,0)
>>> print(x.shape)
(1, 2, 2)
```

15.2.18 mindspore.numpy.flip

`mindspore.numpy.flip(m, axis=None)`

沿给定轴反转数组中的元素顺序。数组的 `shape` 保持不变，但元素的顺序会被重新排列。

参数：

- **m** (Tensor) - 输入数组。
- **axis** (Union[int, tuple(int), None], 可选) - 反转的轴或轴的 **tuple**。默认值：`axis=None`，表示对输入数组的所有轴进行反转。如果 `axis` 为负数，则从最后一个轴开始计数。如果 `axis` 是整数 **tuple**，则对 **tuple** 中指定的所有轴进行反转。

返回：

Tensor，沿指定 `axis` 的元素被反转。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> A = np.arange(8.0).reshape((2,2,2))
>>> output = np.flip(A)
>>> print(output)
[[[7. 6.]
  [5. 4.]]
 [[3. 2.]
  [1. 0.]]]
>>> output = np.flip(A, (0, 2))
>>> print(output)
[[[5. 4.]
  [7. 6.]]
 [[1. 0.]
  [3. 2.]]]
```

15.2.19 mindspore.numpy.fliplr

`mindspore.numpy.fliplr(m)`

在左右方向上翻转每行中的元素。单个列保持不变，但列之间的顺序会发生改变。

参数：

- **m** (Tensor) - 输入数组。

返回：

Tensor。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> A = np.arange(8.0).reshape((2,2,2))
>>> output = np.fliplr(A)
>>> print(output)
[[[2. 3.]
  [0. 1.]]
 [[6. 7.]
  [4. 5.]]]
```

15.2.20 mindspore.numpy.flipud

`mindspore.numpy.flipud(m)`

在上下方向上翻转每列中的元素。单个行保持不变，但行之间的顺序会发生改变。

参数：

- **m** (Tensor) - 输入数组。

返回：

Tensor。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> A = np.arange(8.0).reshape((2,2,2))
>>> output = np.flipud(A)
>>> print(output)
[[[4. 5.]
  [6. 7.]]]
```

(下页继续)

(续上页)

```
[[0. 1.]
 [2. 3.]]
```

15.2.21 mindspore.numpy.hsplitt

`mindspore.numpy.hsplitt(x, indices_or_sections)`

水平（按列）将 Tensor 分割为多个 sub-tensor。此操作等同于使用 `axis = 1`（默认值）进行分割，不论数组的维度如何，始终沿第二轴分割。

参数：

- **x** (Tensor) - 待分割的 Tensor。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 如果是整数 N ，Tensor 将沿轴分割为 N 个相等的 sub-tensor。如果是 tuple(int)、list(int) 或排序后的整数，则指示沿轴的分割位置。例如，对于 `axis = 0`，`[2, 3]` 将产生三个 sub-tensor: `x[:2]`、`x[2:3]` 和 `x[3:]`。如果索引超出轴上数组的维度，则相应地返回空子数组。

返回：

sub-tensor 列表。

异常：

- **TypeError** - 如果参数 `indices_or_sections` 不是整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input_x = np.arange(6).reshape((2, 3)).astype('float32')
>>> output = np.hsplitt(input_x, 3)
>>> print(output)
(Tensor(shape=[2, 1], dtype=Float32,
value=[[ 0.00000000e+00],
       [ 3.00000000e+00]]),
 Tensor(shape=[2, 1], dtype=Float32,
value=[[ 1.00000000e+00],
       [ 4.00000000e+00]]),
 Tensor(shape=[2, 1], dtype=Float32,
value=[[ 2.00000000e+00],
       [ 5.00000000e+00]]))
```

15.2.22 mindspore.numpy.hstack

`mindspore.numpy.hstack(tup)`

按顺序水平堆叠 Tensor。此操作等效于沿第二轴连接 Tensor，但当输入为一维 Tensor 时，则沿第一轴连接。

参数：

- **tup** (Union[Tensor, tuple, list]) - 一组一维或二维 Tensor。Tensor 在除第二轴外的所有轴上必须具有相同的 shape，一维 Tensor 的长度可以不同。

返回：

堆叠后的 Tensor，由给定的 Tensor 堆叠而成。

异常：

- **TypeError** - 如果 `tup` 不是 `Tensor`、`list` 或 `tuple`。
- **ValueError** - 如果 `tup` 为空。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.array([1, 2, 3]).astype('float32')
>>> x2 = np.array([4, 5, 6]).astype('float32')
>>> output = np.hstack((x1, x2))
>>> print(output)
[1. 2. 3. 4. 5. 6.]
```

15.2.23 mindspore.numpy.intersect1d

`mindspore.numpy.intersect1d(ar1, ar2, assume_unique=False, return_indices=False)`

查找两个 `Tensor` 的交集。返回两个输入 `Tensor` 中都存在的、已排序去重的值。

参数：

- **ar1** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入 `Tensor`。
- **ar2** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入 `Tensor`。
- **assume_unique** (`bool`) - 如果为 `True`，则假设输入 `Tensor` 没有重复的元素，这可以加快计算速度。若为 `True` 但 `ar1` 或 `ar2` 不唯一，可能会导致结果不正确或索引超出范围。默认值：`False`。
- **return_indices** (`bool`) - 如果为 `True`，返回与交集对应的索引。如果值出现多次，则使用第一次出现的索引。默认值：`False`。

返回：

`Tensor` 或 `Tensor` 的 `tuple`。若 `return_indices` 为 `False`，则返回交集 `Tensor`；否则返回 `Tensor` 的 `tuple`。

异常：

- **TypeError** - 如果输入的 `ar1` 或 `ar2` 不是类似数组的对象。
- **TypeError** - 如果 `assume_unique` 或 `return_indices` 不是 `bool` 类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> np.intersect1d([1, 3, 4, 3], [3, 1, 2, 1])
Tensor(shape=[2], dtype=Int32, value=[1, 3])
```

15.2.24 mindspore.numpy.moveaxis

`mindspore.numpy.moveaxis(a, source, destination)`

将数组的轴移动到新位置。其他轴保持原始顺序不变。

参数：

- **a** (Tensor) - 原数组，返回数组的 `shape` 和数据类型与 `a` 相同。待修改
- **source** (int, ints 的序列) - 要移动的轴的原始位置。这些位置必须唯一。
- **destination** (int, ints 的序列) - 每个原始轴的新位置。这些位置也必须唯一。

返回：

Tensor, 已经移动过轴的数组。

异常：

- **ValueError** - 如果轴超出范围 `[-a.ndim, a.ndim)`，或者轴中包含重复项。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.zeros((3, 4, 5))
>>> output = np.moveaxis(x, 0, -1)
>>> print(output.shape)
(4, 5, 3)
>>> output = np.moveaxis(x, -1, 0)
>>> print(output.shape)
(5, 3, 4)
>>> output = np.moveaxis(x, [0, 1, 2], [-1, -2, -3])
>>> print(output.shape)
(5, 4, 3)
```

15.2.25 mindspore.numpy.pieceswise

`mindspore.numpy.pieceswise(x, condlist, funclist, *args, **kw)`

执行分段定义的函数。根据一组条件和相应的函数，在条件为真的位置对输入数据进行函数求值。

参数：

- **x** (Union[int, float, bool, list, tuple, Tensor]) - 输入定义域。
- **condlist** (Union[bool, list[Tensor, bool]]) - 每个 `boolean` 数组与 `funclist` 中的一个函数相对应。在 `condlist[i]` 为真时，使用 `funclist[i](x)` 作为输出值。`condlist` 中的每个 `boolean` 数组选择 `x` 的一个部分，因此应该与 `x` 的 `shape` 相同。`condlist` 的长度必须与 `funclist` 的长度相对应。如果给出了一个额外的函数，即 `len(funclist) == len(condlist) + 1`，则该额外的函数为默认值，用于所有条件均为假的地方。
- **funclist** (Union[list[callables], list[scalars]]) - 每个函数在其相应的条件为真时对 `x` 进行计算。它应该接受一个一维数组作为输入，并输出一个一维数组或标量值。如果提供的不是可调用对象，而是标量，则假定为常数函数 (`lambda x: scalar`)。
- **args** (any) - 传递给 `pieceswise` 的任何进一步参数都会在执行时传递给函数，例如，如果调用 `pieceswise(..., ..., 1, 'a')`，则每个函数都将被调用为 `f(x, 1, 'a')`。

- **kw (any)** - 用于调用 `piecewise` 的关键字参数, 在执行时传递给函数, 例如, 如果调用 `piecewise(..., ..., alpha=1)`, 则每个函数将被调用为 `f(x, alpha=1)`。

返回:

Tensor, 输出与 `x` 具有相同的 `shape` 和类型, 通过对 `x` 的对应部分调用 `funclist` 中的函数找到输出值, 这些部分由 `condlist` 中的 **boolean** 数组定义。未被任何条件覆盖的部分默认值为 0。

异常:

- **ValueError** - 如果 `funclist` 的长度不在 $(len(condlist), len(condlist) + 1)$ 范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.linspace(-2.5, 2.5, 6)
>>> print(np.piecewise(x, [x < 0, x >= 0], [-1, 1]))
[-1 -1 -1  1  1  1]
```

15.2.26 mindspore.numpy.ravel

`mindspore.numpy.ravel(x)`

返回一个连续的展平 **Tensor**。返回一个一维 **Tensor**, 包含输入 **Tensor** 的所有元素。

参数:

- **x (Tensor)** - 需要展平的 **Tensor**。

返回:

展平后的 **Tensor**, 与原始 **Tensor** `x` 具有相同的数据类型。

异常:

- **TypeError** - 如果 `x` 不是 **Tensor**。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.ones((2,3,4))
>>> output = np.ravel(x)
>>> print(output.shape)
(24,)
```

15.2.27 mindspore.numpy.repeat

`mindspore.numpy.repeat(a, repeats, axis=None)`
重复数组的元素。

参数：

- **a** (Tensor) - 输入数组。
- **repeats** (int, ints 的序列) - 每个元素的重复次数。repeats 将广播到适应指定轴的 shape。
- **axis** (int, 可选) - 沿着该轴重复元素。如果未指定轴，输入数组将被展平，并返回一个一维的输出数组。默认值：None。

返回：

Tensor，输出数组，除了指定的轴外，具有与 a 相同的 shape。

异常：

- **ValueError** - 如果 axis 超出范围。
- **TypeError** - 如果输入 a 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.repeat(np.array(3), 4)
>>> print(output)
[3 3 3 3]
>>> x = np.array([[1,2],[3,4]])
>>> output = np.repeat(x, 2)
>>> print(output)
[1 1 2 2 3 3 4 4]
>>> output = np.repeat(x, 3, axis=1)
>>> print(output)
[[1 1 1 2 2 2]
 [3 3 3 4 4 4]]
>>> output = np.repeat(x, [1, 2], axis=0)
>>> print(output)
[[1 2]
 [3 4]
 [3 4]]
```

15.2.28 mindspore.numpy.reshape

`mindspore.numpy.reshape(x, new_shape)`
在不改变数据的情况下重塑一个 Tensor。

参数：

- **x** (Tensor) - 需要重塑的 Tensor。
- **new_shape** (Union[int, list(int), tuple(int)]) - Tensor 的新 shape。必须与原始 shape 兼容。如果 new_shape 是一个只有一个元素的 tuple，则结果将是一个具有该长度的一维 Tensor。new_shape 中的一个维度可以是-1，此时该维度的值将根据 Tensor 的总长度和其他维度的大小推断得出。

返回：

重塑后的 Tensor，其数据类型与原始 Tensor x 相同。

异常：

- **TypeError** - 如果 new_shape 既不是整数、list 或 tuple，或者 x 不是 Tensor。
- **ValueError** - 如果 new_shape 与 x 的原始 shape 不兼容。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.asarray([[ -0.1,  0.3,  3.6], [ 0.4,  0.5, -3.2]])
>>> output = np.reshape(x, (3, 2))
>>> print(output)
[[-0.1  0.3]
 [ 3.6  0.4]
 [ 0.5 -3.2]]
>>> output = np.reshape(x, (3, -1))
>>> print(output)
[[-0.1  0.3]
 [ 3.6  0.4]
 [ 0.5 -3.2]]
>>> output = np.reshape(x, (6, ))
>>> print(output)
[-0.1  0.3  3.6  0.4  0.5 -3.2]
```

15.2.29 mindspore.numpy.roll

`mindspore.numpy.roll(a, shift, axis=None)`

将 Tensor 沿给定的轴进行滚动。滚动超出最后位置的元素会重新回到第一个位置。

参数：

- **a** (Tensor) - 输入 Tensor。
- **shift** (Union[int, tuple(int)]) - 元素滚动的位移数量。如果为 tuple，则 axis 必须是相同大小的 tuple，并且每个给定的轴按对应的位移数量滚动。如果 shift 是整数，而 axis 是整数 tuple，则所有给定的轴都使用相同的位移值。
- **axis** (Union[int, tuple(int)], 可选) - 沿哪个轴或哪些轴滚动元素。默认情况下，数组在滚动前会被展平，然后恢复原始 shape。默认值: None。

返回：

Tensor，shape 与 a 相同的 Tensor。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 axis 超过 a.ndim，或者 shift 和 axis 不能广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.reshape(np.arange(12), (3, 4))
>>> print(np.roll(a, [2, -3], [0, -1]))
[[ 7  4  5  6]
 [11  8  9 10]
 [ 3  0  1  2]]
```

15.2.30 mindspore.numpy.rollaxis

`mindspore.numpy.rollaxis(x, axis, start=0)`

将指定轴向后滚动，直到它位于给定的位置。其他轴的相对位置保持不变。

参数：

- **x** (Tensor) - 要进行调换的 Tensor。
- **axis** (int) - 要滚动的轴。
- **start** (int) - 默认值: 0。如果 $start \leq axis$ ，则将轴向后滚动，直到它位于这个位置 ($start$)。如果 $start > axis$ ，则将轴滚动直到它位于此位置之前 ($start$)。如果 $start < 0$ ，则 $start$ 会被归一化为非负数（详细信息见源码）。

返回：

转置后的 Tensor，数据类型与原 Tensor x 相同。

异常：

- **TypeError** - 如果 $axis$ 或 $start$ 不是整数，或者 x 不是 Tensor。
- **ValueError** - 如果 $axis$ 不在 $[-ndim, ndim - 1]$ 范围内，或者 $start$ 不在 $[-ndim, ndim]$ 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.ones((2, 3, 4))
>>> output = np.rollaxis(x, 0, 2)
>>> print(output.shape)
(3, 2, 4)
```

15.2.31 mindspore.numpy.rot90

`mindspore.numpy.rot90(a, k=1, axes=(0, 1))`

将 Tensor 在指定的轴平面内旋转 90 度。旋转方向是从第一个轴指向第二个轴。

参数：

- **a** (Tensor) - 输入的二维或更多维的 Tensor。
- **k** (int) - Tensor 旋转 90 度的次数。默认值: 1。
- **axes** (Union[tuple(int), list(int)]，可选) - Tensor 在由轴定义的平面内进行旋转。默认值: (0, 1)。轴必须不同，并且 shape 为 (2,)。

返回：

Tensor。

异常：

- **TypeError** - 如果输入 `a` 不是 Tensor，或参数 `k` 不是整数，或参数 `axes` 不是整数的 tuple 或 list。
- **ValueError** - 如果任何轴超出范围，或 `axes` 的长度不是 2。

支持平台：

GPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.arange(24).reshape((2, 3, 4))
>>> output = np.rot90(a)
>>> print(output)
[[[ 8  9 10 11]
  [20 21 22 23]]
 [[ 4  5  6  7]
  [16 17 18 19]]
 [[ 0  1  2  3]
  [12 13 14 15]]]
>>> output = np.rot90(a, 3, (1, 2))
>>> print(output)
[[[ 8  4  0]
  [ 9  5  1]
  [10  6  2]
  [11  7  3]]
 [[20 16 12]
  [21 17 13]
  [22 18 14]
  [23 19 15]]]
```

15.2.32 mindspore.numpy.select

`mindspore.numpy.select (condlist, choicelist, default=0)`

根据条件从 `choicelist` 中的元素中返回数组。

参数：

- **condlist** (Union[int, float, bool, list, tuple, Tensor]) - 条件列表，用于确定从 `choicelist` 中哪个数组中取出输出元素。当多个条件满足时，使用 `condlist` 中遇到的第一个条件。
- **choicelist** (Union[int, float, bool, list, tuple, Tensor]) - 数组列表，从中取出输出元素。其长度必须与 `condlist` 相同。
- **default** (scalar, 可选) - 当所有条件均为 False 时插入输出的元素。默认值: 0。

返回：

Tensor，输出位置 `m` 是 `choicelist` 中数组的第 `m-th` 个元素，其中 `condlist` 对应数组的第 `m-th` 个元素为 True。

异常：

- **ValueError** - 如果 `len(condlist) != len(choicelist)`。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> condlist = [[True, True, True, False, False], [False, False, True, False, True]]
>>> choicelist = [[0, 1, 2, 3, 4], [0, 1, 4, 9, 16]]
>>> output = np.select(condlist, choicelist)
>>> print(output)
[ 0  1  2  0 16]
```

15.2.33 mindspore.numpy.setdiff1d

`mindspore.numpy.setdiff1d(ar1, ar2, assume_unique=False)`

计算两个 Tensor 的差集。返回在 `ar1` 中, 不在 `ar2` 中的唯一值。

参数:

- **ar1** (Union[int, float, bool, list, tuple, Tensor]) - 输入 Tensor。
- **ar2** (Union[int, float, bool, list, tuple, Tensor]) - 输入 Tensor。
- **assume_unique** (bool, 可选) - 如果为 `True`, 则假定输入 Tensor 没有重复元素, 这可以加速计算。如果为 `True` 但 `ar1` 或 `ar2` 有重复元素时, 可能会导致不正确的结果或超出范围的索引。默认值: `False`。

返回:

一维 Tensor, 包含在 `ar1` 中但不在 `ar2` 中的值。当 `assume_unique` 为 `False` 时, 结果会被排序, 否则仅当输入已排序时才排序。

异常:

- **TypeError** - 如果输入的 `ar1` 或 `ar2` 不是类似数组的对象。
- **TypeError** - 如果 `assume_unique` 不是 `bool` 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.array([1, 2, 3, 2, 4, 1])
>>> b = np.array([3, 4, 5, 6])
>>> np.setdiff1d(a, b)
Tensor(shape=[2], dtype=Int32, value=[1, 2])
```

15.2.34 mindspore.numpy.size

`mindspore.numpy.size(a, axis=None)`

返回沿给定轴的元素数量。

参数:

- **a** (Union[int, float, bool, list, tuple, Tensor]) - 输入数据。
- **axis** (int, 可选) - 计算元素数量的轴。默认值: `None`。如果为 `None`, 则返回总的元素数量。

返回:

沿指定轴的元素数量。

异常:

- **TypeError** - 如果输入不是类似数组的类型或 `axis` 不是整数。
- **ValueError** - 如果任意轴超出范围或存在重复的轴。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.arange(10).reshape(2, 5).astype('float32')
>>> print(np.size(x))
10
>>> print(np.size(x, axis=1))
5
```

15.2.35 mindspore.numpy.split

`mindspore.numpy.split(x, indices_or_sections, axis=0)`

将一个 Tensor 沿指定轴分割为多个 sub-tensor。

参数:

- **x** (Tensor) - 待分割的 Tensor。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 如果是整数 N , Tensor 将沿轴分割为 N 个相等的 sub-tensor。如果是 tuple(int)、list(int) 或排序后的整数, 则指示沿轴的分割位置。例如, 对于 $axis = 0$, `[2, 3]` 将产生三个 sub-tensor: `x[:2]`、`x[2:3]` 和 `x[3:]`。如果索引超出轴上数组的维度, 则相应地返回空子数组。
- **axis** (int, 可选) - 指定进行分割的轴。默认值: 0。

返回:

sub-tensor 的 tuple。

异常:

- **TypeError** - 如果参数 `indices_or_sections` 不是整数、tuple(int) 或 list(int), 或者参数 `axis` 不是整数。
- **ValueError** - 如果参数 `axis` 超出范围 `[-x.ndim, x.ndim)`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> input_x = np.arange(9).astype("float32")
>>> output = np.split(input_x, 3)
>>> print(output)
(Tensor(shape=[3], dtype=Float32,
  value= [ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00]),
 Tensor(shape=[3], dtype=Float32,
  value= [ 3.00000000e+00,  4.00000000e+00,  5.00000000e+00]),
 Tensor(shape=[3], dtype=Float32,
  value= [ 6.00000000e+00,  7.00000000e+00,  8.00000000e+00]))
```

15.2.36 mindspore.numpy.squeeze

`mindspore.numpy.squeeze(a, axis=None)`

返回删除指定 *axis* 中大小为 1 的维度后的 Tensor。

如果 *axis* = *None*，则删除所有大小为 1 的维度。如果指定了 *axis*，则删除指定 *axis* 中大小为 1 的维度。例如，如果不指定维度 *axis* = *None*，输入的 shape 为 (A, 1, B, C, 1, D)，则输出的 Tensor 的 shape 为 (A, B, C, D)。如果指定维度，squeeze 操作仅在指定维度中进行。如果输入的 shape 为 (A, 1, B)，*axis* = 0 或 *axis* = 2 时不会改变输入的 Tensor，但 *axis* = 1 时会使输入 Tensor 的 shape 变为 (A, B)。

参数：

- **a** (Tensor) - 输入 Tensor 数组。
- **axis** (Union[None, int, list(int), tuple(list)])，可选 - 要压缩的轴，默认值: *None*。

返回：

Tensor，移除了所有或部分长度为 1 的维度。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果指定的轴具有 > 1 的 shape 元素。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.ones((1, 2, 2, 1))
>>> x = np.squeeze(x)
>>> print(x.shape)
(2, 2)
```

15.2.37 mindspore.numpy.stack

`mindspore.numpy.stack(arrays, axis=0)`

沿新轴连接一系列数组。*axis* 参数指定结果中新轴的索引。例如，如果 *axis*=0，它将成为第一个维度；如果 *axis*=-1，它将成为最后一个维度。

说明：不支持 Numpy 的 *out* 参数。

参数：

- **arrays** (Tensor 的序列) - 每个数组必须具有相同的 shape。
- **axis** (int, 可选) - 结果数组中沿输入数组堆叠的轴。默认值: 0。

返回：

Tensor，堆叠后的数组比输入数组多一个维度。

异常：

- **ValueError** - 如果输入不是 Tensor、tuple 或 list。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> arrays = [np.ones((3, 4)) for _ in range(10)]
>>> output = np.stack(arrays, axis=0)
>>> print(output.shape)
(10, 3, 4)
>>> output = np.stack(arrays, axis=1)
>>> print(output.shape)
(3, 10, 4)
>>> output = np.stack(arrays, axis=2)
>>> print(output.shape)
(3, 4, 10)
```

15.2.38 mindspore.numpy.swapaxes

`mindspore.numpy.swapaxes(x, axis1, axis2)`

交换 Tensor 的两个轴。

参数:

- **x** (Tensor) - 要调换的 Tensor。
- **axis1** (int) - 第一个轴。
- **axis2** (int) - 第二个轴。

返回:

调换后的 Tensor, 其数据类型与原始 Tensor x 相同。

异常:

- **TypeError** - 如果 axis1 或 axis2 不是整数, 或 x 不是 Tensor。
- **ValueError** - 如果 axis1 或 axis2 不在 $[-ndim, ndim - 1]$ 的范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.ones((2, 3, 4))
>>> output = np.swapaxes(x, 0, 2)
>>> print(output.shape)
(4, 3, 2)
```

15.2.39 mindspore.numpy.take

`mindspore.numpy.take(a, indices, axis=None, mode='clip')`

从数组中沿指定轴提取元素。当 `axis` 不为 `None` 时，此函数的功能类似于“fancy”索引（使用数组进行索引）；但如果需要沿给定轴提取元素，它可能更易于使用。例如，`np.take(arr, indices, axis=3)` 等价于 `arr[:, :, :, indices, ...]`。

说明：不支持 Numpy 的 `out` 参数。不支持 `mode = 'raise'`，默认 `mode` 为 `'clip'`。

参数：

- **a** (Tensor) - 源数组，shape 为 (N1..., M, Nk...)。
- **indices** (Tensor) - shape 为 (Nj...) 的索引，表示要提取的值。
- **axis** (int, 可选) - 用于选择值的轴。默认情况下，使用展平后的输入数组。默认值：None。
- **mode** ('raise', 'wrap', 'clip', 可选) - 指定超出范围的索引的行为。默认值：'clip'。
 - 'raise' - 抛出错误；
 - 'wrap' - 循环；
 - 'clip' - 限制在范围内。'clip' 模式意味着所有过大的索引都将被替换为该轴最后一个元素的索引。请注意，这将禁止使用负数进行索引。

返回：

Tensor，索引结果。

异常：

- **ValueError** - 如果 `axis` 超出范围。
- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([4, 3, 5, 7, 6, 8])
>>> indices = np.array([0, 1, 4])
>>> output = np.take(a, indices)
>>> print(output)
[4 3 6]
>>> indices = np.array([[0, 1], [2, 3]])
>>> output = np.take(a, indices)
>>> print(output)
[[4 3]
 [5 7]]
```


15.2.40 mindspore.numpy.take_along_axis

`mindspore.numpy.take_along_axis(arr, indices, axis)`

根据一维索引和数据切片从输入数组中提取值。该函数沿指定的轴在索引和数据数组中迭代匹配一维切片，并使用前者在后者中查找值。这些切片可以具有不同的长度。

参数：

- **arr** (Tensor) - 源数组，shape 为 $(N_1 \cdots, M, N_k \cdots)$ 。
- **indices** (Tensor) - shape 为 $(N_1 \cdots, J, N_k \cdots)$ 的索引，用于沿 arr 的每个一维切片取值。必须与 arr 的维度匹配，但维度 N_i 和 N_j 只需要与 arr 进行广播。
- **axis** (int) - 沿该轴进行一维切片取值。如果 axis 为 None，则输入数组将被视作首先被展平为一维。

返回：

Tensor，索引结果，shape 为 $(N_1 \cdots, J, N_k \cdots)$ 。

异常：

- **ValueError** - 如果输入数组和索引的维度数量不同。
- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.arange(12).reshape(3, 4)
>>> indices = np.arange(3).reshape(1, 3)
>>> output = np.take_along_axis(x, indices, 1)
>>> print(output)
[[ 0  1  2]
 [ 4  5  6]
 [ 8  9 10]]
```

15.2.41 mindspore.numpy.tile

`mindspore.numpy.tile(a, reps)`

通过重复 a 指定次数构造一个数组，次数由 reps 给出。如果 reps 的长度为 d，结果将具有 $\max(d, a.\text{ndim})$ 的维度。如果 $a.\text{ndim} < d$ ，则通过在前面添加新轴将 a 提升为 d 维。因此，shape 为 (3,) 的数组会被提升为 (1, 3) 以进行二维复制，或 (1, 1, 3) 以进行三维复制。如果这不是所需的行为，可以在调用此函数之前手动将 a 提升为 d 维。如果 $a.\text{ndim} > d$ ，则通过在前面添加 1 来提升 reps 为 $a.\text{ndim}$ 。因此，对于 shape 为 (2, 3, 4, 5) 的 a，reps 为 (2, 2) 被视为 (1, 1, 2, 2)。

参数：

- **a** (Tensor) - 输入数组。
- **reps** (int, ints 的序列) - 在每个轴上重复 a 的次数。

返回：

Tensor，重复后的输出数组。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.array([0, 1, 2])
>>> output = np.tile(a, 2)
>>> print(output)
[0 1 2 0 1 2]
>>> output = np.tile(a, (2, 2))
>>> print(output)
[[0 1 2 0 1 2]
 [0 1 2 0 1 2]]
>>> output = np.tile(a, (2, 1, 2))
>>> print(output)
[[[0 1 2 0 1 2]]
 [[0 1 2 0 1 2]]]
```

15.2.42 mindspore.numpy.transpose

`mindspore.numpy.transpose(a, axes=None)`

反转或交换 Tensor 的轴; 返回修改后的 Tensor。

参数:

- **a** (Tensor) - 要转置的 Tensor。
- **axes** (Union[None, tuple, list], 可选) - 轴的顺序。如果 axes 为 None, 则转置整个 Tensor。默认值: None。

返回:

Tensor, 转置后的 Tensor 数组。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。
- **ValueError** - 如果 axes 的数量不等于 a.ndim。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.ones((1, 2, 3))
>>> x = np.transpose(x)
>>> print(x.shape)
(3, 2, 1)
```

15.2.43 mindspore.numpy.unique

`mindspore.numpy.unique(x, return_inverse=False)`

返回去重后的 Tensor 元素。当输入 Tensor 有多个维度时，将首先将其展平。

说明：不支持 Numpy 的 `axis`、`return_index` 和 `return_counts` 参数。在 CPU 上，该操作必须在图模式下执行。

参数：

- **x** (Tensor) - 要处理的输入 Tensor。
- **return_inverse** (bool, 可选) - 如果为 True，还返回唯一 Tensor 的索引。默认值：False。

返回：

Tensor 或 Tensor 的 tuple。如果 `return_inverse` 为 False，返回唯一 Tensor；否则，返回 Tensor 的 tuple。

异常：

- **TypeError** - 如果 x 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> import mindspore as ms
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> input_x = np.asarray([1, 2, 2, 2, 3, 4, 5]).astype('int32')
>>> output_x = np.unique(input_x)
>>> print(output_x)
[1 2 3 4 5]
>>> output_x = np.unique(input_x, return_inverse=True)
>>> print(output_x)
(Tensor(shape=[5], dtype=Int32, value= [ 1, 2, 3, 4, 5]), Tensor(shape=[7], dtype=Int32,
value= [0, 1, 1, 1, 2, 3, 4]))
```

15.2.44 mindspore.numpy.unravel_index

`mindspore.numpy.unravel_index(indices, shape, order='C')`

将一维索引或一维索引数组转换为坐标数组的 tuple。

说明：超出边界的索引会被裁剪到 shape 的边界，而不是引发错误。

参数：

- **indices** (Union[int, float, bool, list, tuple, Tensor]) - 一个整数数组，其元素是展平后的数组中对应 shape 维度位置的索引。
- **shape** (tuple(int)) - 用于将索引转换为坐标的数组 shape。
- **order** (Union['C', 'F'], 可选) - 确定索引是否按行主序（C-style）或列主序（Fortran-style）进行处理。默认值：'C'。

返回：

Tensor，包含坐标数组的 tuple，每个数组的 shape 与索引数组相同。

异常：

- **ValueError** - 如果 order 不是 'C' 或 'F'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.unravel_index([22, 41, 37], (7,6)))
(Tensor(shape=[3], dtype=Int32, value= [3, 6, 6]),
 Tensor(shape=[3], dtype=Int32, value= [4, 5, 1]))
>>> print(np.unravel_index([31, 41, 13], (7,6), order='F'))
(Tensor(shape=[3], dtype=Int32, value= [3, 6, 6]),
 Tensor(shape=[3], dtype=Int32, value= [4, 5, 1]))
```

15.2.45 mindspore.numpy.vsplit

`mindspore.numpy.vsplit(x, indices_or_sections)`

垂直（按行）将 Tensor 分割为多个 sub-tensor。此操作等同于使用 `axis = 0`（默认值）进行分割，不论数组的维度如何，始终沿第一轴分割。

参数：

- **x** (Tensor) - 待分割的 Tensor。
- **indices_or_sections** (Union[int, tuple(int), list(int)]) - 如果是整数 N ，Tensor 将沿轴分割为 N 个相等的 sub-tensor。如果是 tuple(int)、list(int) 或排序后的整数，则指示沿轴的分割位置。例如，对于 `axis = 0`，`[2, 3]` 将产生三个 sub-tensor: `x[:2]`、`x[2:3]` 和 `x[3:]`。如果索引超出轴上数组的维度，则相应地返回空子数组。

返回：

sub-tensor 列表。

异常：

- **TypeError** - 如果参数 `indices_or_sections` 不是整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input_x = np.arange(9).reshape((3, 3)).astype('float32')
>>> output = np.vsplit(input_x, 3)
>>> print(output)
(Tensor(shape=[1, 3], dtype=Float32,
 value=[[ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00]]),
 Tensor(shape=[1, 3], dtype=Float32,
 value=[[ 3.00000000e+00,  4.00000000e+00,  5.00000000e+00]]),
 Tensor(shape=[1, 3], dtype=Float32,
 value=[[ 6.00000000e+00,  7.00000000e+00,  8.00000000e+00]]))
```

15.2.46 mindspore.numpy.vstack

`mindspore.numpy.vstack (tup)`

按顺序垂直堆叠 Tensor。此操作等同于沿第一轴进行连接。对于一维 Tensor，应首先将其 shape 变为 (1, N)，然后沿第一轴进行连接。

参数：

- **tup** (Union[Tensor, tuple, list]) - 一组一维或二维 Tensor。Tensor 必须在除第一轴之外的所有轴上具有相同的 shape。一维 Tensor 必须具有相同的 shape。

返回：

堆叠后的 Tensor，由给定的 Tensor 堆叠而成。

异常：

- **TypeError** - 如果 tup 不是 Tensor、list 或 tuple。
- **ValueError** - 如果 tup 为空。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.array([1, 2, 3]).astype('int32')
>>> x2 = np.array([4, 5, 6]).astype('int32')
>>> output = np.vstack((x1, x2))
>>> print(output)
[[1 2 3]
 [4 5 6]]
```

15.2.47 mindspore.numpy.where

`mindspore.numpy.where (condition, x=None, y=None)`

根据 condition 从 x 或 y 中选择元素。

说明：由于不支持 nonzero，x 和 y 必须都是 Tensor 输入。

参数：

- **condition** (Tensor) - 当为 True 时，选择 x 中的值，否则选择 y 中的值。
- **x** (Tensor, 可选) - 选择值的来源。默认值：None。
- **y** (Tensor, 可选) - 选择值的来源。x，y 和 condition 需要能够广播到相同的 shape。默认值：None。

返回：

Tensor 或标量，其中 condition 为 True 的位置取自 x，其他位置取自 y。

异常：

- **ValueError** - 如果操作数不能广播到相同的 shape。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> condition = np.full((1, 1, 2), [False, True])
>>> x = np.full((1, 3, 2), 5)
>>> y = np.full((2, 1, 1), 7)
>>> output = np.where(condition, x, y)
>>> print(output)
[[[7 5]
  [7 5]
  [7 5]]
 [[7 5]
  [7 5]
  [7 5]]]
```

15.3 逻辑运算

逻辑运算类算子主要进行各类逻辑相关的运算。

相等 (equal) 和小于 (less) 计算代码示例如下:

```
input_x = np.arange(0, 5)
input_y = np.arange(0, 10, 2)
output = np.equal(input_x, input_y)
print("output of equal:", output)
output = np.less(input_x, input_y)
print("output of less:", output)
```

运行结果如下:

```
output of equal: [ True False False False False]
output of less: [False True True True True]
```

接口名	概述	支持平台
<code>mindspore.numpy.array_equal</code>	当输入数组 <code>shape</code> 相同且所有元素相等时, 返回 <code>True</code> 。	GPU CPU Ascend
<code>mindspore.numpy.array_equiv</code>	当输入数组 <code>shape</code> 一致且所有元素相等时, 返回 <code>True</code> 。	Ascend GPU CPU
<code>mindspore.numpy.equal</code>	逐元素返回 $(x1 == x2)$ 的真值。	Ascend GPU CPU
<code>mindspore.numpy.greater</code>	逐元素返回 $(x1 > x2)$ 的真值。	Ascend GPU CPU
<code>mindspore.numpy.greater_equal</code>	逐元素返回 $(x1 \geq x2)$ 的真值。	Ascend GPU CPU
<code>mindspore.numpy.in1d</code>	测试一维数组的每个元素是否也存在于第二个数组中。	Ascend GPU CPU
<code>mindspore.numpy.isclose</code>	返回一个 <code>bool</code> 类型的 <code>Tensor</code> , 用于表示两个 <code>Tensor</code> 在给定的容差范围内是否逐元素相等。	Ascend GPU CPU
<code>mindspore.numpy.isfinite</code>	逐元素测试是否为有限数 (不是无穷大或非数值)。	Ascend GPU CPU
<code>mindspore.numpy.isin</code>	在 <code>test_elements</code> 中的元素上计算, 并仅在 <code>element</code> 上进行广播。	Ascend GPU CPU
<code>mindspore.numpy.isinf</code>	逐元素测试是否为正无穷大或负无穷大。	GPU CPU
<code>mindspore.numpy.isnan</code>	逐元素测试是否为 NaN, 并将结果返回为 <code>bool</code> 数组。	GPU CPU

下页继续

表 3 - 续上页

<code>mindspore.numpy.isneginf</code>	逐元素测试是否为负无穷大, 并将结果返回为 <code>bool</code> 数组。	GPU CPU
<code>mindspore.numpy.isposinf</code>	逐元素测试是否为正无穷大, 并将结果返回为 <code>bool</code> 数组。	GPU CPU
<code>mindspore.numpy.isscalar</code>	如果元素的类型是标量类型, 则返回 <code>True</code> 。	Ascend GPU CPU
<code>mindspore.numpy.less</code>	逐元素返回 $(x1 < x2)$ 的真值。	Ascend GPU CPU
<code>mindspore.numpy.less_equal</code>	逐元素返回 $(x1 \leq x2)$ 的真值。	Ascend GPU CPU
<code>mindspore.numpy.logical_and</code>	逐元素计算 <code>x1</code> 和 <code>x2</code> 的逻辑与 (AND) 的真值。	Ascend GPU CPU
<code>mindspore.numpy.logical_not</code>	逐元素计算 <code>a</code> 的逻辑非 (NOT) 的真值。	Ascend GPU CPU
<code>mindspore.numpy.logical_or</code>	逐元素计算 <code>x1</code> 和 <code>x2</code> 的逻辑或 (OR) 的真值。	Ascend GPU CPU
<code>mindspore.numpy.logical_xor</code>	逐元素计算 <code>x1</code> 和 <code>x2</code> 的逻辑异或 (XOR) 的真值。	Ascend GPU CPU
<code>mindspore.numpy.not_equal</code>	逐元素返回 $(x1 \neq x2)$ 的真值。	Ascend GPU CPU
<code>mindspore.numpy.signbit</code>	逐元素扫描元素的符号位, 如果符号位为 1 (即元素小于 0) 则返回 <code>True</code> 。	Ascend GPU CPU
<code>mindspore.numpy.sometrue</code>	测试沿给定轴是否有任意数组元素为 <code>True</code> 。	Ascend GPU CPU

15.3.1 mindspore.numpy.array_equal

`mindspore.numpy.array_equal(a1, a2, equal_nan=False)`
当输入数组 `shape` 相同且所有元素相等时, 返回 `True`。

说明: 在 MindSpore 中, 会返回一个 `bool tensor`, 因为在图模式下, 该值无法在编译时追踪和计算。由于 Ascend 平台对 `nan` 的处理不同, 目前在 Ascend 上不支持 `equal_nan` 参数。

参数:

- **a1/a2** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入数组。
- **equal_nan** (`bool`, 可选) - 是否将 `NaN` 视为相等。默认值: `False`。

返回:

标量 `bool tensor`, 如果输入相等, 值为 `True`, 否则为 `False`。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。

支持平台:

GPU CPU Ascend

样例:

```
>>> import mindspore.numpy as np
>>> a = [0,1,2]
>>> b = [[0,1,2], [0,1,2]]
>>> print(np.array_equal(a,b))
False
```

15.3.2 mindspore.numpy.array_equiv

`mindspore.numpy.array_equiv(a1, a2)`

当输入数组 `shape` 一致且所有元素相等时, 返回 `True`。`shape` 一致意味着它们要么具有相同的 `shape`, 要么可以通过广播使其中一个输入数组与另一个具有相同的 `shape`。

说明: 在 MindSpore 中, 会返回一个 `bool tensor`, 因为在图模式下, 该值无法在编译时追踪和计算。

参数:

- **a1/a2** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入数组。

返回:

Scalar `bool tensor`, 如果输入等效, 值为 `True`, 否则为 `False`。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = [0,1,2]
>>> b = [[0,1,2], [0,1,2]]
>>> print(np.array_equiv(a,b))
True
```

15.3.3 mindspore.numpy.equal

`mindspore.numpy.equal(x1, x2, dtype=None)`

逐元素返回 `(x1 == x2)` 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (`Tensor`) - 输入数组。
- **x2** (`Tensor`) - 输入数组。如果 `x1.shape != x2.shape`, 则它们必须能够广播到一个共同的 `shape` (该 `shape` 成为输出的 `shape`)。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 `Tensor` 的 `dtype`。

返回:

`Tensor` 或标量, `x1` 和 `x2` 逐元素比较的结果。通常为 `bool` 类型, 除非传递了 `dtype`。如果 `x1` 和 `x2` 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.equal(np.array([0, 1, 3]), np.arange(3))
>>> print(output)
[ True  True False]
```

15.3.4 mindspore.numpy.greater

`mindspore.numpy.greater(x1, x2, dtype=None)`

逐元素返回 $(x1 > x2)$ 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。如果 $x1.shape \neq x2.shape$ ，则它们必须能够广播到一个共同的 shape (该 shape 成为输出的 shape)。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, $x1$ 和 $x2$ 逐元素比较的结果。通常为 bool 类型, 除非传递了 dtype。如果 $x1$ 和 $x2$ 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.greater(np.array([4, 2]), np.array([2, 2]))
>>> print(output)
[ True False]
```

15.3.5 mindspore.numpy.greater_equal

`mindspore.numpy.greater_equal(x1, x2, dtype=None)`

逐元素返回 $(x1 \geq x2)$ 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。如果 $x1.shape \neq x2.shape$ ，则它们必须能够广播到一个共同的 shape (该 shape 成为输出的 shape)。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量，x1 和 x2 逐元素比较的结果。通常为 bool 类型，除非传递了 dtype。如果 x1 和 x2 都是标量，则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.greater_equal(np.array([4, 2, 1]), np.array([2, 2, 2]))
>>> print(output)
[ True  True False]
```

15.3.6 mindspore.numpy.in1d

`mindspore.numpy.in1d(ar1, ar2, invert=False)`

测试一维数组的每个元素是否也存在于第二个数组中。返回与 ar1 长度相同的 boolean 数组，当 ar1 的某个元素存在于 ar2 中时，该位置为 True，否则为 False。

说明：不支持 Numpy 的 `assume_unique` 参数，因为该实现不依赖于输入数组的唯一性。

参数：

- **ar1** (Union[int, float, bool, list, tuple, Tensor]) - shape 为 (M,) 的输入数组。
- **ar2** (Union[int, float, bool, list, tuple, Tensor]) - 用于测试 ar1 中每个值的数组。
- **invert** (boolean, 可选) - 如果为 True，返回数组中的值将被反转（即当 ar1 的元素在 ar2 中时为 False，否则为 True）。默认值：False。

返回：

Tensor，shape 为 (M,)。值为 ar1 中元素是否存在于 ar2 中的真值。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> test = np.array([0, 1, 2, 5, 0])
>>> states = [0, 2]
>>> mask = np.in1d(test, states)
>>> print(mask)
[ True False  True False  True]
>>> mask = np.in1d(test, states, invert=True)
>>> print(mask)
[False  True False  True False]
```

15.3.7 mindspore.numpy.isclose

`mindspore.numpy.isclose(a, b, rtol=1e-05, atol=1e-08, equal_nan=False)`

返回一个 `bool` 类型的 `Tensor`，用于表示两个 `Tensor` 在给定的容差范围内是否逐元素相等。容差值为正数，通常是非常小的数字。相对差异 ($rtol * abs(b)$) 和绝对差异 `atol` 相加后，与 `a` 和 `b` 的绝对差异进行比较。

说明：对于有限值，`isclose` 使用以下公式来测试两个浮点数是否相等： $absolute(a - b) \leq (atol + rtol * absolute(b))$ 。在 `Ascend` 平台上，不支持包含 `inf` 或 `NaN` 的输入数组。

参数：

- **a** (`Union[Tensor, list, tuple]`) - 要比较的第一个输入 `Tensor`。
- **b** (`Union[Tensor, list, tuple]`) - 要比较的第二个输入 `Tensor`。
- **rtol** (`numbers.Number`，可选) - 相对容差参数（见说明）。默认值：`1e-05`。
- **atol** (`numbers.Number`，可选) - 绝对容差参数（见说明）。默认值：`1e-08`。
- **equal_nan** (`bool`，可选) - 是否将 `NaN` 视为相等。如果为 `True`，`a` 中的 `NaN` 在输出 `Tensor` 中将被视为与 `b` 中的 `NaN` 相等。默认值：`False`。

返回：

在给定的容差范围内，表示 `a` 和 `b` 是否相等的 `bool` 类型 `Tensor`。

异常：

- **TypeError** - 如果输入参数非上述给定的类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([0,1,2,float('inf'),float('inf'),float('nan')])
>>> b = np.array([0,1,-2,float('-inf'),float('inf'),float('nan')])
>>> print(np.isclose(a, b))
[ True  True False False  True False]
>>> print(np.isclose(a, b, equal_nan=True))
[ True  True False False  True  True]
```

15.3.8 mindspore.numpy.isfinite

`mindspore.numpy.isfinite(x, dtype=None)`

逐元素测试是否为有限数（不是无穷大或非数值）。结果将以 `bool` 数组的形式返回。

说明：不支持 `Numpy` 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。在 `GPU` 上，支持的 `dtype` 为 `np.float16` 和 `np.float32`。

参数：

- **x** (`Tensor`) - 输入值。

- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 如果 x 不是正无穷大、负无穷大或 NaN, 则返回为 True; 否则为 False。如果 x 是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.isfinite(np.array([np.inf, 1., np.nan]).astype('float32'))
>>> print(output)
[False True False]
```

15.3.9 mindspore.numpy.isin

mindspore.numpy.isin (element, test_elements, invert=False)

在 test_elements 中的元素上计算, 并仅在 element 上进行广播。返回一个与 element 形状相同的 bool 数组, 其中 element 的元素在 test_elements 中时为 True, 否则为 False。

说明: 由于实现不依赖于输入数组的唯一性, 因此不支持 NumPy 的 assume_unique 参数。

参数:

- **element** (Union[int, float, bool, list, tuple, Tensor]) - 输入数组。
- **test_elements** (Union[int, float, bool, list, tuple, Tensor]) - 用于测试 element 中每个值的对比值。
- **invert** (boolean, 可选) - 如果为 True, 返回数组中的值将取反, 相当于计算 element 不在 test_elements 中的情况。默认值: False。

返回:

Tensor, 与 element 具有相同的 shape。element[isin] 的值在 test_elements 中。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> element = 2*np.arange(4).reshape((2, 2))
>>> test_elements = [1, 2, 4, 8]
>>> mask = np.isin(element, test_elements)
>>> print(mask)
[[False True]
 [ True False]]
>>> mask = np.isin(element, test_elements, invert=True)
>>> print(mask)
[[ True False]
 [False  True]]
```

15.3.10 mindspore.numpy.isinf

`mindspore.numpy.isinf(x, dtype=None)`

逐元素测试是否为正无穷大或负无穷大。返回一个与 `x` 具有相同 `shape` 的 `bool` 数组，当 `x == +/-inf` 时为 `True`，否则为 `False`。

说明：不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。目前仅支持 `np.float32`。

参数：

- `x` (Tensor) - 输入值。
- `dtype` (`mindspore.dtype`, 可选) - 默认值：None。覆盖输出 Tensor 的 `dtype`。

返回：

Tensor 或标量，当 `x` 为正无穷大或负无穷大时为 `True`，否则为 `False`。如果 `x` 是标量，则返回标量。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.isinf(np.array(np.inf, np.float32))
>>> print(output)
True
>>> output = np.isinf(np.array([np.inf, -np.inf, 1.0, np.nan], np.float32))
>>> print(output)
[ True  True False False]
```

15.3.11 mindspore.numpy.isnan

`mindspore.numpy.isnan(x, dtype=None)`

逐元素测试是否为 NaN，并将结果返回为 `bool` 数组。

说明：不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。目前仅支持 `np.float32`。

参数：

- `x` (Tensor) - 输入值。
- `dtype` (`mindspore.dtype`, 可选) - 默认值：None。覆盖输出 Tensor 的 `dtype`。

返回：

Tensor 或标量，当 `x` 为 NaN 时为 `True`，否则为 `False`。如果 `x` 是标量，则返回标量。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.isnan(np.array(np.nan, np.float32))
>>> print(output)
True
>>> output = np.isnan(np.array(np.inf, np.float32))
>>> print(output)
False
```

15.3.12 mindspore.numpy.isneginf

`mindspore.numpy.isneginf(x)`

逐元素测试是否为负无穷大，并将结果返回为 `bool` 数组。

说明：不支持 Numpy 的 `out` 参数。目前仅支持 `np.float32`。

参数：

- **x** (Tensor) - 输入值。

返回：

Tensor 或标量，当 `x` 为负无穷大时为 `True`，否则为 `False`。如果 `x` 是标量，则返回标量。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.isneginf(np.array([-np.inf, 0., np.inf, np.nan], np.float32))
>>> print(output)
[ True False False False]
```

15.3.13 mindspore.numpy.isposinf

`mindspore.numpy.isposinf(x)`

逐元素测试是否为正无穷大，并将结果返回为 `bool` 数组。

说明：不支持 Numpy 的 `out` 参数。目前仅支持 `np.float32`。

参数：

- **x** (Tensor) - 输入值。

返回：

Tensor 或标量，当 `x` 为正无穷大时为 `True`，否则为 `False`。如果 `x` 是标量，则返回标量。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.isposinf(np.array([-np.inf, 0., np.inf, np.nan], np.float32))
>>> print(output)
[False False  True False]
```

15.3.14 mindspore.numpy.isscalar

`mindspore.numpy.isscalar(element)`

如果元素的类型是标量类型，则返回 `True`。

说明：仅支持 MindSpore 解析器识别的对象类型，包括在 MindSpore 范围内定义的对象、类型、方法和函数。不支持其他内置类型。

参数：

- **element** (any) - 输入参数，可以是任意类型和 shape。

返回：

Boolean，如果 `element` 是标量类型，则返回 `True`，否则返回 `False`。

异常：

- **TypeError** - 如果 `element` 的类型不受 MindSpore 解析器支持。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.isscalar(3.1)
>>> print(output)
True
>>> output = np.isscalar(np.array(3.1))
>>> print(output)
False
>>> output = np.isscalar(False)
>>> print(output)
True
>>> output = np.isscalar('numpy')
>>> print(output)
True
```

15.3.15 mindspore.numpy.less

`mindspore.numpy.less(x1, x2, dtype=None)`
逐元素返回 $(x1 < x2)$ 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。如果 $x1.shape \neq x2.shape$ ，则它们必须能够广播到一个共同的 shape（该 shape 成为输出的 shape）。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, $x1$ 和 $x2$ 逐元素比较的结果。通常为 bool 类型, 除非传递了 dtype。如果 $x1$ 和 $x2$ 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.less(np.array([1, 2]), np.array([2, 2]))
>>> print(output)
[ True False]
```

15.3.16 mindspore.numpy.less_equal

`mindspore.numpy.less_equal(x1, x2, dtype=None)`
逐元素返回 $(x1 \leq x2)$ 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。如果 $x1.shape \neq x2.shape$ ，则它们必须能够广播到一个共同的 shape（该 shape 成为输出的 shape）。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, $x1$ 和 $x2$ 逐元素比较的结果。通常为 bool 类型, 除非传递了 dtype。如果 $x1$ 和 $x2$ 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.less_equal(np.array([4, 2, 1]), np.array([2, 2, 2]))
>>> print(output)
[False True True]
```

15.3.17 mindspore.numpy.logical_and

`mindspore.numpy.logical_and(x1, x2, dtype=None)`

逐元素计算 $x1$ 和 $x2$ 的逻辑与 (AND) 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 输入 Tensor。
- **x2** (Tensor) - 输入 Tensor。如果 $x1.shape \neq x2.shape$ ，则它们必须能够广播到一个共同的 shape (该 shape 成为输出的 shape)。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。对 $x1$ 和 $x2$ 中的元素执行逻辑与 (AND) 操作的 Boolean 结果; shape 由广播确定。如果 $x1$ 和 $x2$ 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x1 = np.array([True, False])
>>> x2 = np.array([False, False])
>>> output = np.logical_and(x1, x2)
>>> print(output)
[False False]
```

15.3.18 mindspore.numpy.logical_not

`mindspore.numpy.logical_not(a, dtype=None)`

逐元素计算 a 的逻辑非 (NOT) 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **a** (Tensor) - 输入 Tensor, 其 dtype 为 bool。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。对 a 中的元素执行逻辑非（NOT）操作的 Boolean 结果，具有与 a 相同的 shape。如果 a 是标量，则返回标量。

异常：

- **TypeError** - 如果输入不是 Tensor 或其 dtype 不是 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([True, False])
>>> output = np.logical_not(a)
>>> print(output)
[False  True]
```

15.3.19 mindspore.numpy.logical_or

`mindspore.numpy.logical_or(x1, x2, dtype=None)`

逐元素计算 x1 和 x2 的逻辑或（OR）的真值。

说明：不支持 Numpy 的 out、where、casting、order、subok、signature 和 extobj 参数。

参数：

- **x1** (Tensor) - 输入 Tensor。
- **x2** (Tensor) - 输入 Tensor。如果 $x1.shape \neq x2.shape$ ，则它们必须能够广播到一个共同的 shape（该 shape 成为输出的 shape）。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量，x1 和 x2 的逐元素逻辑或（OR）的比较结果。通常为 bool 类型，除非传入 dtype=object。如果 x1 和 x2 都是标量，则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.array([True, False])
>>> x2 = np.array([False, True])
>>> output = np.logical_or(x1, x2)
>>> print(output)
[ True  True]
```

15.3.20 mindspore.numpy.logical_xor

`mindspore.numpy.logical_xor(x1, x2, dtype=None)`
逐元素计算 $x1$ 和 $x2$ 的逻辑异或 (XOR) 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 输入 Tensor。
- **x2** (Tensor) - 输入 Tensor。如果 $x1.shape \neq x2.shape$, 则它们必须能够广播到一个共同的 shape (该 shape 成为输出的 shape)。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。对 $x1$ 和 $x2$ 中的元素执行逻辑异或 (XOR) 操作的 Boolean 结果; shape 由广播确定。如果 $x1$ 和 $x2$ 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x1 = np.array([True, False])
>>> x2 = np.array([False, False])
>>> output = np.logical_xor(x1, x2)
>>> print(output)
[True False]
```

15.3.21 mindspore.numpy.not_equal

`mindspore.numpy.not_equal(x1, x2, dtype=None)`
逐元素返回 $(x1 \neq x2)$ 的真值。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 第一个待比较的输入 Tensor。
- **x2** (Tensor) - 第二个待比较的输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 表示 $x1$ 和 $x2$ 逐元素比较的结果。通常为 bool 类型, 除非传入 dtype。如果 $x1$ 和 $x2$ 都是标量, 则返回标量。

异常:

- **TypeError** - 如果输入不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.asarray([1, 2])
>>> b = np.asarray([[1, 3], [1, 4]])
>>> print(np.not_equal(a, b))
[[False  True]
 [False  True]]
```

15.3.22 mindspore.numpy.signbit

`mindspore.numpy.signbit(x, dtype=None)`

逐元素扫描元素的符号位，如果符号位为 1（即元素小于 0）则返回 `True`。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入值。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor。

异常:

- **TypeError** - 如果输入不是类似数组的对象，或者 `dtype` 不是 `None` 或 `bool`。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([1, -2.3, 2.1]).astype('float32')
>>> output = np.signbit(x)
>>> print(output)
[False  True False]
```

15.3.23 mindspore.numpy.sometrue

`mindspore.numpy.sometrue(a, axis=None, keepdims=False)`

测试沿给定轴是否有任意数组元素为 `True`。默认返回单一 `boolean`，除非 `axis` 不为 `None`。

参数：

- **a** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入 `Tensor` 或可以转换为数组的对象。
- **axis** (`Union[None, int, tuple(int)]`) - 执行逻辑或 (OR) 操作的轴或轴的 `tuple`。默认值: `None`。如果为 `None`，则对输入数组的所有维度进行逻辑或 (OR) 操作。如果为负数，则从最后一个轴到第一个轴进行计算。如果为整数 `tuple`，则在多个轴上进行操作，而不是单个轴或所有轴。
- **keepdims** (`bool`) - 默认值: `False`。如果为 `True`，则保留被处理的轴作为大小为 1 的维度。启用此选项后，结果将能够与输入数组正确广播。如果传递了默认值，则 `keepdims` 不会传递给 `ndarray` 子类的任何方法，但任何非默认值会传递。如果子类方法不实现 `keepdims`，可能会引发异常。

返回：

返回单一 `boolean`，除非 `axis` 不为 `None`。

异常：

- **TypeError** - 如果输入不是类似数组的对象，或者 `axis` 不是整数或整数 `tuple`，或者 `keepdims` 不是 `bool` 类型。
- **ValueError** - 如果任何轴超出范围或存在重复轴。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.array([1, -2.3, 2.1]).astype('float32')
>>> output = np.sometrue(x)
>>> print(output)
True
```

15.4 数学运算

数学运算类算子包括各类数学相关的运算：加减乘除乘方，以及指数、对数等常见函数等。

数学计算支持类似 NumPy 的广播特性。

- 加法

以下代码实现了 `input_x` 和 `input_y` 两数组相加的操作：

```
input_x = np.full((3, 2), [1, 2])
input_y = np.full((3, 2), [3, 4])
output = np.add(input_x, input_y)
print(output)
```

运行结果如下：

```
[[4 6]
 [4 6]
 [4 6]]
```

• 矩阵乘法

以下代码实现了 *input_x* 和 *input_y* 两矩阵相乘的操作：

```
input_x = np.arange(2*3).reshape(2, 3).astype('float32')
input_y = np.arange(3*4).reshape(3, 4).astype('float32')
output = np.matmul(input_x, input_y)
print(output)
```

运行结果如下：

```
[[20. 23. 26. 29.]
 [56. 68. 80. 92.]]
```

• 求平均值

以下代码实现了求 *input_x* 所有元素的平均值的操作：

```
input_x = np.arange(6).astype('float32')
output = np.mean(input_x)
print(output)
```

运行结果如下：

```
2.5
```

• 指数

以下代码实现了自然常数 *e* 的 *input_x* 次方的操作：

```
input_x = np.arange(5).astype('float32')
output = np.exp(input_x)
print(output)
```

运行结果如下：

```
[ 1.          2.7182817  7.389056  20.085537  54.59815 ]
```

接口名	概述	支持平台
<i>mindspore.numpy.absolute</i>	逐元素计算绝对值。	Ascend GPU CPU
<i>mindspore.numpy.add</i>	逐元素相加两个参数。	Ascend GPU CPU
<i>mindspore.numpy.amax</i>	返回数组的最大值或沿指定轴的最大值。	Ascend GPU CPU
<i>mindspore.numpy.amin</i>	返回数组的最小值或沿指定轴的最小值。	Ascend GPU CPU
<i>mindspore.numpy.arccos</i>	逐元素计算反余弦函数。	Ascend GPU CPU
<i>mindspore.numpy.arccosh</i>	逐元素计算反双曲余弦函数。	Ascend GPU CPU
<i>mindspore.numpy.arcsin</i>	逐元素计算反正弦函数。	Ascend GPU CPU
<i>mindspore.numpy.arcsinh</i>	逐元素计算反双曲正弦函数。	Ascend GPU CPU
<i>mindspore.numpy.arctan</i>	逐元素计算反正切函数。	Ascend GPU CPU
<i>mindspore.numpy.arctan2</i>	逐元素计算 <i>x1/x2</i> 的反正切，并正确选择象限。	Ascend GPU CPU
<i>mindspore.numpy.arctanh</i>	逐元素计算反双曲正切。	Ascend CPU
<i>mindspore.numpy.argmax</i>	返回沿指定轴最大值的索引。	Ascend GPU CPU
<i>mindspore.numpy.argmin</i>	返回沿指定轴最小值的索引。	Ascend GPU CPU
<i>mindspore.numpy.around</i>	向给定的小数位数四舍五入。	Ascend GPU CPU
<i>mindspore.numpy.average</i>	沿指定轴计算加权平均值。	Ascend GPU CPU

[下页继续](#)

表 4 - 续上页

<code>mindspore.numpy.bincount</code>	计算数组中非负 <code>int</code> 值的出现次数。	Ascend GPU CPU
<code>mindspore.numpy.bitwise_and</code>	逐元素计算两个数组的按位与运算。	Ascend CPU
<code>mindspore.numpy.bitwise_or</code>	逐元素计算两个数组的按位或运算。	Ascend CPU
<code>mindspore.numpy.bitwise_xor</code>	逐元素计算两个数组的按位异或运算。	Ascend CPU
<code>mindspore.numpy.cbrt</code>	返回 <code>Tensor</code> 的立方根, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.ceil</code>	返回输入的向上取整结果, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.clip</code>	裁剪 (限制) 数组的值。	Ascend GPU CPU
<code>mindspore.numpy.convolve</code>	返回两个一维序列的离散线性卷积。	GPU CPU
<code>mindspore.numpy.copysign</code>	将 <code>x1</code> 的符号更改为 <code>x2</code> 的符号, 逐元素执行。	Ascend GPU CPU
<code>mindspore.numpy.corrcoef</code>	返回皮尔逊积矩相关系数。	Ascend GPU CPU
<code>mindspore.numpy.correlate</code>	两个一维序列的互相关。	Ascend GPU CPU
<code>mindspore.numpy.cos</code>	逐元素计算余弦。	Ascend GPU CPU
<code>mindspore.numpy.cosh</code>	逐元素计算双曲余弦。	Ascend CPU
<code>mindspore.numpy.count_nonzero</code>	计算 <code>Tensor x</code> 中的非零值数量。	Ascend GPU CPU
<code>mindspore.numpy.cov</code>	给定数据和权重, 估算一个协方差矩阵。	Ascend GPU CPU
<code>mindspore.numpy.cross</code>	返回两个向量 (数组) 的叉积。	Ascend GPU CPU
<code>mindspore.numpy.cumprod</code>	返回沿给定 <code>axis</code> 的元素的累计乘积。	Ascend GPU
<code>mindspore.numpy.cumsum</code>	返回沿给定 <code>axis</code> 的元素的累计和。	Ascend GPU CPU
<code>mindspore.numpy.deg2rad</code>	将角度从角度制转换为弧度制。	Ascend GPU CPU
<code>mindspore.numpy.diff</code>	计算沿给定 <code>axis</code> 的 <code>n</code> 阶离散差分。	Ascend GPU CPU
<code>mindspore.numpy.digitize</code>	返回输入数组中每个值所属的桶的索引。	Ascend GPU CPU
<code>mindspore.numpy.divide</code>	返回输入的真除法, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.divmod</code>	同时返回逐元素的商和余数。	Ascend GPU CPU
<code>mindspore.numpy.dot</code>	返回两个数组的点积。	Ascend GPU CPU
<code>mindspore.numpy.ediff1d</code>	计算 <code>Tensor</code> 中连续元素之间的差值。	Ascend GPU CPU
<code>mindspore.numpy.exp</code>	计算输入数组中所有元素的指数。	Ascend GPU CPU
<code>mindspore.numpy.exp2</code>	计算输入数组中所有值 <code>p</code> 的 2^{**p} 。	Ascend GPU CPU
<code>mindspore.numpy.expm1</code>	计算数组中所有元素的 $\exp(x) - 1$ 。	Ascend GPU CPU
<code>mindspore.numpy.fix</code>	舍入至最接近零的相邻整数。	Ascend GPU CPU
<code>mindspore.numpy.float_power</code>	第一个数组逐元素计算幂次方, 指数为第二个数组中对应的元素。	Ascend GPU CPU
<code>mindspore.numpy.floor</code>	返回输入的向下取整, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.floor_divide</code>	对输入进行除法运算, 返回小于等于除法结果的最大整数。	Ascend GPU CPU
<code>mindspore.numpy.fmod</code>	返回除法的逐元素余数。	Ascend GPU CPU
<code>mindspore.numpy.gcd</code>	返回 $ x1 $ 和 $ x2 $ 的最大公约数。	Ascend GPU CPU
<code>mindspore.numpy.gradient</code>	返回一个 <code>N</code> 维数组的梯度。	Ascend GPU CPU
<code>mindspore.numpy.heaviside</code>	计算 <code>Heaviside</code> 阶跃函数。	Ascend GPU CPU
<code>mindspore.numpy.histogram</code>	计算数据集的直方图。	Ascend GPU CPU
<code>mindspore.numpy.histogram2d</code>	计算数据的二维直方图。	Ascend GPU CPU
<code>mindspore.numpy.histogramdd</code>	计算数据的多维直方图。	Ascend GPU CPU
<code>mindspore.numpy.hypot</code>	给定直角三角形的直角边, 返回其斜边。	Ascend GPU CPU
<code>mindspore.numpy.inner</code>	返回两个 <code>Tensor</code> 的内积。	Ascend GPU CPU
<code>mindspore.numpy.interp</code>	用于单调递增的样本点的一维线性插值。	Ascend GPU CPU
<code>mindspore.numpy.invert</code>	逐元素计算按位取反或按位非。	Ascend
<code>mindspore.numpy.kron</code>	两个数组的 <code>Kronecker</code> 积。	Ascend GPU CPU
<code>mindspore.numpy.lcm</code>	返回 $ x1 $ 和 $ x2 $ 的最小公倍数。	Ascend GPU CPU
<code>mindspore.numpy.log</code>	返回自然对数, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.log10</code>	<code>x</code> 的以 10 为底的对数。	Ascend GPU CPU

下页继续

表 4 - 续上页

<code>mindspore.numpy.log1p</code>	返回 1 加上输入数组的自然对数, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.log2</code>	x 的以 2 为底的对数。	Ascend GPU CPU
<code>mindspore.numpy.logaddexp</code>	计算输入指数取幂的和的对数。	Ascend GPU CPU
<code>mindspore.numpy.logaddexp2</code>	计算输入指数取幂的和的以 2 为底对数。	Ascend GPU CPU
<code>mindspore.numpy.matmul</code>	返回两个数组的矩阵乘积。	Ascend GPU CPU
<code>mindspore.numpy.matrix_power</code>	计算方阵以整数 n 为指数的幂。	Ascend GPU CPU
<code>mindspore.numpy.maximum</code>	逐元素比较两个数组, 返回每对数组元素中的最大值。	Ascend GPU CPU
<code>mindspore.numpy.mean</code>	沿指定轴计算算术平均值。	Ascend GPU CPU
<code>mindspore.numpy.minimum</code>	逐元素比较两个数组, 返回每对数组元素中的最小值。	Ascend GPU CPU
<code>mindspore.numpy.multi_dot</code>	计算两个或更多个数组的点积, 同时自动选择最快的计算顺序。	Ascend GPU CPU
<code>mindspore.numpy.multiply</code>	参数逐元素相乘。	Ascend GPU CPU
<code>mindspore.numpy.nancumsum</code>	返回给定轴上数组元素的累积和, 将 NaN(非数值) 视为零。	GPU CPU
<code>mindspore.numpy.nanmax</code>	返回数组的最大值或沿某个轴的最大值, 忽略 NaN。	GPU CPU
<code>mindspore.numpy.nanmean</code>	沿指定轴计算算术平均值, 忽略 NaN。	GPU CPU
<code>mindspore.numpy.nanmin</code>	返回数组的最大值或沿某个轴的最大值, 忽略 NaN。	GPU CPU
<code>mindspore.numpy.nanstd</code>	计算指定轴上元素的标准差, 忽略 NaN。	GPU CPU
<code>mindspore.numpy.nansum</code>	计算指定轴上元素的总和, 将 NaN(非数值) 视为零。	GPU CPU
<code>mindspore.numpy.nanvar</code>	计算指定轴上的方差, 忽略 NaN。	GPU CPU
<code>mindspore.numpy.negative</code>	数值符号取反, 逐元素操作。	Ascend GPU CPU
<code>mindspore.numpy.norm</code>	矩阵或向量的范数。	Ascend GPU CPU
<code>mindspore.numpy.outer</code>	计算两个向量的外积。	Ascend GPU CPU
<code>mindspore.numpy.polyadd</code>	找到两个多项式的和。	Ascend GPU CPU
<code>mindspore.numpy.polyder</code>	返回多项式指定阶数的导数。	Ascend GPU CPU
<code>mindspore.numpy.polyint</code>	返回多项式的一个反导数 (不定积分)。	Ascend GPU CPU
<code>mindspore.numpy.polymul</code>	求两个多项式的积。	GPU
<code>mindspore.numpy.polysub</code>	两个多项式的差 (减法)。	Ascend GPU CPU
<code>mindspore.numpy.polyval</code>	在特定值处求多项式的值。	Ascend GPU CPU
<code>mindspore.numpy.positive</code>	数值取正, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.power</code>	第一个数组的元素以第二个数组的元素为指数逐元素求幂。	Ascend GPU CPU
<code>mindspore.numpy.promote_types</code>	返回 <code>type1</code> 与 <code>type2</code> 都可以安全转换的最小位数和最小标量类型的数据类型。	Ascend GPU CPU
<code>mindspore.numpy.ptp</code>	沿某个轴的值范围 (最大值 - 最小值)。	Ascend GPU CPU
<code>mindspore.numpy.rad2deg</code>	将角从弧度制转换为角度制。	Ascend GPU CPU
<code>mindspore.numpy.radians</code>	将角从角度制转换为弧度制。	Ascend GPU CPU
<code>mindspore.numpy.ravel_multi_index</code>	将元素为索引数组的 <code>tuple</code> 转换为展平的索引数组, 并对多重索引应用边界模式。	GPU
<code>mindspore.numpy.reciprocal</code>	逐元素返回入参的倒数。	Ascend GPU CPU
<code>mindspore.numpy.remainder</code>	逐元素返回除法余数。	Ascend GPU CPU
<code>mindspore.numpy.result_type</code>	返回对入参使用类型提升规则所得的类型。	Ascend GPU CPU
<code>mindspore.numpy rint</code>	将数组的元素四舍五入到最接近的整数。	Ascend GPU CPU

下页继续

表 4 - 续上页

<code>mindspore.numpy.searchsorted</code>	找到每个元素的插入索引, 使得插入后的数组保持原有升降序。	Ascend GPU CPU
<code>mindspore.numpy.sign</code>	逐元素返回数的符号。	Ascend GPU CPU
<code>mindspore.numpy.sin</code>	逐元素计算三角正弦函数。	Ascend GPU CPU
<code>mindspore.numpy.sinh</code>	逐元素计算双曲正弦函数。	Ascend CPU
<code>mindspore.numpy.sqrt</code>	逐元素返回数组的非负平方根。	Ascend GPU CPU
<code>mindspore.numpy.square</code>	逐元素返回输入的平方。	Ascend GPU CPU
<code>mindspore.numpy.std</code>	沿指定轴计算标准差。	Ascend GPU CPU
<code>mindspore.numpy.subtract</code>	逐元素减去给定参数。	Ascend GPU CPU
<code>mindspore.numpy.sum</code>	返回指定轴上数组元素的总和。	Ascend GPU CPU
<code>mindspore.numpy.tan</code>	逐元素计算正切值。	Ascend CPU
<code>mindspore.numpy.tanh</code>	逐元素计算双曲正切值。	Ascend GPU CPU
<code>mindspore.numpy.tensordot</code>	沿指定轴计算 Tensor 的点积。	Ascend GPU CPU
<code>mindspore.numpy.trapz</code>	使用复合梯形规则沿给定轴进行积分。	Ascend GPU CPU
<code>mindspore.numpy.true_divide</code>	返回输入的真除法, 逐元素计算。	Ascend GPU CPU
<code>mindspore.numpy.trunc</code>	逐元素返回输入的截断值。	Ascend GPU CPU
<code>mindspore.numpy.unwrap</code>	通过加或减 2π , 改变数组相邻元素的差值实现解卷绕。	Ascend GPU CPU
<code>mindspore.numpy.var</code>	计算沿指定轴的方差。	Ascend GPU CPU

15.4.1 mindspore.numpy.absolute

`mindspore.numpy.absolute(x, dtype=None)`

逐元素计算绝对值。

说明: 不支持 **Numpy** 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。目前后端内核仅支持浮点数计算, 如果输入不是 `float`, 则会被转换为 `mstype.float32` 类型并再转换回来。

参数:

- **x** (**Tensor**) - 用于计算的 **Tensor**。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。用于覆盖输出 **Tensor** 的 `dtype`。

返回:

Tensor。

异常:

- **TypeError** - 如果输入参数非上述给定的类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.asarray([1, 2, 3, -4, -5], np.float32)
>>> output = np.absolute(x)
>>> print(output)
[1. 2. 3. 4. 5.]
```

15.4.2 mindspore.numpy.add

`mindspore.numpy.add(x1, x2, dtype=None)`
逐元素相加两个参数。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x1** (Tensor) - 参与相加的第一个输入。
- **x2** (Tensor) - 参与相加的第二个输入。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。用于覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, `x1` 和 `x2` 逐元素相加后的和。如果 `x1` 和 `x2` 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x1 = np.full((3, 2), [1, 2])
>>> x2 = np.full((3, 2), [3, 4])
>>> output = np.add(x1, x2)
>>> print(output)
[[4 6]
 [4 6]
 [4 6]]
```

15.4.3 mindspore.numpy.amax

`mindspore.numpy.amax(a, axis=None, keepdims=False, initial=None, where=True)`
返回数组的最大值或沿指定轴的最大值。

说明: 不支持 Numpy 的 `out` 参数。在 GPU 上, 支持的数据类型为 `np.float16` 和 `np.float32`。

参数:

- **a** (Tensor) - 输入数据。
- **axis** (Union[int, tuple(int), None], 可选) - 默认值: `None`。指定操作的轴或多个轴。默认情况下, 将使用展平后的输入。如果该参数是整数组成的 `tuple`, 则会在多个轴上选择最大值, 而不是像之前那样选择单个轴或所有轴上的最大值。
- **keepdims** (boolean, 可选) - 默认值: `False`。如果设置为 `True`, 则保留被缩减的轴, 作为结果中大小为一的维度。使用此选项, 结果将与输入数组正确广播。
- **initial** (scalar, 可选) - 默认值: `None`。输出元素的最小值。必须存在才能在空切片上进行计算。
- **where** (boolean Tensor, 可选) - 默认值: `True`。一个布尔数组, 被广播以匹配数组的维度, 并选择包含在计算中的元素。如果传递了非默认值, 则 `initial` 也必须提供。

返回：

Tensor 或标量，a 的最大值。如果 axis 为 None，则结果是一个标量值。如果给定 axis，则结果是一个维度为 `a.ndim - 1` 的数组。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.arange(4).reshape((2,2)).astype('float32')
>>> output = np.amax(a)
>>> print(output)
3.0
>>> output = np.amax(a, axis=0)
>>> print(output)
[2. 3.]
>>> output = np.amax(a, axis=1)
>>> print(output)
[1. 3.]
>>> output = np.amax(a, where=np.array([False, True]), initial=-1, axis=0)
>>> print(output)
[-1.  3.]
```

15.4.4 mindspore.numpy.amin

`mindspore.numpy.amin(a, axis=None, keepdims=False, initial=None, where=True)`

返回数组的最小值或沿指定轴的最小值。

说明：不支持 Numpy 的 `out` 参数。在 GPU 上，支持的数据类型为 `np.float16` 和 `np.float32`。

参数：

- **a** (Tensor) - 输入数据。
- **axis** (Union[int, tuple(int), None], 可选) - 默认值：None。指定操作的轴或多个轴。默认情况下，将使用展平后的输入。如果该参数是整数组成的 tuple，则会在多个轴上选择最小值，而不是像之前那样选择单个轴或所有轴上的最小值。
- **keepdims** (bool, 可选) - 默认值：False。如果设置为 True，则保留被缩减的轴，作为结果中大小为一的维度。使用此选项，结果将与输入数组正确广播。
- **initial** (Number, 可选) - 默认值：None。输出元素的最大值。必须存在才能在空切片上进行计算。
- **where** (bool Tensor, 可选) - 默认值：True。一个布尔数组，被广播以匹配数组的维度，并选择包含在计算中的元素。如果传递了非默认值，则 `initial` 也必须提供。

返回：

Tensor 或标量，a 的最小值。如果 axis 为 None，则结果是一个标量值。如果给定 axis，则结果是一个维度为 `a.ndim - 1` 的数组。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.arange(4).reshape((2,2)).astype('float32')
>>> output = np.amin(a)
>>> print(output)
0.0
>>> output = np.amin(a, axis=0)
>>> print(output)
[0. 1.]
>>> output = np.amin(a, axis=1)
>>> print(output)
[0. 2.]
>>> output = np.amin(a, where=np.array([False, True]), initial=10, axis=0)
>>> print(output)
[10. 1.]
```

15.4.5 mindspore.numpy.arccos

`mindspore.numpy.arccos(input, dtype=None)`

逐元素计算反余弦函数。

说明：不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数：

- **input** (Tensor) - 输入 Tensor，表示单位圆上的 x 坐标。对于实数输入，定义域为 $[-1, 1]$ 。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input = np.asarray([1, -1], np.float32)
>>> output = np.arccos(input)
>>> print(output)
[0.          3.1415927]
```

15.4.6 mindspore.numpy.arccosh

`mindspore.numpy.arccosh(x, dtype=None)`
逐元素计算反双曲余弦函数。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量。如果 `x` 是标量, 则返回结果为标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.arange(1, 5).astype('float32')
>>> print(np.arccosh(x))
[0.          1.316958  1.7627472 2.063437 ]
```

15.4.7 mindspore.numpy.arcsin

`mindspore.numpy.arcsin(x, dtype=None)`
逐元素计算反正弦函数。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x** (Tensor) - 输入 Tensor, 表示单位圆上的 `y` 坐标。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

输出 Tensor。

异常:

- **TypeError** - 如果输入不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.asarray([1, -1], np.float32)
>>> output = np.arcsin(x)
>>> print(output)
[ 1.5707964 -1.5707964]
```

15.4.8 mindspore.numpy.arcsinh

`mindspore.numpy.arcsinh(x, dtype=None)`

逐元素计算反双曲正弦函数。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量。如果 `x` 是标量, 则返回结果为标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([1., 2., 3., 4.], dtype=np.float32)
>>> print(np.arcsinh(x))
[0.8813736 1.4436355 1.8184465 2.0947125]
```

15.4.9 mindspore.numpy.arctan

`mindspore.numpy.arctan(x, dtype=None)`

逐元素计算反正切函数。这是正切函数的反函数, 如果 $y = \tan(x)$, 则 $x = \arctan(y)$ 。

说明: 不支持 Numpy 的 `out`、`where`、`casting`、`order`、`subok`、`signature` 和 `extobj` 参数。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量。如果 `x` 是标量, 则返回结果为标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.arange(5).astype('float32')
>>> print(np.arctan(x))
[0.          0.7853982  1.1071488  1.2490457  1.3258177]
```

15.4.10 mindspore.numpy.arctan2

`mindspore.numpy.arctan2(x1, x2, dtype=None)`
逐元素计算 $x1/x2$ 的反正切, 并正确选择象限。

说明: 不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数:

- **x1** (Tensor) - 输入 Tensor。
- **x2** (Tensor) - 输入 Tensor。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量, $x1$ 与 $x2$ 逐元素的计算结果。如果 $x1$ 与 $x2$ 都是标量, 则结果也是标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x1 = np.array([-1, +1, +1, -1])
>>> x2 = np.array([-1, -1, +1, +1])
>>> output = np.arctan2(x1, x2)
>>> print(output)
[-2.3561945   2.3561945   0.78539819 -0.78539819]
```

15.4.11 mindspore.numpy.arctanh

`mindspore.numpy.arctanh(x, dtype=None)`
逐元素计算反双曲正切。

说明: 不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量。如果 x 是标量, 则结果也是标量。

支持平台:

Ascend CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([-0.99, -0.75, -0.5, 0, 0.5]).astype('float32')
>>> print(np.arctanh(x))
[-2.646653   -0.97295505 -0.54930615  0.          0.54930615]
```

15.4.12 mindspore.numpy.argmax

`mindspore.numpy.argmax(a, axis=None)`

返回沿指定轴最大值的索引。

说明: 不支持 NumPy 参数 `out`。在 Ascend 上, 如果存在多个最大值的情况, 返回的索引可能不一定对应于第一次出现的值。

参数:

- **a** (Union[int, float, bool, list, tuple, Tensor]) - 输入数组。
- **axis** (int, 可选) - 默认情况下, 索引进入展平的数组, 否则沿指定 *axis*。默认值: None。

返回:

Tensor, 原数组中元素的索引的数组。与移除指定 *axis* 后的入参 *a* 的 `shape` 相同。

异常:

- **ValueError** - 如果 *axis* 超出范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.arange(10, 16).reshape(2, 3)
>>> print(np.argmax(a))
5
>>> print(np.argmax(a, axis=0))
[1 1 1]
>>> print(np.argmax(a, axis=1))
[2 2]
```


15.4.13 mindspore.numpy.argmaxin

`mindspore.numpy.argmaxin(a, axis=None)`
 返回沿指定轴最小值的索引。

说明: 不支持 NumPy 参数 `out`。在 Ascend 上, 如果存在多个最小值的情况, 返回的索引可能不一定对应于第一次出现的值。

参数:

- **a** (Union[int, float, bool, list, tuple, Tensor]) - 输入数组。
- **axis** (int, 可选) - 默认情况下, 索引进入展平的数组, 否则沿指定 `axis`。默认值: `None`。

返回:

Tensor, 原数组中元素的索引的数组。与移除指定 `axis` 后的入参 `a` 的 `shape` 相同。

异常:

- **ValueError** - 如果 `axis` 超出范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.arange(10, 16).reshape(2, 3)
>>> print(np.argmaxin(a))
0
>>> print(np.argmaxin(a, axis=0))
[0 0 0]
>>> print(np.argmaxin(a, axis=1))
[0 0]
```

15.4.14 mindspore.numpy.around

`mindspore.numpy.around(a, decimals=0)`
 向给定的小数位数四舍五入。

说明: 不支持 NumPy 参数 `out`。不支持复数。

参数:

- **a** (Union[int, float, list, tuple, Tensor]) - 输入数据。
- **decimals** (int) - 需要四舍五入到的小数位数。默认值: `0`。

返回:

Tensor。一个与 `a` 同类型的 Tensor, 包含四舍五入后的数值。`float` 类型的数值四舍五入后的结果仍是 `float` 类型。

异常:

- **TypeError** - 如果输入不能被转化为一个 Tensor 或参数 `decimals` 不是整数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([-1.3, 0.0, 0.5, 1.5, 2.5])
>>> print(np.around(a))
[-1. 0. 0. 2. 2.]
```

15.4.15 mindspore.numpy.average

`mindspore.numpy.average(x, axis=None, weights=None, returned=False)`

沿指定轴计算加权平均值。

参数：

- **x** (Tensor) - 需要平均的 Tensor。
- **axis** (Union[None, int, tuple(int)], 可选) - 沿 *axis* 对 *x* 进行平均。默认值：None。如果 *axis* 为 None，它将对张量 *x* 的所有元素进行平均。如果 *axis* 是负数，它将从最后一个 *axis* 数回到第一个 *axis*。
- **weights** (Union[None, Tensor], 可选) - *weights* 与 *x* 中的值相关联。默认值：None。如果 *weights* 为 None，所有 *x* 中的数据权重假设为 1。如果 *weights* 是一个 1-D 的 Tensor，其长度必须与给定的 *axis* 的长度相等。否则，*weights* 应该与 *x* 具有相同的 shape。
- **returned** (bool, 可选) - 默认值：False。如果为 True，函数将返回 tuple (average, sum_of_weights)。如果为 False，只返回平均值。

返回：

平均后的 Tensor。如果 *returned* 为 True，返回 tuple。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input_x = np.array([[1., 2.], [3., 4.]])
>>> output = np.average(input_x, axis=0, weights=input_x, returned=True)
>>> print(output)
(Tensor(shape=[2], dtype=Float32, value= [ 2.50000000e+00,  3.33333325e+00]),
 Tensor(shape=[2], dtype=Float32, value= [ 4.00000000e+00,  6.00000000e+00]))
```

15.4.16 mindspore.numpy.bincount

`mindspore.numpy.bincount(x, weights=None, minlength=0, length=None)`

计算数组中非负 int 值的出现次数。结果返回一个桶 (大小为 1) 数组，桶数组长度比 *x* 中的最大值大 1。如果指定了 *minlength*，输出数组至少会有 *minlength* 个桶 (如果指定了该参数，根据 *x* 的长度，可能会让输出数组更长)。每个桶记录了其索引值在 *x* 中出现的次数。如果指定了 *weights*，则输入数组会根据权重加权，即如果在位置 *i* 处找到了值 *n*，则以 `out[n] += weight` 代替 `out[n] += 1`。

说明：附加的参数 *length* 定义了桶的数量 (覆盖 $x.\max() + 1$)，在 *graph* 模式下必须提供该参数。如果 *x* 包含了负值，不会引发错误，负值将被视为 0。

参数：

- **x** (Union[list, tuple, Tensor]) - 1-d 输入数组。
- **weights** (Union[int, float, bool, list, tuple, Tensor], 可选) - 权重，与 *x* 的 shape 相同的数组。默认值：0。
- **minlength** (int, 可选) - 输出数组的最小桶数。默认值：0。
- **length** (int, 可选) - 桶的数量。默认值：None。

返回：

Tensor，即输入数组分桶后的结果。输出的长度等于 $\text{np.amax}(x) + 1$ 。

异常：

- **ValueError** - 如果 *x* 不是一维的，或 *x* 和 *weights* 的 shape 不同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.bincount(np.arange(5)))
[1. 1. 1. 1. 1.]
>>> print(np.bincount(np.array([0, 1, 1, 3, 2, 1, 7])))
[1. 3. 1. 1. 0. 0. 0. 1.]
>>> w = np.array([0.3, 0.5, 0.2, 0.7, 1., -0.6]) # weights
>>> x = np.array([0, 1, 1, 2, 2, 2])
>>> print(np.bincount(x, weights=w))
[0.3 0.7 1.1]
```

15.4.17 mindspore.numpy.bitwise_and

`mindspore.numpy.bitwise_and(x1, x2, dtype=None)`

逐元素计算两个数组的按位与运算。计算输入数组中整数的二进制表示的按位与。此函数实现了 C/Python 中的操作符 `&`。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。只处理 int 和 bool 类型。如果 $x1.shape \neq x2.shape$ ，它们必须能广播到一个共同的 shape (即输出的 shape)。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。若 *x* 是标量，则返回值也是标量。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.bitwise_and(13, 17))
1
```

15.4.18 mindspore.numpy.bitwise_or

`mindspore.numpy.bitwise_or(x1, x2, dtype=None)`

逐元素计算两个数组的按位或运算。计算输入数组中整数的二进制表示的按位或。此函数实现了 C/Python 中的操作符 `|`。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。只处理 `int` 和 `bool` 类型。如果 `x1.shape != x2.shape`，它们必须能广播到一个共同的 `shape`(即输出的 `shape`)。
- **dtype** (`mindspore.dtype`, 可选) - 默认值：`None`。覆盖输出 Tensor 的 `dtype`。

返回：

Tensor 或标量。若 `x` 是标量，则返回值也是标量。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.bitwise_or(13, 16))
29
```

15.4.19 mindspore.numpy.bitwise_xor

`mindspore.numpy.bitwise_xor(x1, x2, dtype=None)`

逐元素计算两个数组的按位异或运算。计算输入数组中整数的二进制表示的按位异或。此函数实现了 C/Python 中的操作符 `^`。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **x1** (Tensor) - 输入数组。

- **x2** (Tensor) - 输入数组。只处理 `int` 和 `bool` 类型。如果 `x1.shape != x2.shape`，它们必须能广播到一个共同的 `shape`(即输出的 `shape`)。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量。若 `x` 是标量，则返回值也是标量。

支持平台:

Ascend CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.bitwise_xor(13, 17))
28
```

15.4.20 mindspore.numpy.cbrt

`mindspore.numpy.cbrt` (`x`, `dtype=None`)

返回 Tensor 的立方根，逐元素计算。

说明: 不支持 NumPy 参数 `casting`、`order`、`subok`、`signature`、`extobj`。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.asarray([1, -1, 3, -8, 64])
>>> output = np.cbrt(a)
>>> print(output)
[ 1.          -1.          1.4422495 -2.          4.          ]
```

15.4.21 mindspore.numpy.ceil

`mindspore.numpy.ceil(x, dtype=None)`

返回输入的向上取整结果，逐元素计算。标量 x 的向上取整得到最小的整数 i ，满足 $i \geq x$ 。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。在 GPU 上，支持的类型有 `np.float16` 和 `np.float32`。

参数：

- **x** (Tensor) - 输入值。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量， x 中每个元素的向上取整结果。如果 x 是标量，则结果也是标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
>>> output = np.ceil(a)
>>> print(output)
[-1. -1. -0.  1.  2.  2.  2.]
```

15.4.22 mindspore.numpy.clip

`mindspore.numpy.clip(x, xmin, xmax, dtype=None)`

裁剪(限制)数组的值。给定一个区间，区间外的值会被限制到区间边界。例如，如果给定区间 $[0, 1]$ ，则小于 0 的值变为 0，大于 1 的值变为 1。

参数：

- **x** (Tensor) - 包含需要限制的元素的 Tensor。
- **xmin** (Tensor, scalar, None) - 最小值。如果为 None，则不在下边界进行限制。`xmin` 和 `xmax` 不可以同时为 None。
- **xmax** (Tensor, scalar, None) - 最大值。如果为 None，则不在上边界进行限制。`xmin` 和 `xmax` 不可以同时为 None。如果 `xmin` 或 `xmax` 是 Tensor，则这三个 Tensor 将会广播以匹配它们的 shape。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor，一个包含 x 的元素的 Tensor，但小于 `xmin` 的值替换为 `xmin`，大于 `xmax` 的值替换为 `xmax`。

异常：

- **TypeError** - 如果输入的类型不符合上述规定。
- **ValueError** - 如果 `x1` 和 `x2` 的 shape 无法被广播，或者 `xmin` 和 `xmax` 同时为 None。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.asarray([1, 2, 3, -4, 0, 3, 2, 0])
>>> output = np.clip(x, 0, 2)
>>> print(output)
[1 2 2 0 0 2 2 0]
```

15.4.23 mindspore.numpy.convolve

`mindspore.numpy.convolve(a, v, mode='full')`

返回两个一维序列的离散线性卷积。

说明: 如果 v 比 a 长, 计算前会交换这两个 Tensor。

参数:

- **a** (Union[list, tuple, Tensor]) - 第一个输入的一维 Tensor。
- **v** (Union[list, tuple, Tensor]) - 第二个输入的一维 Tensor。
- **mode** (str, 可选) - 默认为 'full', 这此 *mode* 下, 卷积在每个重叠点计算, 输出的 shape 为 $(N + M - 1)$ 。在卷积的端点, 信号可能不完全重叠, 可能会看到边界效应。如果 *mode* 为 'same', 则返回值的长度为 $\max(M, N)$ 。依旧可以观察到边界效应。如果 *mode* 为 'valid', 则返回值的长度为 $\max(M, N) - \min(M, N) + 1$ 。卷积仅在信号完全重叠的点给出。信号边界外的值无效。

返回:

Tensor, a 和 v 的离散线性卷积。

异常:

- **TypeError** - 如果输入的类型不符合上述规定。
- **ValueError** - 如果 a 和 v 为空或维数错误。

支持平台:

GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.convolve([1., 2., 3., 4., 5.], [2., 3.], mode="valid")
>>> print(output)
[ 7. 12. 17. 22.]
```

15.4.24 mindspore.numpy.copysign

`mindspore.numpy.copysign(x1, x2, dtype=None)`

将 $x1$ 的符号更改为 $x2$ 的符号，逐元素执行。如果 $x2$ 是一个标量，则它的符号会被复制到 $x1$ 的所有元素。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。不支持复数输入。

参数：

- **x1** (Union[int, float, list, tuple, Tensor]) - 需要更改符号的值。
- **x2** (Union[int, float, list, tuple, Tensor]) - $x1$ 的符号将更改为 $x2$ 的符号。如果 $x1.shape \neq x2.shape$ ，它们必须能广播到一个共同的 shape(即输出的 shape)。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。更改为与 $x2$ 相同符号的 $x1$ 的值。如果 $x1$ 与 $x2$ 都是标量，则结果也是标量。

异常：

- **TypeError** - 如果输入的 dtype 不在规定的类型内或输入不能被转换为 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.copysign(np.array([1, -1, -1]), np.array([-1, 1, -1]))
>>> print(output)
[-1  1 -1]
```

15.4.25 mindspore.numpy.corrcoef

`mindspore.numpy.corrcoef(x, y=None, rowvar=True, dtype=None)`

返回皮尔逊积矩相关系数。请参考有关协方差 (cov) 的文档以获取更多详情。相关系数矩阵 R 和协方差矩阵 C 的关系表示为 $R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} * C_{jj}}}$ 。R 的值介于 -1 和 1 之间，包括 -1 和 1。

说明：目前不支持复数。

参数：

- **x** (Union[int, float, bool, tuple, list, Tensor]) - 这是一个 1-D 或 2-D 的数组，包含多个变量和观测值。每一行的 x 代表一个变量，每一列代表所有这些变量的一个观测值。详细见下文 `rowvar` 的说明。
- **y** (Union[int, float, bool, tuple, list, Tensor], 可选) - 这是一个附加的变量和观测值集合，默认值：None。
- **rowvar** (bool, 可选) - 如果 `rowvar` 是 `True` (默认)，则每一行代表一个变量，列中包含观测值。否则，关系被转置：每一列代表一个变量，行包含观测值。默认值：`True`。
- **dtype** (mindspore.dtype, 可选) - 结果的数据类型。默认情况下，返回的数据类型至少为 float32 精度。默认值：None。

返回：

Tensor。变量的相关系数矩阵。

异常：

- **TypeError** - 如果输入的类型不符合上述规定。
- **ValueError** - 如果 x 和 y 的维数不正确。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.corrcoef([[2., 3., 4., 5.], [0., 2., 3., 4.], [7., 8., 9., 10.]])
>>> print(output)
[[1.          0.9827076  1.          ]
 [0.9827077  0.99999994 0.9827077 ]
 [1.          0.9827076  1.          ]]
```

15.4.26 mindspore.numpy.correlate

`mindspore.numpy.correlate(a, v, mode='valid')`

两个一维序列的互相关。

函数根据信号处理教材中通常定义的方式计算相关性： $c_{av}[k] = \sum_n a[n+k] * conj(v[n])$ 。

其中，序列 a 和 v 在必要时将进行零填充， $conj$ 为共轭。

说明：

- *correlate* 目前仅用于 *mindscience* 科学计算场景，并不支持其他使用场景。
- *correlate* 还未在 Windows 平台上得到支持。

参数：

- **a** (Union[list, tuple, Tensor]) - 第一个输入序列。
- **v** (Union[list, tuple, Tensor]) - 第二个输入序列。
- **mode** (str, 可选) - 指定的填充模式。可选值：“same”、“valid”、“full”。默认值：“valid”。
 - “same”：返回的输出长度为 $\max(M, N)$ 。仍然可见边界效应。
 - “valid”：返回的输出长度为 $\max(M, N) - \min(M, N) - 1$ 。卷积乘积仅在信号完全重叠的点给出。信号边界外的值不产生效果。
 - “full”：在每个重叠点返回卷积，输出的 shape 为 $(N + M - 1)$ 。在卷积的端点，信号不完全重叠，可能会看到边界效应。

返回：

Tensor， a 和 v 的离散互相关。

异常：

- **TypeError** - 如果 a 或 v 不是 Tensor。

- **TypeError** - 如果 a 和 v 的类型不同。
- **ValueError** - 如果 a 和 v 为空或维数错误。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as mnp
>>> from mindspore import Tensor
>>> output = mnp.correlate(Tensor([1., 2., 3.]), Tensor([0., 1., 0.5]))
>>> print(output)
Tensor(shape=[1], dtype=Float32, value= [ 3.50000000e+00])
>>> output = mnp.correlate(Tensor([1., 2., 3.]), Tensor([0., 1., 0.5]), mode="same")
>>> print(output)
Tensor(shape=[3], dtype=Float32, value= [ 2.00000000e+00,  3.50000000e+00,  3.00000000e+00])
>>> output = mnp.correlate(Tensor([1., 2., 3.]), Tensor([1., 2.]), mode="full")
>>> print(output)
Tensor(shape=[4], dtype=Float32, value= [ 2.00000000e+00,  5.00000000e+00,  8.00000000e+00,  3.00000000e+00])
```

15.4.27 mindspore.numpy.cos

`mindspore.numpy.cos(x, dtype=None)`

逐元素计算余弦。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。如果 x 是标量，则结果也是标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.arange(5).astype('float32')
>>> print(np.cos(x))
[ 1.          0.5403023 -0.41614684 -0.9899925 -0.6536436 ]
```

15.4.28 mindspore.numpy.cosh

`mindspore.numpy.cosh(x, dtype=None)`
逐元素计算双曲余弦。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。如果 *x* 是标量，则结果也是标量。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.arange(5).astype('float32')
>>> print(np.cosh(x))
[ 1.          1.5430807  3.7621956 10.067662 27.308233 ]
```

15.4.29 mindspore.numpy.count_nonzero

`mindspore.numpy.count_nonzero(x, axis=None, keepdims=False)`
计算 Tensor *x* 中的非零值数量。

参数：

- **x** (Tensor) - 需要统计非零值数量的 Tensor。
- **axis** (Union[int,tuple], 可选) - 指定沿着哪些轴统计非零元素个数，可以是单个轴，也可以用 tuple 表示多个轴。默认值为 None，此时沿完全展平的 *x* 计算非零值数量。默认值：None。
- **keepdims** (bool, 可选) - 如果设置为 True，结果会保留计数所沿的 *axis*，该维度的大小为 1。若使用此选项，结果会广播到和 *x* 同一个维度数。默认值：False。

返回：

Tensor，表示 *x* 沿给定 *axis* 的非零值数量。否则，返回 *x* 中非零值的总数。

异常：

- **TypeError** - 如果 *axis* 不是 int 或 tuple。
- **ValueError** - 如果 *axis* 不在范围 $[-x.ndim, x.ndim)$ 内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.asarray([1, 2, 3, -4, 0, 3, 2, 0])
>>> output = np.count_nonzero(x)
>>> print(output)
6
```

15.4.30 mindspore.numpy.cov

`mindspore.numpy.cov(m, y=None, rowvar=True, bias=False, ddof=None, fweights=None, aweights=None, dtype=None)`

给定数据和权重，估算一个协方差矩阵。

协方差表示两个变量共同变化的程度。如果以 N 维样本为例， $X = [x_1, x_2, \dots, x_N]^T$ ，则协方差矩阵元素 C_{ij} 是 x_i 和 x_j 的协方差。元素 C_{ii} 是 x_i 的方差。

说明: *fweights* 和 *aweights* 必须都为正，在 NumPy 中，如果检测到负值，将引发 `ValueError`。在 MindSpore 中，所有值将转化为正值。

参数：

- **m** (Union[`Tensor`, list, tuple]) - 一个 1-D 或 2-D 的 `Tensor`，包含多个变量和观测值。*m* 的每一行代表一个变量，每一列代表所有变量的某一观测值。另请见下述 *rowvar*。
- **y** (Union[`Tensor`, list, tuple], 可选) - 一个附加的变量和观测值的集合。*y* 的形式与 *m* 相同，默认值: `None`。
- **rowvar** (bool, 可选) - 如果 *rowvar* 为 `True` (默认值)，则每行代表一个变量，每列代表一个观测值。否则，关系被转置：每列代表一个变量，每行代表一个观测值。
- **bias** (bool, 可选) - 默认的归一化 (`False`) 是通过 $N - 1$ 计算的，其中 N 是观测值的数量。如果 *bias* 为 `True`，则归一化通过 N 计算。这些值可通过关键词 *ddof* 被覆盖。
- **ddof** (int, 可选) - 如果不为 `None`，由 *bias* 设定的默认值将被覆盖。请注意，即使指定了 *fweights* 和 *aweights*，*ddof* = 1 也将返回无偏估计，*ddof* = 0 将返回简单平均值。默认值: `None`。
- **fweights** (Union[`Tensor`, list, tuple], 可选) - 一个 1-D 的整数频率权重 `Tensor`。表示每个观测向量应该重复的次数。默认值: `None`。
- **aweights** (Union[`Tensor`, list, tuple], 可选) - 一个 1-D 的观测向量权重 `Tensor`。被认为更重要的观测值的权重相对较大，被认为不那么重要的观测值的权重相对较小。如果 *ddof* = 0，权重 `Tensor` 可用于给观测向量分配概率。默认值: `None`。
- **dtype** (Union[mindspore.dtype, str], 可选) - 结果的数据类型。默认情况下，返回的数据类型将具有 `mstype.float32` 精度。默认值: `None`。

返回：

`Tensor`，变量的协方差矩阵。

异常：

- **TypeError** - 如果输入的类型不是上述指定类型。
- **ValueError** - 如果 *m* 和 *y* 的维数错误。
- **RuntimeError** - 如果 *aweights* 和 *fweights* 的维数大于 2。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.cov([[2., 3., 4., 5.], [0., 2., 3., 4.], [7., 8., 9., 10.]])
>>> print(output)
[[1.66666666 2.16666667 1.66666666]
 [2.16666667 2.91666667 2.16666667]
 [1.66666666 2.16666667 1.66666666]]
```

15.4.31 mindspore.numpy.cross

`mindspore.numpy.cross(a, b, axisa=-1, axisb=-1, axisc=-1, axis=None)`

返回两个向量 (数组) 的叉积。

在 R^3 中, 叉积是垂直于 a 和 b 的一个向量。如果 a 和 b 是向量数组, 向量默认由 a 和 b 的最后一个轴定义, 这些轴可以为二或三维。当 a 和 b 其中任意一个为二维时, 输入向量的第三个分量将被假设为零, 并相应计算叉积。在两个输入向量都为二维的情况下, 返回叉积的 z -分量。

参数:

- **a** (Union[list, tuple, Tensor]) - 第一个向量的分量。
- **b** (Union[list, tuple, Tensor]) - 第二个向量的分量。
- **axisa** (int, 可选) - 定义 a 中用于计算的向量的轴。默认为最后一个轴。
- **axisb** (int, 可选) - 定义 b 中用于计算的向量的轴。默认为最后一个轴。
- **axisc** (int, 可选) - 定义 c 中包含叉积结果的轴。如果两个输入向量都为二维, 此项被忽略。默认为最后一个轴。
- **axis** (int, 可选) - 如果定义该参数, $axis$ 将定义 a 和 b 中用于计算的向量的轴, 以及 c 中包含叉积结果的轴。即覆盖 $axisa$ 、 $axisb$ 和 $axisc$ 。默认值: `None`。

返回:

Tensor, 向量叉积。

异常:

- **ValueError** - 当向量的维数不是 2 或 3 时。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([[1, 2, 3], [4, 5, 6]])
>>> y = np.array([[4, 5, 6], [1, 2, 3]])
>>> output = np.cross(x, y)
>>> print(output)
[[-3  6 -3]
 [ 3 -6  3]]
>>> output = np.cross(x, y, axisc=0)
>>> print(output)
[[-3  3]
 [ 6 -6]
 [-3  3]]
```

15.4.32 mindspore.numpy.cumprod

`mindspore.numpy.cumprod(a, axis=None, dtype=None)`

返回沿给定 *axis* 的元素的累计乘积。

说明: 不支持 NumPy 参数 *out* 。

参数:

- **a** (Union[int, float, bool, list, tuple, Tensor]) - 输入 Tensor。
- **axis** (int, 可选) - 计算累积乘积所沿的轴。默认情况下输入会被展开。默认值: None。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor。

异常:

- **TypeError** - 如果输入不能转换为 Tensor 或者 *axis* 不是整数。
- **ValueError** - 如果 *axis* 超出范围。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([1, 2, 3])
>>> print(np.cumprod(x))
[1 2 6]
```

15.4.33 mindspore.numpy.cumsum

`mindspore.numpy.cumsum(a, axis=None, dtype=None)`

返回沿给定 *axis* 的元素的累计和。

说明: 如果 *a.dtype* 为 *int8*、*int16* 或 *bool*，结果的 *dtype* 将提升至 *int32*。

参数:

- **a** (Tensor) - 输入 Tensor。
- **axis** (int, 可选) - 计算累计和所沿的轴。默认为 *None*，计算被展开的数组的累计和。
- **dtype** (*mindspore.dtype*, 可选) - 如果未指定，保持与输入 Tensor *a* 相同，除非 *a* 的整数精度低于默认平台整数的精度。在这种情况下，使用默认平台整数类型。默认值: None。

返回:

Tensor。

异常:

- **TypeError** - 如果输入参数的类型不是上述指定类型。
- **ValueError** - 如果 *axis* 超出范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.cumsum(np.ones((3,3)), axis=0)
>>> print(output)
[[1. 1. 1.]
 [2. 2. 2.]
 [3. 3. 3.]]
```

15.4.34 mindspore.numpy.deg2rad

`mindspore.numpy.deg2rad(x, dtype=None)`

将角度从角度制转换为弧度制。

参数：

- **x** (Tensor) - 角度，单位为度。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor，弧度制下的对应角度。如果 *x* 是 Tensor 标量，则结果也是 Tensor 标量。

异常：

- **TypeError** - 如果 *x* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.asarray([1, 2, 3, -4, -5])
>>> output = np.deg2rad(x)
>>> print(output)
[ 0.01745329  0.03490658  0.05235988 -0.06981317 -0.08726647]
```

15.4.35 mindspore.numpy.diff

`mindspore.numpy.diff(a, n=1, axis=-1, prepend=None, append=None)`

计算沿给定 *axis* 的 *n* 阶离散差分。

一阶差分由 $out[i] = a[i+1] - a[i]$ 给出，沿给定 *axis*，通过迭代使用 *diff* 计算更高阶的差分。

说明：由于 MindSpore 不支持 zero-shaped 的 Tensor，如果遇到空 Tensor 将引发 ValueError。

参数：

- **a** (Tensor) - 输入 Tensor。
- **n** (int, 可选) - 差分阶数。如果为零，输出将按原样返回。默认值：1。
- **axis** (int, 可选) - 取差分的轴。默认为最后一个轴。默认值：-1。
- **prepend/append** (Tensor, 可选) - 在执行差分之前在 *axis* 的前/后向 *a* 填充的值。如果是标量值，会沿着 *axis* 方向填充，填充宽度为 1，在所有其他方向上与输入数组 *shape* 相同的数组。否则，除了指定的 *axis*，其余维度和 *shape* 必须与输入数组 *a* 匹配。默认值：None。

返回：

n 阶差分。输出的 *shape* 除了在指定的 *axis* 方向上的维数比 *a* 小 *n* 外，其他与 *a* 相同。输出的类型与 *a* 中任意两个元素之间差分的类型相同，在大多数情况下，与 *a* 的类型相同。

异常：

- **TypeError** - 如果输入的类型不符合上述指定类型。
- **ValueError** - 如果 *n* < 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> arr = np.array([1, 3, -1, 0, 4])
>>> print(np.diff(arr, n=2))
[-6  5  3]
```

15.4.36 mindspore.numpy.digitize

`mindspore.numpy.digitize(x, bins, right=False)`

返回输入数组中每个值所属的桶的索引。如果 *x* 中的值超出了桶的界限，将返回 0 或 `len(bins)` 作为相应的索引。

参数：

- **x** (Union[int, float, bool, list, tuple, Tensor]) - 待分桶的输入数组。
- **bins** (Union[list, tuple, Tensor]) - 桶的数组。必须是一维且单调的。
- **right** (boolean, 可选) - 表示区间是否包含右边界或左边界。默认为 (`right==False`)，表示区间不包括右边界。在这种情况下，左边界是开放的，即 `bins[i-1] <= x < bins[i]` 是单调递增桶的默认行为。

返回：

元素为 int 的 Tensor，输出元素为索引的数组，与 *x* 有相同的 *shape*。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.array([1.2, 10.0, 12.4, 15.5, 20.])
>>> bins = np.array([0, 5, 10, 15, 20])
>>> inds = np.digitize(x, bins)
```

(下页继续)

(续上页)

```
>>> print(inds)
[1 3 3 4 5]
```

15.4.37 mindspore.numpy.divide

`mindspore.numpy.divide(x1, x2, dtype=None)`

返回输入的真除法，逐元素计算。

该函数与 Python 默认的“向下取整除”不同，它返回真除法的结果。

说明：不支持 numpy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x1** (Tensor) - 被除数。
- **x2** (Tensor) - 除数。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或 Tensor，如果 *x1* 和 *x2* 都是 Tensor，返回 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.full((3, 2), [1, 2])
>>> x2 = np.full((3, 2), [3, 4])
>>> output = np.divide(x1, x2)
>>> print(output)
[[0.33333334 0.5      ]
 [0.33333334 0.5      ]
 [0.33333334 0.5      ]]
```

15.4.38 mindspore.numpy.divmod

`mindspore.numpy.divmod(x1, x2, dtype=None)`

同时返回逐元素的商和余数。

参数：

- **x1** (Union[Tensor]) - 被除 Tensor。
- **x2** (Union[Tensor, int, float, bool]) - 除数。如果 `x1.shape != x2.shape`，它们必须能被广播到一个共同的形状。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

逐元素返回商和向下取整除的余数，格式为 (商, 余数)

异常:

- **TypeError** - 如果 $x1$ 和 $x2$ 不是 Tensor 或 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.array([1, 2, 3, 4, 5])
>>> print(np.divmod(a, 1.5))
(Tensor(shape=[5], dtype=Float32,
value= [ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00,  2.00000000e+00,  3.
↪00000000e+00]),
Tensor(shape=[5], dtype=Float32,
value= [ 1.00000000e+00,  5.00000000e-01,  0.00000000e+00,  1.00000000e+00,  5.00000000e-
↪01]))
```

15.4.39 mindspore.numpy.dot

`mindspore.numpy.dot(a, b)`

返回两个数组的点积。

具体来说, 如果 a 和 b 都是 1-D 数组, 它是向量的内积 (没有复数共轭)。如果 a 和 b 都是 2-D 数组, 它是矩阵乘法。如果 a 或 b 是 0-D 的 (标量), 则等同于乘法。如果 a 为 N-D 数组且 b 为 1-D 数组, 它在 a 的最后一个轴上与 b 计算求和乘积。如果 a 是 N-D 数组且 b 是 M-D 数组 (其中 $M \geq 2$), 它在 a 的最后一个轴上与 b 的倒数第二个轴上计算求和乘积: `dot(a, b)[i, j, k, m] = sum(a[i, j, :] * b[k, :, m])`。

说明: 不支持 NumPy 的 `out` 参数。在 GPU 上, 支持的数据类型为 `np.float16`, `np.float32` 和 `np.float64`。在 CPU 上, 支持的数据类型为 `np.float16`, `np.float32` 和 `np.float64`。

参数:

- **a** (Tensor) - 输入 Tensor。
- **b** (Tensor) - 输入 Tensor。

返回:

Tensor 或标量, a 和 b 的点积。如果 a 和 b 都是标量或都是 1-D 数组, 则返回一个标量, 否则返回一个数组。

异常:

- **ValueError** - 如果 a 的最后一个维度的大小与 b 的倒数第二个维度的大小不同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.full((1, 3), 7).astype('float32')
>>> b = np.full((2, 3, 4), 5).astype('float32')
>>> output = np.dot(a, b)
>>> print(output)
```

(下页继续)

(续上页)

```
[[[105. 105. 105. 105.]
  [105. 105. 105. 105.]]]
```

15.4.40 mindspore.numpy.ediff1d

`mindspore.numpy.ediff1d(ary, to_end=None, to_begin=None)`

计算 Tensor 中连续元素之间的差值。

参数：

- **ary** (Tensor) - 如果需要，将在取差值之前被展平。
- **to_end** (Tensor, scalar, 可选) - 在返回的差值末尾添加的数字。默认值：None。
- **to_begin** (Tensor, scalar, 可选) - 在返回的差值开头添加的数字。默认值：None。

返回：

差值。

异常：

- **TypeError** - 如果输入类型不是上述指定的类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> arr = np.array([1, 3, -1, 0, 4])
>>> print(np.ediff1d(arr))
[ 2 -4  1  4]
```

15.4.41 mindspore.numpy.exp

`mindspore.numpy.exp(x, dtype=None)`

计算输入数组中所有元素的指数。

说明：不支持 NumPy 参数 *casting*、*order*、*subok*、*signature* 和 *extobj*。当使用 *where* 时，*out* 必须 Tensor 值。*out* 不支持用于存储结果，但可以与 *where* 一起使用来设置指定索引的值。在 GPU 上，支持的数据类型有 `np.float16` 和 `np.float32`。在 CPU 上，支持的数据类型有 `np.float16`，`np.float32` 和 `np.float64`。

参数：

- **x** (Tensor) - 输入数据。
- **dtype** (`mindspore.dtype`, 可选) - 默认值：None。覆盖输出 Tensor 的 `dtype`。

返回：

Tensor 或标量，逐元素计算的指数值。如果 *x1* 和 *x2* 都是标量，返回标量。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.exp(np.arange(5).astype(np.float32))
>>> print(output)
[ 1.          2.718282   7.3890557 20.085537  54.598145 ]
```

15.4.42 mindspore.numpy.exp2

`mindspore.numpy.exp2(x, dtype=None)`

计算输入数组中所有值 p 的 2^{**p} 。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 GPU 上, 支持的数据类型为 `np.float16` 和 `np.float32`。

参数:

- **x** (Tensor) - 输入值。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量, 逐元素计算 2 的 x 次幂。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([2, 3]).astype(np.float32)
>>> output = np.exp2(x)
>>> print(output)
[4. 8.]
```

15.4.43 mindspore.numpy.expm1

`mindspore.numpy.expm1(x, dtype=None)`

计算数组中所有元素的 $\exp(x) - 1$ 。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 GPU 上, 支持的数据类型有 `np.float16` 和 `np.float32`。在 CPU 上, 支持的数据类型有 `np.float16`, `np.float32` 和 `np.float64`。

参数:

- **x** (Tensor) - 输入数据。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量。逐元素计算 $\exp(x) - 1$ 。如果 $x1$ 和 $x2$ 都是标量, 返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.expm1(np.arange(5).astype(np.float32))
>>> print(output)
[ 0.          1.7182819  6.389056  19.085537  53.59815 ]
```

15.4.44 mindspore.numpy.fix

`mindspore.numpy.fix(x)`

舍入至最接近零的相邻整数。

将浮点数数组每个元素舍入为最接近零的相邻整数。舍入后的值以浮点数格式返回。

说明：不支持 NumPy 参数 `out`。

参数：

- **x** (Tensor) - 需要舍入的 float 数组。

返回：

Tensor。

异常：

- **TypeError** - 如果输入不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.fix(np.array([2.1, 2.9, -2.1, -2.9]))
>>> print(output)
[ 2.  2. -2. -2.]
```

15.4.45 mindspore.numpy.float_power

`mindspore.numpy.float_power(x1, x2, dtype=None)`

第一个数组逐元素计算幂次方，指数为第二个数组中对应的元素。

将 `x1` 中的每个基数以其位置对应 `x2` 中的元素作为指数，计算幂次方。`x1` 和 `x2` 必须能广播到相同的 `shape`。这与 `power` 函数的不同之处在于 `int`、`float16` 和 `float64` 都会提升为至少具有 `float32` 精度的浮点数，这样结果总是不精确的。这个函数意为为负幂返回一个可用的值，并且让正幂很少溢出。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。`int` 和 `float` 将被提升至 `float32` 而不是 `float64`。

参数：

- **x1** (Tensor) - 基数。
- **x2** (Tensor) - 指数。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。 $x1$ 中的基数以 $x2$ 中的对应元素为指数计算得到的幂次方。如果 $x1$ 和 $x2$ 都是标量，返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.arange(6)
>>> x2 = np.array(3)
>>> output = np.float_power(x1, x2)
>>> print(output)
[ 0.  1.  8. 27. 64. 125.]
```

15.4.46 mindspore.numpy.floor

`mindspore.numpy.floor(x, dtype=None)`

返回输入的向下取整，逐元素计算。

标量 x 的向下取整是满足 $i \leq x$ 的最大整数 i 。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 GPU 上，支持的数据类型有 `np.float16` 和 `np.float32`。在 CPU 上，支持的数据类型有 `np.float16`，`np.float32` 和 `np.float64`。

参数：

- **x** (Tensor) - 输入数据。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。 x 中各元素的向下取整。如果 x 是标量，返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.floor(np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0]))
>>> print(output)
[-2. -2. -1.  0.  1.  1.  2.]
```

15.4.47 mindspore.numpy.floor_divide

`mindspore.numpy.floor_divide(x1, x2, dtype=None)`

对输入进行除法运算，返回小于等于除法结果的最大整数。该函数等同于 Python 中的 `//` 运算符，并与运算符 `%` (取余) 配对使用，使得 $a = a \% b + b * (a // b)$ 在四舍五入后成立。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.floor_divide(np.array([1., 2., 3., 4.]), np.array(2.5))
>>> print(output)
[0. 0. 1. 1.]
```

15.4.48 mindspore.numpy.fmod

`mindspore.numpy.fmod(x1, x2, dtype=None)`

返回除法的逐元素余数。

这是 C 语言库中函数 `fmod` 的 NumPy 实现，余数与被除数 `x1` 的符号相同。它等同于 Matlab(TM) 的 `rem` 函数，勿与 Python 的取模运算符 `x1 % x2` 混淆。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **x1** (Tensor) - 第一个输入数组。
- **x2** (Tensor) - 第二个输入数组。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。`x1` 除以 `x2` 所得余数。如果 `x1` 和 `x2` 都是标量，返回标量。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.fmod(np.array([-3, -2, -1, 1, 2, 3]), np.array(2))
>>> print(output)
[-1  0 -1  1  0  1]
```

15.4.49 mindspore.numpy.gcd

`mindspore.numpy.gcd(x1, x2, dtype=None)`

返回 $|x1|$ 和 $|x2|$ 的最大公约数。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x1** (Tensor) - 输入数据。
- **x2** (Tensor) - 输入数据。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。输入的绝对值的最大公约数。如果 *x1* 和 *x2* 都是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.gcd(np.arange(6), np.array(20))
>>> print(output)
[20  1  2  1  4  5]
```

15.4.50 mindspore.numpy.gradient

`mindspore.numpy.gradient(f, *varargs, axis=None, edge_order=1)`

返回一个 *N* 维数组的梯度。该梯度是通过在内部点使用二阶精确的中心差分法, 在边界使用一阶或二阶精确的单侧(前向或后向)差分法计算得出的。因此返回的梯度与输入数组具有相同 *shape*。

说明: 目前我们仅支持 *edge_order* = 1 且 *varargs* 间距均匀。

参数:

- **f** (Union[tuple, list, Tensor]) - 一个包含标量函数样本的 *N* 维数组。
- **varargs** (Union[tuple[number], tuple[tensor scalar]], 可选) - *f* 值之间的间距。在所有维度下默认为单位间距。间距可以使用以下方式指定: 1. 单个标量指定所有维度的样本距离。2. *N* 个标量指定每个维度的恒定样本距离。

- **axis** (Union[None, int, tuple(int), list(int)], 可选) - 梯度只沿给定的单个或多个轴计算。默认情况下 (`axis = None`) 计算输入 Tensor 的所有轴的梯度。`axis` 可能为负, 在这种情况下, 倒数计数。
- **edge_order** (int, 可选) - 梯度使用 N 阶边界差分计算。默认值: 1。

返回:

梯度, 元素为 Tensor 的 list(如果只有一个维度需要计算, 则为单个 Tensor)。每个导数的 shape 与 f 相同。

异常:

- **TypeError** - 如果输入类型不符合上述规定。
- **ValueError** - 如果 `axis` 超出范围, 或 f 的 shape 中有维度小于 1
- **NotImplementedError** - 如果 `edge_order != 1`, 或 `varargs` 中包含非标量条目。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.gradient([[1, 2, 6], [3, 4, 5]], axis=-1)
>>> print(output)
[[1.  2.5 4. ]
 [1.  1.  1. ]]
```

15.4.51 mindspore.numpy.heaviside

`mindspore.numpy.heaviside(x1, x2, dtype=None)`

计算 Heaviside 阶跃函数。

说明: 不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数:

- **x1** (Tensor) - 输入值。
- **x2** (Tensor) - 当 $x1$ 为 0 时函数的值。如果 `x1.shape != x2.shape`, 他们必须能广播到一个共同的 shape(即输出的 shape)。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量, 输出数组, 逐元素计算 $x1$ 的 Heaviside 阶跃函数。如果 $x1$ 和 $x2$ 都是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.heaviside(np.array([-1.5, 0, 2.0]), np.array(0.5))
>>> print(output)
[0.  0.5 1. ]
```

(下页继续)

(续上页)

```
>>> output = np.heaviside(np.array([-1.5, 0, 2.0]), np.array(1))
>>> print(output)
[0. 1. 1.]
```

15.4.52 mindspore.numpy.histogram

`mindspore.numpy.histogram(a, bins=10, range=None, weights=None, density=False)`

计算数据集的直方图。

说明: 不支持 *bins* 为 `string` 类型。不支持已弃用的 NumPy 参数 *normed* 。

参数:

- **a** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入数据。直方图在展平的数组上计算。
- **bins** (`Union[int, tuple, list, Tensor]`, 可选) - 如果 *bins* 为 `int`, 它定义了给定范围 (默认为 10) 中等宽的桶的数量。如果 *bins* 是一个序列, 它定义了桶的边界, 包括最右边界, 允许非均匀的桶宽度。默认值: 10。
- **range** (`(float, float)`, 可选) - 桶的上下界。如果未提供, *range* 简单定义成 `(a.min(), a.max())`。范围之外的值将被忽略。*range* 的第一个元素必须小于等于第二个元素。默认值: `None`。
- **weights** (`Union[int, float, bool, list, tuple, Tensor]`, 可选) - 一个与 *a* `shape` 相同的权重数组。如果 *density* 为 `True`, 权重将被归一化, 使得范围内的密度总和为 1。默认值: `None`。
- **density** (`boolean`, 可选) - 如果为 `False`, 结果将包含每个桶中的样本数量。如果为 `True`, 结果是桶的位置处概率密度函数的值, 归一化使得其在整个范围上的积分为 1。请注意, 直方图值的总和不会等于 1, 除非选择了单位宽度的桶。它不是一个概率质量函数。默认值: `False`。

返回:

(`Tensor, Tensor`), 直方图的值和桶的边界。

异常:

- **ValueError** - 如果 *x* 和 *weights* 的 `size` 不相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import numpy as np
>>> print(np.histogram([1, 2, 1], bins=[0, 1, 2, 3]))
(Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  2.00000000e+00,  1.
↪00000000e+00]),
Tensor(shape=[4], dtype=Int32, value= [0, 1, 2, 3]))
>>> print(np.histogram(np.arange(4), bins=np.arange(5), density=True))
(Tensor(shape=[4], dtype=Float32, value=
[ 2.50000000e-01,  2.50000000e-01,  2.50000000e-01,  2.50000000e-01]),
Tensor(shape=[5], dtype=Int32, value= [0, 1, 2, 3, 4]))
>>> print(np.histogram([1, 2, 1], [1, 0, 1], bins=[0,1,2,3]))
(Tensor(shape=[3], dtype=Float32, value= [ 1.00000000e+00,  4.00000000e+00,  1.
↪00000000e+00]),
Tensor(shape=[4], dtype=Int32, value= [0, 1, 2, 3]))
```

15.4.53 mindspore.numpy.histogram2d

`mindspore.numpy.histogram2d(x, y, bins=10, range=None, weights=None, density=False)`
 计算数据的二维直方图。

说明: 不支持已弃用的 NumPy 参数 *normed*。

参数:

- **x** (Union[list, tuple, Tensor]) - 一个 shape 为 (N,) 的数组, 包含要进行直方图统计的点的 x 坐标。
- **y** (Union[list, tuple, Tensor]) - 一个 shape 为 (N,) 的数组, 包含要进行直方图统计的点的 y 坐标。
- **bins** (Union[int, tuple, list], 可选) - 桶的规格: 如果为 int, 为两个维度指定桶的数量 ($n_x=n_y=bins$)。如果为 array_like, 指定两个维度的桶的边界 ($x\_edges=y\_edges=bins$)。如果为 [int, int], 指定每个维度的桶的数量 ($n_x, n_y = bins$)。如果为 [array, array], 指定每个维度的桶的边界 ($x\_edges, y\_edges = bins$)。如果为组合 [int, array] 或 [array, int], 其中 int 是桶的数量, array 是桶的边界。默认值: 10。
- **range** (Union[list, tuple], 可选) - shape 为 (2, 2) 的数组, 指定每个维度的桶的最左和最右边界 (如果在 bins 参数中没有明确指定): $[[xmin, xmax], [ymin, ymax]]$ 。范围之外的所有值都将被视为异常值, 不计入直方图。默认值: None。
- **weights** (Union[list, tuple, Tensor], 可选) - 一个 shape 为 (N,) 的数组, 包含每个样本 (x_i, y_i) 的权重 w_i 。默认值: None。
- **density** (boolean, 可选) - 如果为 False, 默认返回每个桶中的样本计数。如果为 True, 则返回桶位置处的概率密度函数值, $bin\_count / sample\_count / bin\_volume$ 。默认值: False。

返回:

(Tensor, Tensor, Tensor), 双向直方图的值及其在第一和第二维度上的桶的边界。

异常:

- **ValueError** - 如果 range 的 size 不等于样本数量。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import numpy as np
>>> x = np.arange(5)
>>> y = np.arange(2, 7)
>>> print(np.histogram2d(x, y, bins=(2, 3)))
(Tensor(shape=[2, 3], dtype=Float32, value=
[[ 2.00000000e+00,  0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00]]),
Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  2.00000000e+00,  4.00000000e+00]),
Tensor(shape=[4], dtype=Float32, value=
[ 2.00000000e+00,  3.33333349e+00,  4.66666698e+00,  6.00000000e+00]))
```

15.4.54 mindspore.numpy.histogramdd

`mindspore.numpy.histogramdd` (*sample*, *bins=10*, *range=None*, *weights=None*, *density=False*)
计算数据的多维直方图。

说明: 不支持已弃用的 NumPy 参数 *normed*。

参数:

- **sample** (Union[list, tuple, Tensor]) - 要进行直方图统计的数据, 可以是 (N, D) 数组, 或者 (D, N) 的类数组。注意当使用类数组为样本的非常规解释: 当样本数组时, 每行是 D 维空间中的一个坐标, 例如 `histogramdd(np.array([p1, p2, p3]))`。当样本为类数组时, 每个元素是单一坐标系的值列表, 如 `histogramdd((X, Y, Z))`。应优先考虑第一种形式。
- **bins** (Union[int, tuple, list], 可选) - 桶的规格: 一个数组序列用于描述沿每个维度单调递增的桶的边界。每个维度的桶的数量 (`nx, ny, ... =bins`)。所有维度的桶的数量 (`nx=ny...=bins`)。默认值: 10。
- **range** (Union[list, tuple], 可选) - 长度为 D 的序列, 每个维度可以选择一个 (`lower, upper`) tuple 来指定桶的外边界, 如果边界在 *bins* 没有特别给出。序列中的 `None` 项意味着其对应维度使用最小值和最大值。默认值 `None` 相当于给 D 中 Tuple 传递 `None` 值。默认值: `None`。
- **weights** (Union[list, tuple, Tensor], 可选) - 一个 shape 为 $(N,)$ 的数组, 其中的值 `w_i` 表示每个样本的权重 (`x_i, y_i, z_i, ...`)。默认值: `None`。
- **density** (boolean, 可选) - 如果为 `False` (默认值), 则返回每个区间中的样本数量。如果为 `True`, 则返回该区间的概率密度函数 `bin_count / sample_count / bin_volume`。默认值: `False`。

返回:

Tensor, 元素为 Tensor 的 list, 直方图的值和桶边界。

异常:

- **ValueError** - 如果 *range* 的 size 不等于样本数量。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import numpy as np
>>> sample = np.arange(15).reshape(5, 3)
>>> print(sample)
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [12 13 14]]
>>> print(np.histogramdd(sample, bins=(2, 3, 4)))
(Tensor(shape=[2, 3, 4], dtype=Float32, value=
[[[ 1.00000000e+00,  1.00000000e+00,  0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00]],
 [[ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00,  1.00000000e+00,  0.00000000e+00],
 [ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  2.00000000e+00]]]),
 [Tensor(shape=[3], dtype=Float32, value= [ 0.00000000e+00,  6.00000000e+00,  1.
↪200000000e+01]),
```

(下页继续)

(续上页)

```
Tensor(shape=[4], dtype=Float32, value=
[ 1.00000000e+00,  5.00000000e+00,  9.00000000e+00,  1.30000000e+01]),
Tensor(shape=[5], dtype=Float32, value=
[ 2.00000000e+00,  5.00000000e+00,  8.00000000e+00,  1.10000000e+01,  1.40000000e+01]))
```

15.4.55 mindspore.numpy.hypot

`mindspore.numpy.hypot(x1, x2, dtype=None)`

给定直角三角形的直角边，返回其斜边。

等价于逐元素计算 $\sqrt{x1^2 + x2^2}$ 。如果 *x1* 或 *x2* 是类标量 (即可以无歧义地转换为标量)，它将被广播到另一个参数的每个元素执行计算。(参见示例)

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 CPU 上，支持的数据类型有 `np.float16`、`np.float32` 和 `np.float64`。

参数：

- **x1 (Tensor)** - 三角形的直角边。
- **x2 (Tensor)** - 三角形的直角边。如果 `x1.shape != x2.shape`，他们必须可以广播到一个共同的 `shape` (这将成为输出的形状)。
- **dtype (mindspore.dtype, 可选)** - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。三角形的斜边。如果 *x1* 和 *x2* 都是标量，返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.hypot(3*np.ones((3, 3)), 4*np.ones((3, 3)))
>>> print(output)
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
>>> output = np.hypot(3*np.ones((3, 3)), np.array([4.0]))
>>> print(output)
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
```

15.4.56 mindspore.numpy.inner

`mindspore.numpy.inner(a, b)`

返回两个 Tensor 的内积。

对于 1-D 的 Tensor，这是向量的普通内积 (不包含复共轭)。对于更高维的 Tensor，这是在最后一个轴上的求和积。

说明：不支持 NumPy 参数 `out`。在 GPU 上，支持的数据类型有 `np.float16` 和 `np.float32`。在 CPU 上，支持的数据类型有 `np.float16`，`np.float32` 和 `np.float64`。

参数：

- **a** (Tensor) - 输入 Tensor。如果 *a* 和 *b* 都不是标量，它们的最后一个维度必须匹配。
- **b** (Tensor) - 输入 Tensor。如果 *a* 和 *b* 都不是标量，它们的最后一个维度必须匹配。

返回：

Tensor 或标量。

异常：

- **ValueError** - 如果 `x1.shape[-1] != x2.shape[-1]`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((5, 3))
>>> b = np.ones((2, 7, 3))
>>> output = np.inner(a, b)
>>> print(output)
[[[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]
 [[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]
 [[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]
 [[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]
 [[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]
 [[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]
 [[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]
 [[3. 3. 3. 3. 3. 3. 3. 3.]
 [3. 3. 3. 3. 3. 3. 3. 3.]]]
```

15.4.57 mindspore.numpy.interp

`mindspore.numpy.interp(x, xp, fp, left=None, right=None)`

用于单调递增的样本点的一维线性插值。返回包含给定离散数据点 (`xp`, `fp`) 的函数进行一维分段线性插值后 *x* 处的函数值。

说明：Numpy 参数 `period` 不受支持。不支持复数值。

参数：

- **x** (Union[int, float, bool, list, tuple, Tensor]) - 计算插值后的值时使用的 x 坐标。
- **xp** (Union[int, float, bool, list, tuple, Tensor]) - 元素为 float 的 1-D 序列, 输入数据点的 x 坐标, 必须递增。
- **fp** (Union[int, float, bool, list, tuple, Tensor]) - 元素为 float 的 1-D 序列, 输入数据点的 y 坐标, 与 *xp* 等长。
- **left** (float, 可选) - $x < xp[0]$ 时返回的值, 一旦存在, 默认值为 $fp[0]$ 。默认值: None。
- **right** (float, 可选) - $x > xp[-1]$ 时返回的值, 一旦存在, 默认值为 $fp[-1]$ 。默认值: None。

返回:

Tensor, 插值得到的值, 其 shape 与 *x* 相同。

异常:

- **ValueError** - 如果 *xp* 或 *fp* 不是一维的, 或如果 *xp* 和 *fp* 的长度不同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> xp = [1, 2, 3]
>>> fp = [3, 2, 0]
>>> print(np.interp([0, 1, 1.5, 2.72, 3.14], xp, fp))
[3.      3.      2.5      0.55999994 0.      ]
>>> UNDEF = -99.0
>>> print(np.interp(3.14, xp, fp, right=UNDEF))
-99.0
```

15.4.58 mindspore.numpy.invert

`mindspore.numpy.invert(x, dtype=None)`

逐元素计算按位取反或按位非。对输入数组中整数的底层二进制表示执行按位非操作。该通用函数实现了 C/Python 运算符 `~`。对于有符号的整数输入, 返回其二进制补码表示。在二进制补码系统中, 负数由其绝对值的二进制补码表示。这是计算机上表示有符号整数的最常见方法。一个 *N* 位的二进制补码系统可以表示范围从 $-2^{\{N-1\}}$ 到 $+2^{\{N-1\}}-1$ 的所有整数。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 Ascend 上支持的数据类型为 `np.int16` 和 `np.uint16`。

参数:

- **x** (Tensor) - 仅处理 int 和 bool 类型。
- **dtype** (*mindspore.dtype*, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。

支持平台:

Ascend

样例:

```
>>> import mindspore.numpy as np
>>> print(np.invert(np.array(13, dtype=np.uint16)))
65522
```

15.4.59 mindspore.numpy.kron

`mindspore.numpy.kron(a, b)`

两个数组的 Kronecker 积。

计算 Kronecker 积，即先将第二个数组分块，然后使用第一个数组缩放后组成的复合数组。

说明：不支持 bool 值。

参数：

- **a** (Union[int, float, list, tuple, Tensor]) - 输入值。
- **b** (Union[int, float, list, tuple, Tensor]) - 输入值。

返回：

Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.kron([1,10,100], [5,6,7])
>>> print(output)
[ 5  6  7 50 60 70 500 600 700]
>>> output = np.kron([5,6,7], [1,10,100])
>>> print(output)
[ 5 50 500  6 60 600  7 70 700]
>>> output = np.kron(np.eye(2), np.ones((2,2)))
>>> print(output)
[[1. 1. 0. 0.]
 [1. 1. 0. 0.]
 [0. 0. 1. 1.]
 [0. 0. 1. 1.]
```

15.4.60 mindspore.numpy.lcm

`mindspore.numpy.lcm(x1, x2, dtype=None)`

返回 $|x1|$ 和 $|x2|$ 的最小公倍数。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x1** (Tensor) - 输入数据。

- **x2** (Tensor) - 输入数据。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 输入绝对值的最小公倍数。如果 *x1* 和 *x2* 都是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.lcm(np.arange(6), np.array(20))
>>> print(output)
[ 0 20 20 60 20 20]
```

15.4.61 mindspore.numpy.log

`mindspore.numpy.log(x, dtype=None)`

返回自然对数, 逐元素计算。

自然对数 $\log$ 是指数函数的逆函数, 因此 $\log(\exp(x)) = x$ 。自然对数是以 *e* 为底的对数。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 GPU 上, 支持的数据类型有 `np.float16` 和 `np.float32`。在 CPU 上, 支持的数据类型有 `np.float16`, `np.float32` 和 `np.float64`。

参数:

- **x** (Tensor) - 输入数组。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, *x* 的自然对数, 逐元素计算。如果 *x* 是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([2, 3, 4]).astype('float32')
>>> output = np.log(x)
>>> print(output)
[0.69314575 1.09861 1.3862929 ]
```

15.4.62 mindspore.numpy.log10

`mindspore.numpy.log10(x, dtype=None)`
 x 的以 10 为底的对数。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。如果 x 是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([10, 100, 1000]).astype('float16')
>>> output = np.log10(x)
>>> print(output)
[1. 2. 3.]
```

15.4.63 mindspore.numpy.log1p

`mindspore.numpy.log1p(x, dtype=None)`
返回 1 加上输入数组的自然对数, 逐元素计算。
计算 $\log(1 + x)$ 。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x** (Tensor) - 输入数组。
- **dtype** (mindspore.dtype) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。如果 x 是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([1, 2, 3]).astype('float16')
>>> output = np.log1p(x)
>>> print(output)
[0.6934 1.099 1.387 ]
```

15.4.64 mindspore.numpy.log2

`mindspore.numpy.log2(x, dtype=None)`

x 的以 2 为底的对数。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。如果 x 是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([2, 4, 8]).astype('float16')
>>> output = np.log2(x)
>>> print(output)
[1. 2. 3.]
```

15.4.65 mindspore.numpy.logaddexp

`mindspore.numpy.logaddexp(x1, x2, dtype=None)`

计算输入指数取幂的和对数。计算 $\log(\exp(x1) + \exp(x2))$ 。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。如果 $x1.shape \neq x2.shape$, 它们必须可以广播到一个共同的 shape(即输出的 shape)。
- **dtype** (mindspore.dtype) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。如果 $x1$ 和 $x2$ 都是标量, 返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.array([1, 2, 3]).astype('float16')
>>> x2 = np.array(2).astype('float16')
>>> output = np.logaddexp(x1, x2)
>>> print(output)
[2.312 2.693 3.312]
```

15.4.66 mindspore.numpy.logaddexp2

`mindspore.numpy.logaddexp2(x1, x2, dtype=None)`

计算输入指数取幂的和的以 2 为底对数。

计算 $\log_2(2^{x1} + 2^{x2})$ 。在机器学习中，这个函数在计算的概率非常小以至于可能超出正常浮点数范围时很有用。即当概率值很小时，使用概率值的以 2 为底的对数表示。这个函数以这种方式存储概率值和。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x1** (Tensor) - 输入 Tensor。
- **x2** (Tensor) - 输出 Tensor。如果 `x1.shape != x2.shape`，则它们必须可以广播到一个共同的 shape(即输出的 shape)。
- **dtype** (mindspore.dtype) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。如果 *x1* 和 *x2* 都是标量，返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.array([2, 4, 8]).astype('float16')
>>> x2 = np.array(2).astype('float16')
>>> output = np.logaddexp2(x1, x2)
>>> print(output)
[3. 4.32 8.02]
```

15.4.67 mindspore.numpy.matmul

`mindspore.numpy.matmul(x1, x2, dtype=None)`
 返回两个数组的矩阵乘积。

说明: 不支持 NumPy 参数 *out*、*casting*、*order*、*subok*、*signature*、*extobj*。在 GPU 上, 支持的数据类型有 `np.float16` 和 `np.float32`。在 CPU 上, 支持的数据类型有 `np.float16` 和 `np.float32`。

参数:

- **x1** (Tensor) - 输入 Tensor, 不允许标量。
- **x2** (Tensor) - 输入 Tensor, 不允许标量。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: *None*。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量, 输入的矩阵乘积。如果 *x1* 和 *x2* 都是 1-d 向量, 返回标量。

异常:

- **ValueError** - 如果 *x1* 的最后一个维度的大小不等于 *x2* 的倒数第二个维度的大小, 或者传入了标量值。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x1 = np.arange(2*3*4).reshape(2, 3, 4).astype('float32')
>>> x2 = np.arange(4*5).reshape(4, 5).astype('float32')
>>> output = np.matmul(x1, x2)
>>> print(output)
[[[ 70.   76.   82.   88.   94.]
 [ 190.  212.  234.  256.  278.]
 [ 310.  348.  386.  424.  462.]]
 [[ 430.  484.  538.  592.  646.]
 [ 550.  620.  690.  760.  830.]
 [ 670.  756.  842.  928. 1014.]]]
```

15.4.68 mindspore.numpy.matrix_power

`mindspore.numpy.matrix_power(a, n)`
 计算方阵以整数 *n* 为指数的幂。

对于正整数 *n*, 通过重复的矩阵平方和矩阵乘法来计算幂。如果 *n* == 0, 返回与 *M* 相同 shape 的单位矩阵。

说明: 目前不支持堆叠的对象矩阵, 也不支持 *n* < 0。

参数:

- **a** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入矩阵。
- **n** (`int`) - 指数可以是任意整数或长整数, 正数或零。

返回：

Tensor。

异常：

- **TypeError** - 如果输入不能转换为 Tensor 或指数不是整数。
- **ValueError** - 如果输入的维度少于 2 或最后两个维度不是方阵。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import numpy as np
>>> a = np.arange(16).reshape(4, 4).astype('float32')
>>> print(np.matrix_power(a, 2))
[[ 56.  62.  68.  74.]
 [152. 174. 196. 218.]
 [248. 286. 324. 362.]
 [344. 398. 452. 506.]]
```

15.4.69 mindspore.numpy.maximum

`mindspore.numpy.maximum(x1, x2, dtype=None)`

逐元素比较两个数组，返回每对数组元素中的最大值。

比较两个数组，并返回一个包含逐元素最大值的新数组。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 Ascend 上，不支持包含 inf 或 NaN 的输入数组。

参数：

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 包含要比较元素的数组。如果 `x1.shape != x2.shape`，它们必须可以广播到一个共同的 shape(即输出的 shape)。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量，逐元素返回 *x1* 和 *x2* 中的最大值。如果 *x1* 和 *x2* 都是标量，返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.maximum(np.array([2, 3, 4]), np.array([1, 5, 2]))
>>> print(output)
[2 5 4]
```

15.4.70 mindspore.numpy.mean

`mindspore.numpy.mean(a, axis=None, keepdims=False, dtype=None)`

沿指定轴计算算术平均值。

返回数组元素的平均值。默认情况下，平均值是在展平的数组上计算的，否则在指定的轴上计算。

说明：不支持 NumPy 参数 `out`。在 GPU 上，支持的数据类型有 `np.float16` 和 `np.float32`。

参数：

- **a** (Tensor) - 包含要计算均值的数字的输入 Tensor。如果 *a* 不是数组，将尝试进行转换。
- **axis** (Union[int, tuple(int), None], 可选) - 计算均值的所沿的一个或多个轴。默认计算展平数组的均值。如果这是一个整数 tuple，将在多个轴上计算均值。默认值: *None*。
- **keepdims** (bool, 可选) - 如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和输入 Tensor 同一个维度数。默认值: *False*。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量，包含元素为所得均值的数组。

异常：

- **ValueError** - 如果 *axes* 的范围超过 $[-a.ndim, a.ndim)$ ，或如果 *axes* 包含重复项。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.arange(6, dtype='float32')
>>> output = np.mean(a, 0)
>>> print(output)
2.5
```

15.4.71 mindspore.numpy.minimum

`mindspore.numpy.minimum(x1, x2, dtype=None)`

逐元素比较两个数组，返回每对数组元素中的最小值。

比较两个数组，并返回一个包含逐元素最小值的新数组。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。在 Ascend 上，不支持包含 `inf` 或 `NaN` 的输入数组。

参数：

- **x1** (Tensor) - 第一个需比较的输入数组。
- **x2** (Tensor) - 第二个需比较的输入数组。

- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 逐元素返回 $x1$ 和 $x2$ 中的最小值。

异常:

- **TypeError** - 如果输入类型不是上述指定类型。
- **ValueError** - 如果 $x1$ 和 $x2$ 不能被广播。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.asarray([1, 2])
>>> b = np.asarray([[1, 3], [1, 4]])
>>> print(np.minimum(a, b))
[[1 2]
 [1 2]]
```

15.4.72 mindspore.numpy.multi_dot

`mindspore.numpy.multi_dot(arrays)`

计算两个或更多个数组的点积, 同时自动选择最快的计算顺序。

multi_dot 将链式调用 `numpy.dot`, 并使用矩阵的最佳括号化。根据矩阵的 **shape**, 这可以大大加快乘法速度。如果第一个参数是一维的, 则将其视为行向量。如果最后一个参数是一维的, 则将其视为列向量。其他参数必须是二维的。

说明: 不支持 NumPy 参数 *out*。

参数:

- **arrays** (类数组的序列) - 如果第一个参数是一维的, 则将其视为行向量。如果最后一个参数是一维的, 则将其视为列向量。其他参数必须是二维的。

返回:

Tensor, 所提供数组的点积。

异常:

- **ValueError** - 数组不是二维的。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> A = np.ones((10000, 100))
>>> B = np.ones((100, 100))
>>> C = np.ones((100, 5))
>>> D = np.ones((5, 333))
```

(下页继续)

(续上页)

```
>>> output = np.multi_dot([A, B, C, D])
>>> print(output)
[[50000. 50000. 50000. ... 50000. 50000. 50000.]
 [50000. 50000. 50000. ... 50000. 50000. 50000.]
 [50000. 50000. 50000. ... 50000. 50000. 50000.]
 ...
 [50000. 50000. 50000. ... 50000. 50000. 50000.]
 [50000. 50000. 50000. ... 50000. 50000. 50000.]
 [50000. 50000. 50000. ... 50000. 50000. 50000.]]
```

15.4.73 mindspore.numpy.multiply

`mindspore.numpy.multiply(x1, x2, dtype=None)`

参数逐元素相乘。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x1** (Tensor) - 需相乘的输入 Tensor。
- **x2** (Tensor) - 需相乘的输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 逐元素返回 *x1* 和 *x2* 的乘积。如果 *x1* 和 *x2* 都是标量, 返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x1 = np.full((3, 2), [1, 2])
>>> x2 = np.full((3, 2), [3, 4])
>>> output = np.multiply(x1, x2)
>>> print(output)
[[3 8]
 [3 8]
 [3 8]]
```

15.4.74 mindspore.numpy.nancumsum

`mindspore.numpy.nancumsum(a, axis=None, dtype=None)`

返回给定轴上数组元素的累积和，将 NaN(非数值) 视为零。

遇到 NaN 时，累积和不变，数组中的 NaN 将替换为零。对于全是 NaN 或为空的切片，返回零。

说明：如果 `a.dtype` 是 `int8`, `int16` 或 `bool`，结果的 `dtype` 将提升为 `int32`。

参数：

- **a** (Tensor) - 输入 Tensor。
- **axis** (int, 可选) - 计算累积和所沿轴。若取默认值 (`None`)，将在展平的数组上计算累积和。
- **dtype** (`mindspore.dtype`, 可选) - 如果未指定，则与 `a` 相同，除非 `a` 具有精度低于默认平台整数的整数 `dtype`。在这种情况下，使用默认平台整数。默认值: `None`。

返回：

Tensor。

异常：

- **TypeError** - 如果输入参数的类型未在上述范围内。
- **ValueError** - 如果 `axis` 超出范围。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([[1, 2], [3, np.nan]])
>>> output = np.nancumsum(a)
>>> print(output)
[1. 3. 6. 6.]
>>> output = np.nancumsum(a, axis=0)
>>> print(output)
[[1. 2.]
 [4. 2.]]
>>> output = np.nancumsum(a, axis=1)
>>> print(output)
[[1. 3.]
 [3. 3.]]
```

15.4.75 mindspore.numpy.nanmax

`mindspore.numpy.nanmax(a, axis=None, dtype=None, keepdims=False)`

返回数组的最大值或沿某个轴的最大值，忽略 NaN。

说明：不支持 NumPy 参数 `out`。对于全是 NaN 的 slice，返回一个非常小的负数，而不是 NaN。

参数：

- **a** (Union[int, float, list, tuple, Tensor]) - 包含要计算最大值的数的数组。如果 *a* 不是数组，将尝试进行转换。
- **axis** (Union[int, tuple(int), None], 可选) - 默认值: *None*。计算最大值所沿的单个或多个轴。若取默认值，计算展平数组的最大值。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。
- **keepdims** (boolean, 可选) - 默认值: *False*。如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和 *a* 同一个维度数。

返回：

Tensor。

异常：

- **ValueError** - 如果 *axis* 超出 `[-a.ndim, a.ndim)` 范围，或者 *axis* 包含重复项。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([[1, 2], [3, np.nan]])
>>> output = np.nanmax(a)
>>> print(output)
3.0
>>> output = np.nanmax(a, axis=0)
>>> print(output)
[3. 2.]
```

15.4.76 mindspore.numpy.nanmean

`mindspore.numpy.nanmean(a, axis=None, dtype=None, keepdims=False)`

沿指定轴计算算术平均值，忽略 NaN。

返回数组元素的平均值。默认情况下，平均值在展平的数组上计算，否则在指定轴上计算。对于整数输入，中间值和返回值使用 float32 类型。

说明：不支持 NumPy 参数 `out`。

参数：

- **a** (Union[int, float, list, tuple, Tensor]) - 包含要计算均值的数的数组。如果 *a* 不是数组，将尝试进行转换。

- **axis** (Union[int, tuple of int, None], 可选) - 计算均值所沿的单个或多个轴。若取默认值，计算展平数组的均值。默认值: *None*。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。
- **keepdims** (boolean, 可选) - 默认值: *False*。如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和 *a* 同一个维度数。

返回：

Tensor。

异常：

- **ValueError** - 如果 *axis* 超出 $[-a.ndim, a.ndim)$ 范围，或者 *axis* 包含重复项。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([[1, np.nan], [3, 4]])
>>> output = np.nanmean(a)
>>> print(output)
2.6666667
>>> output = np.nanmean(a, axis=0)
>>> print(output)
[2. 4.]
>>> output = np.nanmean(a, axis=1)
>>> print(output)
[1. 3.5]
```

15.4.77 mindspore.numpy.nanmin

`mindspore.numpy.nanmin(a, axis=None, dtype=None, keepdims=False)`

返回数组的最大值或沿某个轴的最大值，忽略 NaN。

说明：不支持 NumPy 参数 *out*。对于全是 NaN 的 slice，返回一个非常小的负数，而不是 NaN。在 Ascend 上，由于目前不支持检查 NaN，不推荐使用 `np.nanmin`。如果数组不包含 NaN，应使用 `np.min`。

参数：

- **a** (Union[int, float, list, tuple, Tensor]) - 包含要计算最小值的数的数组。如果 *a* 不是数组，将尝试进行转换。
- **axis** (Union[int, tuple(int), None], 可选) - 计算最小值所沿的单个或多个轴。若取默认值，计算展平数组的最小值。默认值: *None*。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。
- **keepdims** (boolean, 可选) - 默认值: *False*。如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和 *a* 同一个维度数。

返回：

Tensor。

异常：

- **ValueError** - 如果 *axis* 超出 $[-a.ndim, a.ndim)$ 范围, 或者 *axis* 包含重复项。

支持平台:

GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.array([[1, 2], [3, np.nan]])
>>> output = np.nanmin(a)
>>> print(output)
1.0
>>> output = np.nanmin(a, axis=0)
>>> print(output)
[1. 2.]
```

15.4.78 mindspore.numpy.nanstd

`mindspore.numpy.nanstd(a, axis=None, dtype=None, ddof=0, keepdims=False)`

计算指定轴上元素的标准差, 忽略 NaN。

返回非 NaN 数组元素的标准差, 即分布的离散程度。默认情况下, 标准差在展平数组上计算, 否则在指定的轴上计算。

说明: 不支持 NumPy 参数 *out*。在 GPU 上, 支持的数据类型为 `np.float16` 和 `np.float32`。

参数:

- **a** (`Union[int, float, list, tuple, Tensor]`) - 计算非 NaN 值的标准差。
- **axis** (`Union[int, tuple of int, None]`, 可选) - 计算标准差所沿的单个或多个轴。若取默认值, 计算展平数组的标准差。默认值: *None*。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: *None*。覆盖输出 Tensor 的 `dtype`。
- **ddof** (`int`, 可选) - “自由度偏差”: 计算中使用的除数是 $N - ddof$, 其中 N 表示非 NaN 元素的数量。默认情况下 *ddof* 为零。
- **keepdims** (`boolean`, 可选) - 默认值: *False*。如果设置为 *True*, 减少的轴在结果中保留为大小为 1 的维度。若使用此选项, 结果会广播到和 *a* 同一个维度数。

返回:

Tensor。

异常:

- **ValueError** - 如果 *axis* 超出 $[-a.ndim, a.ndim)$ 范围, 或者 *axis* 包含重复项。

支持平台:

GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.array([[1, np.nan], [3, 4]])
>>> output = np.nanstd(a)
>>> print(output)
1.2472192
>>> output = np.nanstd(a, axis=0)
>>> print(output)
[1. 0.]
>>> output = np.nanstd(a, axis=1)
>>> print(output)
[0. 0.5]
```

15.4.79 mindspore.numpy.nansum

`mindspore.numpy.nansum(a, axis=None, dtype=None, keepdims=False)`

计算指定轴上元素的总和，将 NaN(非数值) 视为零。

说明：不支持 NumPy 参数 `out`。

参数：

- **a** (Union[int, float, list, tuple, Tensor]) - 包含要计算总和的数的数组。如果 *a* 不是数组，将尝试进行转换。
- **axis** (Union[int, tuple of int, None], 可选) - 计算总和所沿的单个或多个轴。若取默认值，计算展平数组的总和。默认值: *None*。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。
- **keepdims** (boolean, 可选) - 默认值: *False*。如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和 *a* 同一个维度数。

返回：

Tensor。

异常：

- **ValueError** - 如果 *axis* 超出 `[-a.ndim, a.ndim)` 范围，或者 *axis* 包含重复项。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([[1, 1], [1, np.nan]])
>>> output = np.nansum(a)
>>> print(output)
3.0
>>> output = np.nansum(a, axis=0)
>>> print(output)
[2. 1.]
```

15.4.80 mindspore.numpy.nanvar

`mindspore.numpy.nanvar(a, axis=None, dtype=None, ddof=0, keepdims=False)`

计算指定轴上的方差，忽略 NaN。

返回数组元素的方差，方差是分布离散程度的度量。默认情况下，方差在展平数组上计算，否则在指定轴上计算。

说明：不支持 NumPy 参数 `out`。在 GPU 上，支持的数据类型为 `np.float16` 和 `np.float32`。

参数：

- **a** (`Union[int, float, list, tuple, Tensor]`) - 包含要计算方差的数的数组。如果 *a* 不是数组，将尝试进行转换。
- **axis** (`Union[int, tuple of int, None]`, 可选) - 计算方差所沿的单个或多个轴。若取默认值，计算展平数组的方差。默认值: *None*。
- **dtype** (`mindspore.dtype`, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。
- **ddof** (`int`, 可选) - “自由度偏差”：计算中使用的除数是 $N - \text{ddof}$ ，其中 *N* 表示非 NaN 元素的数量。默认情况下 *ddof* 为零。
- **keepdims** (`boolean`, 可选) - 默认值: *False*。如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和 *a* 同一个维度数。

返回：

Tensor。

异常：

- **ValueError** - 如果 *axis* 超出 `[-a.ndim, a.ndim)` 范围，或者 *axis* 包含重复项。

支持平台：

GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.array([[1, np.nan], [3, 4]])
>>> output = np.nanvar(a)
>>> print(output)
1.5555557
>>> output = np.nanvar(a, axis=0)
>>> print(output)
[1. 0.]
>>> output = np.nanvar(a, axis=1)
>>> print(output)
[0. 0.25]
```

15.4.81 mindspore.numpy.negative

`mindspore.numpy.negative(a, dtype=None)`
数值符号取反，逐元素操作。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **a** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回：

Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.asarray([1, -1]).astype('float32')
>>> output = np.negative(a)
>>> print(output)
[-1. 1.]
```

15.4.82 mindspore.numpy.norm

`mindspore.numpy.norm(x, ord=None, axis=None, keepdims=False)`

矩阵或向量的范数。根据 *ord* 参数的值，此函数能够返回八种不同的矩阵范数之一，或者无限多种向量范数之一（见下文）。

说明：不支持计算矩阵的核范数和二范数

参数：

- **x** (Union[int, float, bool, list, tuple, Tensor]) - 输入数组。如果 *axis* 为 *None*，则 *x* 必须是 1-D 或 2-D，除非 *ord* 为 *None*。如果 *axis* 和 *ord* 都为 *None*，则返回 *x.ravel* 的二范数。
- **ord** (Union[None, 'fro', 'nuc', inf, -inf, int, float], 可选) - 范数的阶数。inf 表示 NumPy 中的 inf 对象。默认值: *None*。
- **axis** (Union[int, 2-tuple(int), None], 可选) - 如果 *axis* 是整数，则指定计算向量范数所沿的 *x* 的轴。如果 *axis* 是 2-tuple，则指定保存 2-D 矩阵的轴，并计算这些矩阵的矩阵范数。如果 *axis* 为 *None*，则返回向量范数（当 *x* 为 1-D 时）或矩阵范数（当 *x* 为 2-D 时）。默认为 *None*。
- **keepdims** (boolean, 可选) - 默认值: *False*。如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和 *x* 同一个维度数。

返回：

Tensor。矩阵或向量的范数。

异常:

- **ValueError** - 如果范数的阶数未定义。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.norm(np.arange(9).astype(np.float32)))
14.282857
```

15.4.83 mindspore.numpy.outer

`mindspore.numpy.outer(a, b)`

计算两个向量的外积。

指定两个向量, $a = [a_0, a_1, \dots, a_M]$ 和 $b = [b_0, b_1, \dots, b_N]$, 外积为:

```
[[a0*b0 a0*b1 ... a0*bN ]
 [a1*b0 . . ]
 [ ... . . ]
 [aM*b0 aM*bN ]]
```

说明: 不支持 NumPy 参数 `out`。在 GPU 上, 支持的数据类型为 `np.float16`。在 CPU 上, 支持的数据类型为 `np.float32` 和 `np.float64`。

参数:

- **a** (Tensor) - 第一个输入向量。如果不是一维的, 则将其展平。
- **b** (Tensor) - 第二个输入向量。如果不是一维的, 则将其展平。

返回:

Tensor 或标量, $out[i, j] = a[i] * b[j]$ 。

异常:

- **TypeError** - 如果输入不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> a = np.full(7, 2).astype('float32')
>>> b = np.full(4, 3).astype('float32')
>>> output = np.outer(a, b)
>>> print(output)
[[6. 6. 6. 6.]
 [6. 6. 6. 6.]
```

(下页继续)

(续上页)

```
[6. 6. 6. 6.]
[6. 6. 6. 6.]
[6. 6. 6. 6.]
[6. 6. 6. 6.]
[6. 6. 6. 6.]
```

15.4.84 mindspore.numpy.polyadd

`mindspore.numpy.polyadd(a1, a2)`

找到两个多项式的和。返回两个输入多项式的和所得的多项式。

说明：目前不支持 NumPy 对象 `poly1d`。

参数：

- **a1** (Union[int, float, list, tuple, Tensor]) - 输入多项式。
- **a2** (Union[int, float, list, tuple, Tensor]) - 输入多项式。

返回：

Tensor，输入之和。

异常：

- **ValueError** - 如果输入数组的维数超过 1。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.polyadd([1, 2], [9, 5, 4]))
[9 6 6]
```

15.4.85 mindspore.numpy.polyder

`mindspore.numpy.polyder(p, m=1)`

返回多项式指定阶数的导数。

说明：目前不支持 NumPy 对象 `poly1d`。

参数：

- **p** (Union[int, float, bool, list, tuple, Tensor]) - 要求导数的多项式。一个代表多项式系数的序列。
- **m** (int, 可选) - 默认值：1，导数的阶数。

返回：

Tensor，表示导数的新多项式。

异常:

- **ValueError** - 如果 p 的维数超过 1。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.polyder([1, 1, 1, 1]))
[3 2 1]
```

15.4.86 mindspore.numpy.polyint

`mindspore.numpy.polyint(p, m=1, k=None)`

返回多项式的一个反导数 (不定积分)。

说明: 目前不支持 NumPy 对象 `poly1d`。

参数:

- **p** (`Union[int, float, bool, list, tuple, Tensor]`) - 要积分的多项式。一个表示多项式系数的序列。
- **m** (`int`, 可选) - 默认值: 1, 反导数的阶数。
- **k** (`Union[int, list[int]]`, 可选) - 积分常数。按积分顺序给出: 对应最高阶项的常数排在最前面。如果为 **None**(默认), 所有常数都设为零。如果 $m = 1$, 可以给定为标量而非列表。

返回:

`Tensor`, 表示反导数的新多项式。

异常:

- **ValueError** - 如果 p 的维数超过 1。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.polyint([1, 1, 1]))
[0.33333334 0.5      1.      0.      ]
```

15.4.87 mindspore.numpy.polymul

`mindspore.numpy.polymul(a1, a2)`

求两个多项式的积。

说明: 目前不支持 NumPy 对象 `poly1d`。

参数:

- **a1** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入多项式。
- **a2** (`Union[int, float, bool, list, tuple, Tensor]`) - 输入多项式。

返回:

`Tensor`, 表示导数的新多项式。

异常:

- **ValueError** - 如果输入数组的维数超过 1。

支持平台:

GPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.polymul([3, 1, 2], [2, 5]))
[ 6 17  9 10]
```

15.4.88 mindspore.numpy.polysub

`mindspore.numpy.polysub(a1, a2)`

两个多项式的差 (减法)。给定两个多项式 *a1* 和 *a2*, 返回 $a1 - a2$ 。

说明: 目前不支持 NumPy 对象 `poly1d`。

参数:

- **a1** (`Union[int, float, list, tuple, Tensor]`) - 被减多项式。
- **a2** (`Union[int, float, list, tuple, Tensor]`) - 减数多项式。

返回:

`Tensor`, 输入的差。

异常:

- **ValueError** - 如果输入数组的维数超过 1。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.polysub([2, 10, -2], [3, 10, -4]))
[-1  0  2]
```

15.4.89 mindspore.numpy.polyval

`mindspore.numpy.polyval(p, x)`

在特定值处求多项式的值。如果 p 的长度为 N ，该函数返回值为：

$p[0]x^{(N-1)} + p[1]x^{(N-2)} + \dots + p[N-2]x + p[N-1]$ 如果 x 是一个序列，那么返回 $p(x)$ 对应 x 中每个元素的值。如果 x 是另一个多项式，则返回复合多项式 $p(x(t))$ 。

说明：目前不支持 NumPy 对象 `poly1d`。

参数：

- **p** (Union[int, float, bool, list, tuple, Tensor]) - 从最高次项到常数项的一维多项式系数数组 (包括等于零的系数)。
- **x** (Union[int, float, bool, list, tuple, Tensor]) - 求对应 p 值的数或数组。

返回：

Tensor。

异常：

- **ValueError** - 如果 p 的维数超过 1。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.polyval([3., 0., 1.], 5.))
76.0
```

15.4.90 mindspore.numpy.positive

`mindspore.numpy.positive(a, dtype=None)`

数值取正，逐元素计算。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **a** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.asarray([1, -1]).astype('float32')
>>> output = np.positive(a)
>>> print(output)
[1. -1.]
```

15.4.91 mindspore.numpy.power

`mindspore.numpy.power(x1, x2, dtype=None)`

第一个数组的元素以第二个数组的元素为指数逐元素求幂。

将 *x1* 中的每个基数以 *x2* 中对应位置的元素为指数取幂。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 GPU 上，支持的数据类型有 `np.float16` 和 `np.float32`。

参数：

- **x1** (Tensor) - 基数。
- **x2** (Tensor) - 指数。
- **dtype** (`mindspore.dtype`, 可选) - 默认值：None。覆盖输出 Tensor 的 `dtype`。

返回：

Tensor 或标量，*x1* 中的基数按 *x2* 中的指数计算幂的结果。如果 *x1* 和 *x2* 都是标量，返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.full((3, 2), [1, 2]).astype('float32')
>>> x2 = np.full((3, 2), [3, 4]).astype('float32')
>>> output = np.power(x1, x2)
>>> print(output)
[[ 1. 16.]
 [ 1. 16.]
 [ 1. 16.]]
```

15.4.92 mindspore.numpy.promote_types

`mindspore.numpy.promote_types(type1, type2)`

返回 `type1` 与 `type2` 都可以安全转换的最小位数和最小标量类型的数据类型。

说明：类型提升规则与原版 NumPy 略有不同，更类似于 `jax`，因为其更倾向于 32 位而非 64 位数据类型。

参数：

- **type1** (Union[mindspore.dtype, str]) - 第一个数据类型。
- **type2** (Union[mindspore.dtype, str]) - 第二个数据类型。

返回：

提升后的数据类型。

异常：

- **TypeError** - 如果输入不是有效的 `mindspore.dtype` 输入。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.promote_types(np.float32, np.float64)
>>> print(output)
Float64
```

15.4.93 mindspore.numpy.ptp

`mindspore.numpy.ptp(x, axis=None, keepdims=False)`

沿某个轴的值范围 (最大值 - 最小值)。该函数名称来自于 “peak to peak” 的首字母缩写。

说明：不支持 NumPy 参数 `dtype` 和 `out` 。

参数：

- **x** (Tensor) - 输入 Tensor。
- **axis** (Union[None, int, tuple(int)]) - 计算范围所沿的单个或多个轴。默认在计算展平的数组上计算。默认值：None。
- **keepdims** (bool) - 如果设置为 `True`，减少的轴在结果中作为大小为 1 的维度保留。若使用此选项，结果会广播到和输入 Tensor 同一个维度数。如果传入默认值，则 `keepdims` 参数不会传递到 Tensor 子类的 `ptp` 方法中，而任何非默认值将会传递。默认值：False。

返回：

Tensor。

异常：

- **TypeError** - 如果输入类型不是上述指定类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.array([[4.0, 9.0, 2.0, 10.0], [6.0, 9.0, 7.0, 12.0]])
>>> print(np.ptp(x, axis=1))
[8. 6.]
>>> print(np.ptp(x, axis=0))
[2. 0. 5. 2.]
```

15.4.94 mindspore.numpy.rad2deg

`mindspore.numpy.rad2deg(x, dtype=None)`

将角从弧度制转换为角度制。

参数：

- **x** (Tensor) - 弧度制角。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor，对应的角度制角。如果 *x* 是 Tensor 标量，则返回 Tensor 标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.asarray([1, 2, 3, -4, -5])
>>> output = np.rad2deg(x)
>>> print(output)
[ 57.295776  114.59155   171.88733 -229.1831  -286.47888 ]
```

15.4.95 mindspore.numpy.radians

`mindspore.numpy.radians(x, dtype=None)`

将角从角度制转换为弧度制。

参数：

- **x** (Tensor) - 角度制角。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor，对应的弧度制角。如果 *x* 是 Tensor 标量，则返回 Tensor 标量。

异常：

- **TypeError** - 如果 *x* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.asarray([1, 2, 3, -4, -5])
>>> output = np.radians(x)
>>> print(output)
[ 0.01745329  0.03490658  0.05235988 -0.06981317 -0.08726647]
```

15.4.96 mindspore.numpy.ravel_multi_index

`mindspore.numpy.ravel_multi_index` (*multi_index*, *dims*, *mode*='clip', *order*='C')

将元素为索引数组的 `tuple` 转换为展平的索引数组, 并对多重索引应用边界模式。

说明: 不支持 *raise* 模式。默认模式为 *clip*。

参数:

- **multi_index** (类数组的 `tuple`) - 一个元素为整数数组的 `tuple`, 每一维包含一个数组。
- **dims** (`Union[int, tuple(int)]`) - *multi_index* 的索引将应用到的数组的 `shape`。
- **mode** (`{wrap, clip}`, 可选) - 指定如何处理越界索引。默认值: '*clip*'。'*wrap*': 取越界索引除以轴长的余数。'*clip*': 裁剪到范围内。在 '*clip*' 模式下, 取余后的负索引将裁剪至 0。
- **order** (`{C, F}`, 可选) - 确定多重索引是以行优先 (C 风格) 还是列优先 (Fortran 风格) 顺序查看。

返回:

展平的索引数组。表示 *dims* 维的展平的索引值。

异常:

- **TypeError** - 如果 *multi_index* 或 *dims* 不能转换为 `Tensor`, 或者 *dims* 不是整数值的序列。
- **ValueError** - 如果 *multi_index* 和 *dims* 的长度不相等。

支持平台:

GPU

样例:

```
>>> import mindspore.numpy as np
>>> arr = np.array([[3, 6, 6], [4, 5, 1]])
>>> output = np.ravel_multi_index(arr, (7, 6))
>>> print(output)
[22. 41. 37.]
>>> output = np.ravel_multi_index((3, 1, 4, 1), (6, 7, 8, 9))
>>> print(output)
1621.0
```

15.4.97 mindspore.numpy.reciprocal

`mindspore.numpy.reciprocal(x, dtype=None)`

逐元素返回入参的倒数。

计算 $1/x$ 。

说明: 不支持 NumPy 参数 *casting*、*order*、*subok*、*signature*、*extobj*。当使用 *where* 时, *out* 必须具有 Tensor 值。*out* 不支持存储结果, 但可以与 *where* 结合使用, 在 *where* 设置为 *False* 的索引处设定值。

参数:

- **x** (Tensor) - 输入数组。对于绝对值大于 1 的整数参数, 由于 Python 处理整数除法的方式, 结果总是为零。对于整数零, 结果溢出。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 如果 *x* 是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.arange(1, 7).reshape(2, 3).astype('float32')
>>> output = np.reciprocal(x)
>>> print(output)
[[1.          0.5          0.33333334]
 [0.25        0.2         0.16666667]]
```

15.4.98 mindspore.numpy.remainder

`mindspore.numpy.remainder(x1, x2, dtype=None)`

逐元素返回除法余数。

计算与 *floor_divide* 函数互补的余数。相当于 Python 的取模运算符 $x1 \% x2$, 并且具有与除数 *x2* 相同的符号。与 *np.remainder* 等效的 MATLAB 函数是 *mod*。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x1** (Tensor) - 输入数组。
- **x2** (Tensor) - 输入数组。
- **dtype** (mindspore.dtype, 可选) - 默认值: *None*。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 逐元素计算 *floor_divide*(*x1*, *x2*) 得到的余数。如果 *x1* 和 *x2* 都是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.remainder(np.array([4, 7]), np.array([2, 3]))
>>> print(output)
[0 1]
>>> output = np.remainder(np.arange(7), np.array(5))
>>> print(output)
[0 1 2 3 4 0 1]
```

15.4.99 mindspore.numpy.result_type

`mindspore.numpy.result_type(*arrays_and_dtypes)`

返回对入参使用类型提升规则所得的类型。

说明: 提升规则与原版 NumPy 略有不同, 更类似于 `jax`, 因为更倾向于 32 位而不是 64 位数据类型。复数类型不支持。

参数:

- ***arrays_and_dtypes** (Union[int, float, bool, list, tuple, Tensor, mindspore.dtype, str]) - 需要得到类型结果的操作数。

返回:

`mindspore.dtype`, 结果类型。

异常:

- **TypeError** - 如果输入不是有效的数据类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> print(np.result_type('i2', np.float32, True))
Float32
```

15.4.100 mindspore.numpy rint

`mindspore.numpy.rint(x, dtype=None)`

将数组的元素四舍五入到最近的整数。

说明: 不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数:

- **x** (Union[float, list, tuple, Tensor]) - 任意维度的输入 Tensor。

- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 dtype。

返回:

输出 Tensor 的 shape 和类型与 x 相同。如果 x 是标量, 则返回标量。

异常:

- **TypeError** - 如果 x 无法转换为 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([-1.7, -1.5, 0.2, 1.5, 1.7, 2.0])
>>> print(np rint(x))
[-2. -2. 0. 2. 2. 2.]
```

15.4.101 mindspore.numpy.searchsorted

`mindspore.numpy.searchsorted(a, v, side='left', sorter=None)`

找到每个元素的插入索引, 使得插入后的数组保持原有升降序。在排序过的数组 a 中找到索引, 使得如果在这些索引之前插入 v 中的相应元素, a 的顺序能够保持。

参数:

- **a** (Union[list, tuple, Tensor]) - 1-D 输入数组。如果 *sorter* 为 `None`, 则 a 必须按升序排序, 否则 *sorter* 必须是一个排序索引数组。
- **v** (Union[int, float, bool, list, tuple, Tensor]) - 要插入 a 的值。
- **side** ('left', 'right', 可选) - 如果为 'left' (默认值), 则返回第一个合适位置的索引。如果为 'right', 则返回最后一个合适位置的索引。如果没有合适的索引, 则返回 0 或 N (其中 N 为 a 的长度)。
- **sorter** (Union[int, float, bool, list, tuple, Tensor]) - 可选择输入一个一维整数索引数组, 用于将数组 a 排序为升序。通常是 *argsort* 的结果。默认值: `None`。

返回:

Tensor, 与 v shape 相同的元素为插入点的数组。

异常:

- **ValueError** - 如果 *side* 或 *sorter* 参数无效。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import numpy as np
>>> print(np.searchsorted([1,2,3,4,5], 3))
2
>>> print(np.searchsorted([1,2,3,4,5], 3, side='right'))
3
>>> print(np.searchsorted([1,2,3,4,5], [-10, 10, 2, 3]))
[0 5 1 2]
```

15.4.102 mindspore.numpy.sign

`mindspore.numpy.sign(x, dtype=None)`

逐元素返回数的符号。

当 $x < 0$ 时, `sign` 函数返回 -1, 当 $x == 0$ 时, 返回 0; 当 $x > 0$ 时, 返回 1。对于 `nan` 输入, 返回 `nan`。

说明: 不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。目前不支持复数输入。在 Ascend 上, 不支持整数输入。

参数:

- **x** (Union[int, float, list, tuple, Tensor]) - 输入值。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

x 的符号, 当 x 是标量时, 可以是一个 `tensor` 或者标量。

异常:

- **TypeError** - 如果输入的 `dtype` 不在给定类型中或输入不能转换为 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> output = np.sign(np.array([-1., 0., 1., 1.2]))
>>> print(output)
[-1.  0.  1.  1.]
```

15.4.103 mindspore.numpy.sin

`mindspore.numpy.sin(x, dtype=None)`

逐元素计算三角正弦函数。

说明: 不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: `None`。覆盖输出 Tensor 的 `dtype`。

返回:

Tensor 或标量。如果 x 是标量, 则返回标量。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.array([-5, -1, 0, 2, 4, 100]).astype('float32')
>>> output = np.sin(x)
>>> print(output)
[ 0.9589243 -0.84147096  0.    0.9092974 -0.7568025 -0.50636566]
```

15.4.104 mindspore.numpy.sinh

`mindspore.numpy.sinh(x, dtype=None)`

逐元素计算双曲正弦函数。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数:

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量。如果 *x* 是标量, 则返回标量。

支持平台:

Ascend CPU

样例:

```
>>> import mindspore.numpy as np
>>> x = np.arange(5).astype('float32')
>>> print(np.sinh(x))
[ 0.    1.1752012  3.6268604 10.017875 27.289917 ]
```

15.4.105 mindspore.numpy.sqrt

`mindspore.numpy.sqrt(x, dtype=None)`

逐元素返回数组的非负平方根。

说明: 不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。在 GPU 上, 支持的 dtype 为 `np.float16` 和 `np.float32`。

参数:

- **x** (Tensor) - 需计算平方根的值。
- **dtype** (mindspore.dtype, 可选) - 默认值: None。覆盖输出 Tensor 的 dtype。

返回:

Tensor 或标量, 与 *x* shape 相同的数组, 包含 *x* 中每个元素的正平方根。对于负元素, 返回 nan。如果 *x* 是标量, 则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.arange(6).reshape(2, 3).astype('float32')
>>> x_squared = np.square(x)
>>> output = np.sqrt(x_squared)
>>> print(output)
[[ 0.  1.  2.]
 [ 3.  4.  5.]]
```

15.4.106 mindspore.numpy.square

`mindspore.numpy.square(x, dtype=None)`

逐元素返回输入的平方。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。在 GPU 上，支持的 dtype 为 `np.float16` 和 `np.float32`。

参数：

- **x** (Tensor) - 输入数据。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量，逐元素计算 $x \times x$ ，具有与 x 相同的 shape 和 dtype。如果 x 是标量，则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.square(np.arange(6).reshape(2, 3).astype('float32'))
>>> print(x)
[[ 0.  1.  4.]
 [ 9. 16. 25.]]
```

15.4.107 mindspore.numpy.std

`mindspore.numpy.std(x, axis=None, ddof=0, keepdims=False)`

沿指定轴计算标准差。标准差是平方偏差平均值的平方根，即 $std = \sqrt{\text{mean}(\text{abs}(x - x.\text{mean}())^2)}$ 。

返回标准差，默认计算展平数组的标准差，否则在指定轴上计算。

说明：不支持 NumPy 参数 `dtype`、`out`、`where`。

参数：

- **x** (Tensor) - 进行计算的 Tensor。
- **axis** (Union[None, int, tuple(int)]) - 计算标准差所沿的单个或多个轴。默认值：None。如果为 None，计算展平数组的标准差。
- **ddof** (int) - 自由度偏差。计算中使用的除数为 $N - ddof$ ，其中 N 表示元素数量。默认值：0。
- **keepdims** - 如果设置为 True，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和输入 Tensor 同一个维度数。如果传入默认值，则 *keepdims* 参数不会传递到 Tensor 子类的 std 方法中，而任何非默认值将会传递。如果子类中方法未实现 *keepdims*，则会引发异常。默认值：False。

返回：

标准差 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input_x = np.array([1., 2., 3., 4.])
>>> output = np.std(input_x)
>>> print(output)
1.118034
```

15.4.108 mindspore.numpy.subtract

`mindspore.numpy.subtract(x1, x2, dtype=None)`

逐元素减去给定参数。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x1** (Tensor) - 被减数输入。
- **x2** (Tensor) - 减数输入。
- **dtype** (*mindspore.dtype*, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量，*x1* 和 *x2* 的逐元素差。如果 *x1* 和 *x2* 都是标量，则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.full((3, 2), [1, 2])
>>> x2 = np.full((3, 2), [3, 4])
>>> output = np.subtract(x1, x2)
>>> print(output)
```

(下页继续)

(续上页)

```
[[ -2  -2]
 [ -2  -2]
 [ -2  -2]]
```

15.4.109 mindspore.numpy.sum

`mindspore.numpy.sum(a, axis=None, dtype=None, keepdims=False, initial=None)`

返回指定轴上数组元素的总和。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **a** (`Union[int, float, bool, list, tuple, Tensor]`) - 要计算和的元素。
- **axis** (`Union[None, int, tuple(int)]`, 可选) - 执行求和操作的单个或多个轴。默认值：`None`。如果为 `None`，则求输入数组中所有元素的和。如果 `axis` 为负数，则从最后一个轴到第一个轴计数。如果 `axis` 是元素为整数的 `tuple`，则对 `tuple` 中指定的所有轴进行求和，而非前述的单个轴或所有轴。
- **dtype** (`mindspore.dtype`, 可选) - 默认值：`None`。覆盖输出 `Tensor` 的 `dtype`。
- **keepdims** (`bool`, 可选) - 如果设置为 `True`，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和输入 `Tensor` 同一个维度数。如果传入默认值，则 `keepdims` 参数不会传递到 `ndarray` 子类的 `std` 方法中，而任何非默认值将会传递。如果子类中方法未实现 `keepdims`，则会引发异常。默认值：`False`。
- **initial** (`scalar`, 可选) - 求和的初始值，如果为 `None`，不会添加额外的初始值，从第一个元素的值开始累加。默认值：`None`。

返回：

`Tensor`。具有与 `a` 相同 `shape` 的数组，去除了指定的轴。如果 `a` 是 0-d 数组，或者 `axis` 为 `None`，则返回标量。如果指定了输出数组，则返回对 `out` 的引用。

异常：

- **TypeError** - 如果 `input` 不是类数组或 `axis` 不是 `int` 或元素为 `int` 的 `tuple`，或如果 `keepdims` 不是整数或 `initial` 不是标量。
- **ValueError** - 如果任何轴超出范围或存在重复的轴。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> print(np.sum([0.5, 1.5]))
2.0
>>> x = np.arange(10).reshape(2, 5).astype('float32')
>>> print(np.sum(x, axis=1))
[10. 35.]
```

15.4.110 mindspore.numpy.tan

`mindspore.numpy.tan(x, dtype=None)`

逐元素计算正切值。

等价于 $\text{np.sin}(x)/\text{np.cos}(x)$ ，逐元素计算。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。如果 *x* 是标量，则返回标量。

异常：

- **TypeError** - 如果输入不是 Tensor 或 Tensor 的 dtype 为 `mindspore.float64`。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.array([-5, -1, 0, 2, 4, 100]).astype('float32')
>>> print(np.tan(x))
[ 3.380515  -1.5574077  0.          -2.1850398  1.1578213 -0.58721393]
```

15.4.111 mindspore.numpy.tanh

`mindspore.numpy.tanh(x, dtype=None)`

逐元素计算双曲正切值。

说明：不支持 NumPy 参数 *out*、*where*、*casting*、*order*、*subok*、*signature*、*extobj*。

参数：

- **x** (Tensor) - 输入 Tensor。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量。如果 *x* 是标量，则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x = np.arange(5).astype('float32')
>>> print(np.tanh(x))
[0.          0.7615942  0.9640276  0.9950548  0.9993293]
```

15.4.112 mindspore.numpy.tensordot

`mindspore.numpy.tensordot(a, b, axes=2)`

沿指定轴计算 Tensor 的点积。

给定两个 Tensor a 和 b ，以及一个包含两个类数组对象的类数组对象 (a_axes , b_axes)，对 a 和 b 的元素 (分量) 在 a_axes 和 b_axes 指定的轴上求逐元素乘积之和。第三个参数可以是一个非负整数标量 N ，如果是 N ，则对 a 的最后 N 个维度和 b 的前 N 个维度进行求和。常见的三种用例是：

- $axes = 0$: Tensor 积
- $axes = 1$: Tensor 点积
- $axes = 2$: (默认)Tensor 双重缩并

当 $axes$ 为整数时，计算的顺序是：首先是 a 的第 $-N$ 轴和 b 的第 0 轴，最后是 a 的 -1 轴和 b 的第 N 轴。对于多个轴进行求和且它们不是 $a(b)$ 的最后 (b 的第一个) 轴时，参数 $axes$ 应由两个相同长度的序列组成，第一个序列应用于 a ，第二个序列应用于 b ，依此类推。结果的 `shape` 由第一个 Tensor 的未缩并轴以及第二个 Tensor 的未缩并轴组成。

说明：在 CPU 上，支持的 dtype 是 `np.float16` 和 `np.float32`。在 GPU 上，支持的 dtype 是 `np.float16` 和 `np.float32`。

参数：

- **a** (Tensor) - 需计算点积的 Tensor。
- **b** (Tensor) - 需计算点积的 Tensor。
- **axes** (int 或 int 的序列) -

类 int 值：如果是整数 N ，则按顺序对 a 的最后 N 个轴和 b 的前 N 个轴进行求和。对应的轴的大小必须匹配。

int 的序列：一个包含要进行求和的轴的列表，第一个序列应用于 a ，第二个序列应用于 b 。两个类数组元素必须具有相同长度。

返回：

Tensor 或元素为 Tensor 的 list，输入 Tensor 的点积。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.ones((3, 4, 5))
>>> b = np.ones((4, 3, 2))
>>> output = np.tensordot(a, b, axes=([1,0],[0,1]))
>>> print(output.shape)
(5, 2)
```

15.4.113 mindspore.numpy.trapz

`mindspore.numpy.trapz(y, x=None, dx=1.0, axis=-1)`

使用复合梯形规则沿给定轴进行积分。

沿给定轴对 $y(x)$ 进行积分。

参数：

- **y** (Tensor) - 输入要积分的数组。
- **x** (Union[int, float, bool, list, tuple, Tensor], 可选) - 与 y 值对应的样本点。如果 x 为 `None`，则设样本点之间的间隔为 dx 。默认值：`None`。
- **dx** (scalar, 可选) - 当 x 为 `None` 时，样本点之间的间隔。默认值：`1.0`。
- **axis** (int, 可选) - 积分所沿的轴。默认值：`-1`。

返回：

元素为 `float` 的 Tensor，使用梯形规则近似的定积分。

异常：

- **ValueError** - 如果 $axis$ 超出范围 $[-y.ndim, y.ndim)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> a = np.arange(6).reshape(2, 3)
>>> output = np.trapz(a, x=[-2, 1, 2], axis=1)
>>> print(output)
[ 3. 15.]
>>> output = np.trapz(a, dx=3, axis=0)
>>> print(output)
[ 4.5  7.5 10.5]
```

15.4.114 mindspore.numpy.true_divide

`mindspore.numpy.true_divide(x1, x2, dtype=None)`

返回输入的真除法，逐元素计算。

该函数与 Python 默认的“向下取整除”不同，它返回真除法的结果。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **x1** (Tensor) - 被除数。
- **x2** (Tensor) - 除数。
- **dtype** (mindspore.dtype, 可选) - 默认值：`None`。覆盖输出 Tensor 的 `dtype`。

返回：

Tensor 或标量，如果 $x1$ 和 $x2$ 都是标量，则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> x1 = np.full((3, 2), [1, 2])
>>> x2 = np.full((3, 2), [3, 4])
>>> output = np.true_divide(x1, x2)
>>> print(output)
[[0.33333334 0.5      ]
 [0.33333334 0.5      ]
 [0.33333334 0.5      ]]
```

15.4.115 mindspore.numpy.trunc

`mindspore.numpy.trunc(x, dtype=None)`

逐元素返回输入的截断值。

标量 x 的截断值是距离 x 最近且趋近 0 的整数 i ，即 x 的小数部分被舍弃。

说明：不支持 NumPy 参数 `out`、`where`、`casting`、`order`、`subok`、`signature`、`extobj`。

参数：

- **x** (Tensor) - 输入数据。
- **dtype** (mindspore.dtype, 可选) - 默认值：None。覆盖输出 Tensor 的 dtype。

返回：

Tensor 或标量， x 中每个元素的截断值。如果 x 是标量，则返回标量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> output = np.trunc(np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0]))
>>> print(output)
[-1. -1. -0.  0.  1.  1.  2.]
```

15.4.116 mindspore.numpy.unwrap

`mindspore.numpy.unwrap(p, discont=3.141592653589793, axis=-1)`

通过加或减 2π ，改变数组相邻元素的差值实现解卷绕。对于弧度数组 p ，将其大于 $discont$ 的相邻元素之差的绝对值加或减 2π ，直到相邻元素之差的绝对值小于 $discont$ ，沿指定轴执行。

说明：对于相邻元素之差的绝对值与 π 的差值很小时，由于舍入误差的差异，解卷绕可能会与 NumPy 不同。

参数：

- **p** (Union[int, float, bool, list, tuple, Tensor]) - 输入数组。
- **discont** (float, 可选) - 值之间的最大不连续性，默认值：pi。
- **axis** (int, 可选) - 解卷绕操作所沿的轴。默认值：-1。

返回：

Tensor。

异常：

- **ValueError** - 如果轴超出范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> phase = np.add(np.linspace(0, np.pi, num=5), [0, 0, 0, np.pi, np.pi])
>>> print(phase)
[0.          0.7853982  1.5707964  5.4977875  6.2831855]
>>> print(np.unwrap(phase))
[ 0.0000000e+00  7.8539819e-01  1.5707964e+00 -7.8539848e-01 -4.7683716e-07]
```

15.4.117 mindspore.numpy.var

`mindspore.numpy.var(x, axis=None, ddof=0, keepdims=False)`

计算沿指定轴的方差。方差是各值与平均值的平方差的平均数。

默认情况下，返回计算展平数组的方差，否则沿指定轴计算方差。

说明：不支持 NumPy 参数 *dtype*、*out*、*where*。

参数：

- **x** (Tensor) - 要进行计算的 Tensor。
- **axis** (Union[None, int, tuple(int)]) - 计算方差所沿的单个或多个轴。取默认值 *None* 时，在展平数组上计算。
- **ddof** (int) - 自由度偏差。默认值：0。计算中使用的除数是 $N - ddof$ 。其中 N 代表元素的数量。
- **keepdims** (bool) - 如果设置为 *True*，减少的轴在结果中保留为大小为 1 的维度。若使用此选项，结果会广播到和输入 Tensor 同一个维度数。如果传入默认值，则 *keepdims* 参数不会传递到 Tensor 子类的 *var* 方法中，而任何非默认值将会传递。如果子类中方法未实现 *keepdims*，则会引发异常。默认值：False。

返回：

Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore.numpy as np
>>> input_x = np.array([1., 2., 3., 4.])
>>> output = np.var(input_x)
>>> print(output)
1.25
```

15.5 MindSpore Numpy 与 MindSpore 特性结合

`mindspore.numpy` 能够充分利用 MindSpore 的强大功能, 实现算子的自动微分, 并使用图模式加速运算, 帮助用户快速构建高效的模型。同时, MindSpore 还支持多种后端设备, 包括 Ascend、GPU 和 CPU 等, 用户可以根据自己的需求灵活设置。以下提供了几种常用方法:

- `jit` 装饰器: 将代码包裹进图模式, 用于提高代码运行效率。
- `GradOperation`: 用于自动求导。
- `mindspore.set_context`: 用于设置运行模式和后端设备等。
- `mindspore.nn.Cell`: 用于建立深度学习模型。

使用示例如下:

- `jit` 装饰器使用示例

首先, 以神经网络里经常使用到的矩阵乘与矩阵加算子为例:

```
import mindspore.numpy as np

x = np.arange(8).reshape(2, 4).astype('float32')
w1 = np.ones((4, 8))
b1 = np.zeros((8,))
w2 = np.ones((8, 16))
b2 = np.zeros((16,))
w3 = np.ones((16, 4))
b3 = np.zeros((4,))

def forward(x, w1, b1, w2, b2, w3, b3):
    x = np.dot(x, w1) + b1
    x = np.dot(x, w2) + b2
    x = np.dot(x, w3) + b3
    return x

print(forward(x, w1, b1, w2, b2, w3, b3))
```

运行结果如下:

```
[[ 768.  768.  768.  768.]
 [2816. 2816. 2816. 2816.]]
```

对上述示例, 我们可以借助 `jit` 装饰器将所有算子编译到一张静态图里以加快运行效率, 示例如下:

```
from mindspore import jit

forward_compiled = jit(forward)
print(forward_compiled(x, w1, b1, w2, b2, w3, b3))
```

运行结果如下:

```
[[ 768.  768.  768.  768.]
 [2816. 2816. 2816. 2816.]]
```

说明：目前静态图不支持在 Python 交互式模式下运行，并且有部分语法限制。

• GradOperation 使用示例

GradOperation 可以实现自动求导。以下示例可以实现对上述没有用 *jit* 修饰的 *forward* 函数定义的计算求导。

```
from mindspore import ops

grad_all = ops.GradOperation(get_all=True)
print(grad_all(forward)(x, w1, b1, w2, b2, w3, b3))
```

运行结果如下：

```
(Tensor(shape=[2, 4], dtype=Float32, value=
 [[ 5.12000000e+02,  5.12000000e+02,  5.12000000e+02,  5.12000000e+02],
 [ 5.12000000e+02,  5.12000000e+02,  5.12000000e+02,  5.12000000e+02]]),
 Tensor(shape=[4, 8], dtype=Float32, value=
 [[ 2.56000000e+02,  2.56000000e+02,  2.56000000e+02 ... 2.56000000e+02,  2.56000000e+02, ↵
 ↪2.56000000e+02],
 [ 3.84000000e+02,  3.84000000e+02,  3.84000000e+02 ... 3.84000000e+02,  3.84000000e+02, ↵
 ↪3.84000000e+02],
 [ 5.12000000e+02,  5.12000000e+02,  5.12000000e+02 ... 5.12000000e+02,  5.12000000e+02, ↵
 ↪5.12000000e+02],
 [ 6.40000000e+02,  6.40000000e+02,  6.40000000e+02 ... 6.40000000e+02,  6.40000000e+02, ↵
 ↪6.40000000e+02]]),
 ...
 Tensor(shape=[4], dtype=Float32, value= [ 2.00000000e+00,  2.00000000e+00,  2.00000000e+00, ↵
 ↪2.00000000e+00]))
```

如果要对 *jit* 修饰的 *forward* 计算求导，需要提前使用 *set_context* 设置运算模式为图模式，示例如下：

```
from mindspore import jit, set_context, GRAPH_MODE, ops

set_context(mode=GRAPH_MODE)
grad_all = ops.GradOperation(get_all=True)
print(grad_all(jit(forward))(x, w1, b1, w2, b2, w3, b3))
```

运行结果如下：

```
(Tensor(shape=[2, 4], dtype=Float32, value=
 [[ 5.12000000e+02,  5.12000000e+02,  5.12000000e+02,  5.12000000e+02],
 [ 5.12000000e+02,  5.12000000e+02,  5.12000000e+02,  5.12000000e+02]]),
 Tensor(shape=[4, 8], dtype=Float32, value=
 [[ 2.56000000e+02,  2.56000000e+02,  2.56000000e+02 ... 2.56000000e+02,  2.56000000e+02, ↵
 ↪2.56000000e+02],
 [ 3.84000000e+02,  3.84000000e+02,  3.84000000e+02 ... 3.84000000e+02,  3.84000000e+02, ↵
 ↪3.84000000e+02],
 [ 5.12000000e+02,  5.12000000e+02,  5.12000000e+02 ... 5.12000000e+02,  5.12000000e+02, ↵
 ↪5.12000000e+02],
 [ 6.40000000e+02,  6.40000000e+02,  6.40000000e+02 ... 6.40000000e+02,  6.40000000e+02, ↵
 ↪6.40000000e+02]]),
 ...
 Tensor(shape=[4], dtype=Float32, value= [ 2.00000000e+00,  2.00000000e+00,  2.00000000e+00, ↵
 ↪2.00000000e+00]))
```

(下页继续)

(续上页)

更多细节可参考 [API GradOperation](#)。

- `mindspore.set_context` 使用示例

MindSpore 支持多后端运算，可以通过 `mindspore.set_context` 进行设置。`mindspore.numpy` 的多数算子可以使用图模式或者 PyNative 模式运行，也可以运行在 CPU，CPU 或者 Ascend 等多种后端设备上。

```
from mindspore import set_context, GRAPH_MODE, PYNATIVE_MODE

# Execution in static graph mode
set_context(mode=GRAPH_MODE)

# Execution in PyNative mode
set_context(mode=PYNATIVE_MODE)

# Execution on CPU backend
set_context(device_target="CPU")

# Execution on GPU backend
set_context(device_target="GPU")

# Execution on Ascend backend
set_context(device_target="Ascend")
...
```

更多细节可参考 [API mindspore.set_context](#)。

- `mindspore.numpy` 使用示例

这里提供一个使用 `mindspore.numpy` 构建网络模型的示例。

`mindspore.numpy` 接口可以定义在 `nn.Cell` 代码块内进行网络的构建，示例如下：

```
import mindspore.numpy as np
from mindspore import set_context, GRAPH_MODE
from mindspore.nn import Cell

set_context(mode=GRAPH_MODE)

x = np.arange(8).reshape(2, 4).astype('float32')
w1 = np.ones((4, 8))
b1 = np.zeros((8,))
w2 = np.ones((8, 16))
b2 = np.zeros((16,))
w3 = np.ones((16, 4))
b3 = np.zeros((4,))

class NeuralNetwork(Cell):
    def construct(self, x, w1, b1, w2, b2, w3, b3):
        x = np.dot(x, w1) + b1
        x = np.dot(x, w2) + b2
        x = np.dot(x, w3) + b3
        return x

net = NeuralNetwork()

print(net(x, w1, b1, w2, b2, w3, b3))
```

运行结果如下：

```
[[ 768.  768.  768.  768.]  
 [2816. 2816. 2816. 2816.]]
```

MindSpore 中类似 Scipy 的接口。

警告：这些是实验性 API，可能会被更改或删除。只支持 Linux。

16.1 mindspore.scipy.linalg

线性代数子模块。

接口名	概述	支持平台
<code>mindspore.scipy.linalg.block_diag</code>	根据输入的数组创建块对角矩阵。	GPU CPU
<code>mindspore.scipy.linalg.cho_factor</code>	计算矩阵的 cholesky 分解，用于 <code>mindspore.scipy.linalg.cho_solve()</code> 。	GPU CPU
<code>mindspore.scipy.linalg.cho_solve</code>	给定 A 的 cholesky 分解，求解线性方程组。	GPU CPU
<code>mindspore.scipy.linalg.cholesky</code>	计算矩阵的 cholesky 分解。	GPU CPU
<code>mindspore.scipy.linalg.eigh</code>	求解复 Hermitian 矩阵或实对称矩阵的标准或广义特征值问题。	GPU CPU
<code>mindspore.scipy.linalg.inv</code>	计算矩阵的逆。	GPU CPU
<code>mindspore.scipy.linalg.lu</code>	计算通用矩阵的 LU 分解。	GPU CPU
<code>mindspore.scipy.linalg.lu_factor</code>	计算方阵的 LU 分解，其输出可以直接作为 <code>lu_solve</code> 的输入。	GPU CPU
<code>mindspore.scipy.linalg.solve_triangular</code>	假设 a 是一个三角矩阵，求等式系统 $ax = b$ 的解 x 。	Ascend CPU
<code>mindspore.scipy.linalg.lstsq</code>	求解计算线性等式系统 $AX = B$ 的最小二乘问题。	Ascend CPU

16.1.1 mindspore.scipy.linalg.block_diag

`mindspore.scipy.linalg.block_diag(*arrs)`

根据输入的数组创建块对角矩阵。

输入为： A 、 B 和 C 的 Tensor 列表。输出为：在对角线上排列这些 Tensor 的块对角矩阵。

```
[ [A, 0, 0],
  [0, B, 0],
  [0, 0, C]]
```

说明：Windows 平台上还不支持 `block_diag`。

参数：

- **arrs** (list) - 最大支持 2D 的 Tensor 输入。一个或多个 Tensor，维度支持 0D，1D、2D。

返回：

对角线上含有 A 、 B 、 C ，... 的 Tensor，数据类型与 A 相同。

异常：

- **ValueError** - 输入参数中存在维度大于 2 的 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import numpy as onp
>>> from mindspore import Tensor
>>> from mindspore.scipy.linalg import block_diag
>>> A = Tensor(onp.array([[1, 0], [0, 1]]))
>>> B = Tensor(onp.array([[3, 4, 5], [6, 7, 8]]))
>>> C = Tensor(onp.array([[7]]))
>>> P = Tensor(onp.zeros((2, ), dtype='int32'))
>>> print(block_diag(A, B, C))
[[1 0 0 0 0 0]
 [0 1 0 0 0 0]
 [0 0 3 4 5 0]
 [0 0 6 7 8 0]
 [0 0 0 0 0 7]]
>>> print(block_diag(A, P, B, C))
[[1 0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0]
 [0 0 0 0 3 4 5 0]
 [0 0 0 0 6 7 8 0]
 [0 0 0 0 0 0 0 7]]
```

16.1.2 mindspore.scipy.linalg.cho_factor

`mindspore.scipy.linalg.cho_factor(a, lower=False, overwrite_a=False, check_finite=True)`

计算矩阵的 cholesky 分解，用于 `mindspore.scipy.linalg.cho_solve()`。

返回包含 cholesky 分解的矩阵，对于一个 Hermitian 正定矩阵 A ，根据 `lower` 取值，进行如下形式的分解：

- `lower` 为 True: $A = LL^*$
- `lower` 为 False: $A = U^*U$

其中， L^* 为 L 的共轭转置矩阵。其中， U^* 为 U 的共轭转置矩阵。

返回值可以直接作为 `mindspore.scipy.linalg.cho_solve()` 的第一个参数使用。

说明：

- Windows 平台上还不支持 `cho_factor`。
- 仅支持 float32、float64、int32、int64 类型的 Tensor 类型。
- 如果 Tensor 是 int32、int64 类型，它将被强制转换为：mstype.float64 类型。

警告： 返回的矩阵中还包含 cholesky 分解不使用的条目中的随机数据。如果需要将这些条目清零，请改用 `mindspore.scipy.linalg.cholesky()` 函数。

参数：

- **a** (Tensor) - 要分解的 (M, M) 方阵。
- **lower** (bool, 可选) - 是计算上三角还是下三角的 cholesky 分解。默认值：False。
- **overwrite_a** (bool, 可选) - 是否覆盖参数 `a` 中的数据（可能会提高性能）。默认值：False。在 MindSpore 中，这个参数当前不起作用。
- **check_finite** (bool, 可选) - 是否检查输入矩阵是否只包含有限数。禁用可能会带来性能增益，但如果输入确实包含 INF 或 NaN，则可能会导致问题（崩溃、程序不终止）。默认值：True。在 MindSpore 中，当前这个参数不起作用。

返回：

- **c** (Tensor) - 在上三角或下三角中包含 `a` 的 cholesky 因子的矩阵。矩阵的其他部分包含随机数据。
- **lower** (bool) - 表示 cholesky 因子是在下三角形还是上三角形。

异常：

- **ValueError** - 如果输入的 Tensor 不是 2D 方阵。

支持平台：

GPU CPU

样例：

```
>>> import numpy as onp
>>> from mindspore import Tensor
>>> from mindspore.scipy.linalg import cho_factor
>>> a = Tensor(onp.array([[9, 3, 1, 5], [3, 7, 5, 1], [1, 5, 9, 2], [5, 1, 2, 6]]).
↪ astype(onp.float32))
```

(下页继续)

(续上页)

```
>>> c, low = cho_factor(a)
>>> print(c)
[[ 3.          1.          0.33333334  1.66666666 ]
 [ 3.          2.4494898   1.9051585  -0.2721655 ]
 [ 1.          5.          2.2933078   0.8559526 ]
 [ 5.          1.          2.          1.5541857 ]]
```

16.1.3 mindspore.scipy.linalg.cho_solve

`mindspore.scipy.linalg.cho_solve(c_and_lower, b, overwrite_b=False, check_finite=True)`

给定 A 的 cholesky 分解, 求解线性方程组。

$$Ax = b$$

说明:

- 仅支持 float32、float64、int32、int64 类型的 Tensor 类型。
- 如果 Tensor 是 int32、int64 类型, 它将被强制转换为: `mstype.float64` 类型。

参数:

- **c_and_lower** ((Tensor, bool)) - a 的 cholesky 分解, 由: `mindspore.scipy.linalg.cho_factor()` 计算得出。
- **b** (Tensor) - 方程右侧的值。
- **overwrite_b** (bool, 可选) - 是否覆盖: b 中的数据 (可能会提高性能)。默认值: `False`。
- **check_finite** (bool, 可选) - 是否检查输入矩阵是否只包含有限数。禁用可能会带来性能增益, 但如果输入确实包含 INF 或 NaN, 则可能会导致问题 (崩溃、程序不终止)。默认值: `True`。

返回:

Tensor, 线性方程 $Ax = b$ 的解。

支持平台:

GPU CPU

样例:

```
>>> import numpy as onp
>>> import mindspore as ms
>>> a = ms.Tensor(onp.array([[9, 3, 1, 5], [3, 7, 5, 1], [1, 5, 9, 2], [5, 1, 2, 6]]).
↳ astype(onp.float32))
>>> b = ms.Tensor(onp.array([1, 1, 1, 1]).astype(onp.float32))
>>> c, low = ms.scipy.linalg.cho_factor(a)
>>> x = ms.scipy.linalg.cho_solve((c, low), b)
>>> print(x)
[-0.01749266  0.11953348  0.01166185  0.15743434]
```

16.1.4 mindspore.scipy.linalg.cholesky

`mindspore.scipy.linalg.cholesky(a, lower=False, overwrite_a=False, check_finite=True)`

计算矩阵的 `cholesky` 分解。

返回包含 `cholesky` 分解的矩阵，对于一个 `Hermitian` 正定矩阵 A ，根据 `lower` 取值，进行如下形式的分解：

- `lower` 为 `True`: $A = LL^*$
- `lower` 为 `False`: $A = U^*U$

其中， L^* 为 L 的共轭转置矩阵。其中， U^* 为 U 的共轭转置矩阵。

说明：

- Windows 平台上还不支持 `cholesky`。
- 仅支持 `float32`、`float64`、`int32`、`int64` 类型的 `Tensor` 类型。
- 如果 `Tensor` 是 `int32`、`int64` 类型，它将被强制转换为：`mstype.float64` 类型。

参数：

- **a** (`Tensor`) - 要分解的 (M, M) 方阵。
- **lower** (`bool`, 可选) - 是计算上三角还是下三角的 `cholesky` 分解。默认值：`False`。
- **overwrite_a** (`bool`, 可选) - 是否覆盖参数 `a` 中的数据（可能会提高性能）。默认值：`False`。在 `MindSpore` 中，这个参数当前不起作用。
- **check_finite** (`bool`, 可选) - 是否检查输入矩阵是否只包含有限数。禁用可能会带来性能增益，但如果输入确实包含 `INF` 或 `NaN`，则可能会导致问题（崩溃、程序不终止）。默认值：`True`。在 `MindSpore` 中，当前这个参数不起作用。

返回：

`Tensor`， a 的上三角或下三角 `cholesky` 因子。

异常：

- **ValueError** - 如果输入的 `Tensor` 不是 2D 方阵。

支持平台：

GPU CPU

样例：

```
>>> import numpy as onp
>>> from mindspore import Tensor
>>> from mindspore.scipy.linalg import cholesky
>>> a = Tensor(onp.array([[1, 2], [2, 5]]).astype(onp.float32))
>>> L = cholesky(a, lower=True)
>>> print(L)
[[1. 0.]
 [2. 1.]]
```

16.1.5 mindspore.scipy.linalg.eigh

`mindspore.scipy.linalg.eigh(a, b=None, lower=True, eigvals_only=False, overwrite_a=False, overwrite_b=False, turbo=True, eigvals=None, type=1, check_finite=True)`

求解复 Hermitian 矩阵或实对称矩阵的标准或广义特征值问题。

求出 a 的特征值 Tensor w 和可选的特征值 Tensor v ，其中 b 是正定的，使得对于每个特征值 λ (w 的第 i 个条目) 及其特征向量 vi (v 的第 i 列) 满足：

```
a @ vi = λ * b @ vi
vi.conj().T @ a @ vi = λ
vi.conj().T @ b @ vi = 1
```

在标准问题中，假设 b 是单位矩阵。

说明：

- Windows 平台上还不支持 `eigh`。
- 如果 Tensor 是 `int32`、`int64` 类型，它将被强制转换为：`mstype.float64` 类型。

参数：

- **a** (Tensor) - 一个 shape 为 (M, M) 的复 Hermitian 矩阵或实对称矩阵，用于计算其特征值和特征向量。
- **b** (Tensor, 可选) - 一个 shape 为 (M, M) 的复 Hermitian 矩阵或实对称正矩阵。如果缺省，则假定为传入单位矩阵。默认值：`None`。
- **lower** (bool, 可选) - 控制相关的 Tensor 数据是取自 a 和 b 的下三角还是上三角。默认值：`True`。
- **eigvals_only** (bool, 可选) - 是否只计算特征值，不计算特征向量。默认值：`False`。
- **overwrite_a** (bool, 可选) - 是否覆盖 a 中的数据（可能会提高性能）。默认值：`False`。
- **overwrite_b** (bool, 可选) - 是否覆盖 b 中的数据（可能会提高性能）。默认值：`False`。
- **turbo** (bool, 可选) - 使用分而治之算法（速度更快，但占用大量内存，仅适用于需要计算全量特征值的广义特征值问题）。如果不需要计算特征向量，则没有显著影响。默认值：`True`。
- **eigvals** (tuple, 可选) - 要返回的最小和最大（按升序排列）特征值和对应的特征向量的索引： $0 \leq lo \leq hi \leq M - 1$ 。如果缺省，则返回所有特征值和特征向量。默认值：`None`。
- **type** (int, 可选) - 对于广义问题，此参数指定 w 和 v 要解决的问题类型（仅取 1、2、3 作为可能的输入）：

```
1 => a @ v = w @ b @ v
2 => a @ b @ v = w @ v
3 => b @ a @ v = w @ v
```

对于标准问题，会忽略此关键字。默认值：`1`。

- **check_finite** (bool, 可选) - 是否检查输入矩阵是否只包含有限数。禁用可能会带来性能增益，但如果输入确实包含 `INF` 或 `NaN`，则可能会导致问题（崩溃、程序不终止）。默认值：`True`。

返回：

- **w** (Tensor) - 返回 shape 为 $(N,)$ 的 Tensor，其中特征值 $N(1 \leq N \leq M)$ ，按升序排列，根据其多样性重复。
- **v** (Tensor) - 如果 `eigvals_only=False`，返回 shape 为 (M, N) 的 Tensor。

异常：

- **RuntimeError** - 如果特征值计算不收敛或 b 不是正定矩阵，则会触发报错。如果输入矩阵不是对称矩阵或 Hermitian 矩阵，则不会报告错误，但结果将是错误的。
- **TypeError** - 如果 a 不是 Tensor。
- **TypeError** - 如果 low 不是 bool 类型。
- **TypeError** - 如果 $eigvals\_only$ 不是 bool 类型。
- **TypeError** - 如果 $overwrite\_a$ 不是 bool 类型。
- **TypeError** - 如果 $overwrite\_b$ 不是 bool 类型。
- **TypeError** - 如果 $turbo$ 不是 bool 类型。
- **TypeError** - 如果 $check\_finite$ 不是 bool 类型。
- **ValueError** - 如果 a 不是 2D 方阵。
- **ValueError** - 如果 b 不为 None。
- **ValueError** - 如果 $eigvals$ 不是 None。

支持平台：

GPU CPU

样例：

```
>>> import numpy as onp
>>> import mindspore.numpy as mnp
>>> from mindspore import Tensor, dtype
>>> from mindspore.scipy.linalg import eigh
>>> a = Tensor([[6, 3, 1, 5], [3, 0, 5, 1], [1, 5, 6, 2], [5, 1, 2, 2]], dtype.float64)
>>> w, v = eigh(a)
>>> print(onp.allclose(mnp.dot(a, v).asnumpy(), mnp.dot(v, mnp.diag(w)).asnumpy(), 1e-5, 1e-
↪8))
True
```

16.1.6 mindspore.scipy.linalg.inv

`mindspore.scipy.linalg.inv(a, overwrite_a=False, check_finite=True)`

计算矩阵的逆。

说明：

- Windows 平台上还不支持 `inv`。
- 仅支持 float32、float64、int32、int64 类型的 Tensor 类型。
- 如果 Tensor 是 int32、int64 类型，它将被强制转换为：`mstype.float64` 类型。

参数：

- **a** (Tensor) - 要求逆的方阵。
- **overwrite_a** (bool, 可选) - 是否覆盖参数 a 中的数据（可能会提高性能）。默认值：`False`。
- **check_finite** (bool, 可选) - 是否检查输入矩阵是否只包含有限数。禁用可能会带来性能增益，但如果输入确实包含 INF 或 NaN，则可能会导致问题（崩溃、程序不终止）。默认值：`True`。

返回：

Tensor, 矩阵 a 的逆。

异常：

- **LinAlgError** - 如果 a 是单数。
- **ValueError** - 如果 a 不是 2D 方阵。

支持平台：

GPU CPU

样例：

```
>>> import numpy as onp
>>> from mindspore import Tensor
>>> import mindspore.numpy as mnp
>>> from mindspore.scipy.linalg import inv
>>> a = Tensor(onp.array([[1., 2.], [3., 4.]])
>>> print(inv(a))
[[-2.   1. ]
 [ 1.5 -0.5]]
>>> print(mnp.dot(a, inv(a)))
[[1.0000000e+00 0.0000000e+00]
 [8.8817842e-16 1.0000000e+00]]
```

16.1.7 mindspore.scipy.linalg.lu

`mindspore.scipy.linalg.lu(a, permute_l=False, overwrite_a=False, check_finite=True)`

计算通用矩阵的 LU 分解。

分解为：

$$A = PLU$$

其中, P 是一个置换矩阵, L 是对角线元素全为 1 的下三角矩阵, U 是上三角矩阵。

说明：

- Windows 平台上还不支持 LU 。
- 仅支持 float32、float64、int32、int64 类型的 Tensor 类型。
- 如果 Tensor 是 int32、int64 类型, 它将被强制转换为: `mstype.float64` 类型。

参数：

- **a** (Tensor) - 要分解的 (M, N) 方阵。如果输入 Tensor 不是 float 类型, 那么它将被强制转换为: `mstype.float32`。
- **permute_l** (bool, 可选) - 执行乘法运算 PL 。
- **overwrite_a** (bool, 可选) - 是否覆盖 a 中的数据 (可能会提高性能)。默认值: `False`。
- **check_finite** (bool, 可选) - 是否检查输入矩阵是否只包含有限数。禁用可能会带来性能增益, 但如果输入确实包含 INF 或 NaN, 则可能会导致问题 (崩溃、程序不终止)。默认值: `True`。

返回：

如果 `permute_l == False`

- **p** (Tensor) - (M, M) 置换矩阵。
- **l** (Tensor) - (M, K) 对角线元素全为 1 的下三角矩阵或梯形矩阵。 $K = \min(M, N)$ 。
- **u** (Tensor) - (K, N) 上三角矩阵或梯形矩阵。

如果 `permute_l == True`

- **pl** (Tensor) - (M, K) 置换 L 矩阵。 $K = \min(M, N)$ 。
- **u** (Tensor) - (K, N) 上三角矩阵或梯形矩阵。

支持平台：

GPU CPU

样例：

```
>>> import numpy as onp
>>> from mindspore import Tensor
>>> from mindspore.scipy.linalg import lu
>>> a = Tensor(onp.array([[2, 5, 8, 7], [5, 2, 2, 8], [7, 5, 6, 6], [5, 4, 4, 8]]).
↳astype(onp.float64))
>>> p, l, u = lu(a)
>>> print(p)
[[0 1 0 0]
 [0 0 0 1]
 [1 0 0 0]
 [0 0 1 0]]
>>> print(l)
[[ 1.  0.  0.  0.]
 [ 0.2857143  1.  0.  0.]
 [ 0.71428573  0.12  1.  0.]
 [ 0.71428573 -0.44 -0.46153846  1.]]
>>> print(u)
[[ 7.  5.  6.  6.]
 [ 0.  3.57142854  6.28571415  5.28571415]
 [ 0.  0. -1.03999996  3.07999992]
 [ 0. -0. -0.  7.46153831]]
```

16.1.8 mindspore.scipy.linalg.lu_factor

`mindspore.scipy.linalg.lu_factor(a, overwrite_a=False, check_finite=True)`

计算方阵的 LU 分解，其输出可以直接作为 `lu_solve` 的输入。

分解为：

$$a = PLU$$

其中， P 是一个置换矩阵， L 是对角线元素全为 1 的下三角矩阵， U 是上三角矩阵。

说明：

- Windows 平台上还不支持 `lu_factor`。
- 仅支持 float32、float64、int32、int64 的 Tensor 类型。
- 如果 Tensor 是 int32、int64 类型，它将被强制转换为：`mstype.float64` 类型。

参数:

- **a** (Tensor) - 要分解的 (M, M) 方阵。如果输入 Tensor 不是 float 类型, 那么它将被强制转换为: `mstype.float32`。
- **overwrite_a** (bool, 可选) - 是否覆盖 *a* 中的数据 (可能会提高性能)。默认值: `False`。
- **check_finite** (bool, 可选) - 是否检查输入矩阵是否只包含有限数。禁用可能会带来性能增益, 但如果输入确实包含 *INF* 或 *NaN*, 则可能会导致问题 (崩溃、程序不终止)。默认值: `True`。

返回:

- **lu** (Tensor) - 一个 (M, M) 的方阵, 在它的上三角中包含 *u*, 它的下三角形中包含 *l*。不含 *l* 中对角线全为 1 的元素。
- **piv** (Tensor) - shape 为 $(M,)$ 的 Tensor, 表示置换矩阵 *p* 的索引: 索引中的第 *i* 个元素值 *j* 表示矩阵的第 *i* 行与第 *j* 行互换。

异常:

- **ValueError** - 如果 *a* 不是 2D 方阵。

支持平台:

GPU CPU

样例:

```
>>> import numpy as onp
>>> from mindspore import Tensor
>>> from mindspore.scipy.linalg import lu_factor
>>> a = Tensor(onp.array([[2, 5, 8, 7], [5, 2, 2, 8], [7, 5, 6, 6], [5, 4, 4, 8]]).
↳ astype(onp.float64))
>>> lu, piv = lu_factor(a)
>>> print(lu)
[[ 7.         5.         6.         6.         ]
 [ 0.28571429  3.57142857  6.28571429  5.28571429]
 [ 0.71428571  0.12        -1.04         3.08        ]
 [ 0.71428571 -0.44        -0.46153846  7.46153846]]
>>> print(piv)
[2 2 3 3]
```

16.1.9 mindspore.scipy.linalg.solve_triangular

`mindspore.scipy.linalg.solve_triangular(a, b, trans=0, lower=False, unit_diagonal=False, overwrite_b=False, debug=None, check_finite=True)`

假设 *a* 是一个三角矩阵, 求等式系统 $ax = b$ 的解 *x*。

说明:

- `solve_triangular` 目前仅用于 *mindscience* 科学计算场景, 尚不支持其他使用场景。
- `solve_triangular` 尚不支持 Windows 平台。

参数:

- **a** (Tensor) - shape 为 $(*, M, M)$ 的三角方阵, 其中 * 表示零或者更多的 Batch 维度。

- **b** (Tensor) - shape 为 $(*, M, N)$ 或者 $(*, M)$ 的矩阵或向量。
- **trans** (Union[int, str], 可选) - 求解系统的类型，默认值：0。

trans 值	求解系统
0 或 'N'	$ax = b$
1 或 'T'	$a^T x = b$
2 或 'C'	$a^H x = b$

- **lower** (bool, 可选) - a 是否为下三角矩阵，默认值：False。
- **unit_diagonal** (bool, 可选) - 如果为 True，那么 a 的对角线会被全部置为 1，默认值：False。
- **overwrite_b** (bool, 可选) - 在 MindSpore 中，当前这个参数不起作用。默认值：False。
- **debug** (Any, 可选) - 在 MindSpore 中，当前这个参数不起作用。默认值：None。
- **check_finite** (bool, 可选) - 在 MindSpore 中，当前这个参数不起作用。默认值：True。

返回：

Tensor, shape 为 $(*, M, N)$ 或者 $(*, M)$ 的矩阵。 $ax = b$ 等式的解，其中 x 的 shape 与 b 的 shape 相同。

异常：

- **ValueError** - 如果 a 的维度小于 2。
- **ValueError** - 如果 a 不是方阵。
- **TypeError** - 如果 a 和 b 的数据类型不同。
- **ValueError** - 如果 a 和 b 的 shape 不匹配。
- **ValueError** - 如果 $trans$ 不是 0、1、2、'N'、'T' 或 'C'。

支持平台：

Ascend CPU

样例：

```
>>> import numpy as onp
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.scipy.linalg import solve_triangular
>>> a = Tensor(onp.array([[3, 0, 0, 0], [2, 1, 0, 0], [1, 0, 1, 0], [1, 1, 1, 1]], onp.
    ↳float32))
>>> b = Tensor(onp.array([3, 1, 3, 4], onp.float32))
>>> x = solve_triangular(a, b, lower=True, unit_diagonal=False, trans='N')
>>> print(x)
[ 1. -1.  2.  2.]
>>> print(a @ x) # Check the result
[3. 1. 3. 4.]
```

16.1.10 mindspore.scipy.linalg.lstsq

`mindspore.scipy.linalg.lstsq(A, B, rcond=None, driver=None)`

求解计算线性等式系统 $AX = B$ 的最小二乘问题。

说明:

- `lstsq` 目前仅用于 *mindscience* 科学计算场景，尚不支持其他使用场景。
- `lstsq` 尚不支持 Windows 平台。

参数:

- **A** (Tensor) - 等式左边的左乘 Tensor, shape 为 $(*, M, N)$, 其中 $*$ 表示零或者更多的 Batch 维度。
- **B** (Tensor) - 等式右边的 Tensor, shape 为 $(*, M, K)$, 其中 $*$ 表示零或者更多的 Batch 维度。
- **rcond** (number.Number, 可选) - 在 MindSpore 中, 当前这个参数不起作用, 默认值: None。
- **driver** (string, 可选) - 使用哪个 LAPACK 函数求解最小二乘问题, 可选项有 "gels", "gelsy", "gelss", "gelssd", 默认值: None ("gelsy") 如果 A 条件数很小, 且 A 是一个满秩矩阵, 那么 "gels" 能很好地解决最小二乘问题, 如果 A 不一定满秩, 则建议使用 "gelsy", 如果 A 的条件数很大, "gelssd" 能更好地解决该问题, "gelss" 方法在以前更常用, 它占用更少的内存, 但是算得更慢。

返回:

- **solution** (Tensor), 最小二乘的解, shape 为 $(*, N, K)$, 其中 $*$ 等于广播后的 Batch 维度。
- **residues** (Tensor), $AX - B$, 其中 $*$ 等于广播后的 Batch 维度, 当 `driver` 是 ("gels", "gelss", "gelssd") 其中之一, 且 $M > N$ 时才会计算, 否则返回空 Tensor。
- **rank** (Tensor), A 的有效秩数。shape 为 $(*)$, 其中 $*$ 等于 A 的 Batch 维度。当 `driver` 是 ("gelsy", "gelss", "gelssd") 其中之一时才会计算, 否则返回空 Tensor。
- **singular_values** (Tensor), A 的奇异值, shape 为 $(*, \min(M, N))$, 其中 $*$ 等于 A 的 Batch 维度。当 `driver` 是 ("gelss", "gelssd") 其中之一时才会计算, 否则返回空 Tensor。

异常:

- **TypeError** - 如果 A 和 B 的数据类型不同。
- **ValueError** - 如果 A 的维度小于 2。
- **ValueError** - 如果 A 和 B 的 shape 不匹配。
- **ValueError** - 如果 `driver` 不是 None、"gels"、"gelsy"、"gelss" 或 "gelssd"。

支持平台:

Ascend CPU

样例:

```
>>> import numpy as onp
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.scipy.linalg import lstsq
>>> a = Tensor(onp.array([[3, 0, 0, 0], [2, 1, 0, 0], [1, 0, 1, 0], [1, 1, 1, 1]], onp.
    ↳float32))
>>> b = Tensor(onp.array([3, 1, 3, 4], onp.float32))
>>> solution, residual, rank, singular_value = lstsq(a, b)
```

(下页继续)

(续上页)

```
>>> print(solution)
[ 1. -1.  2.  2.]
>>> print(a @ solution)  # Check the result
[3.  1.  3.  4.]
```

16.2 mindspore.scipy.optimize

优化器子模块。

接口名	概述	支持平台
<code>mindspore.scipy.optimize.line_search</code>	满足强 Wolfe 条件的非精确线搜索。	GPU CPU
<code>mindspore.scipy.optimize.linear_sum_assignment</code>	解决线性和分配问题（指派问题）。	Ascend CPU
<code>mindspore.scipy.optimize.minimize</code>	最小化一个或多个变量的标量函数。	GPU CPU

16.2.1 mindspore.scipy.optimize.line_search

`mindspore.scipy.optimize.line_search` (*f, xk, pk, jac=None, gfk=None, old_fval=None, old_old_fval=None, c1=1e-4, c2=0.9, maxiter=20*)

满足强 Wolfe 条件的非精确线搜索。

来自 Wright 和 Nocedal, 'Numerical Optimization', 1999, 第 59-61 页, 算法 3.5 章节。

说明: Windows 平台上还不支持 `line_search`。

参数:

- **f** (function) - 形式为 $f(x)$ 的函数, 其中 x 是一个扁平 Tensor, 并返回一个实数标量。该函数应该由 `vjp` 定义的算子组成。
- **xk** (Tensor) - 初始猜测。
- **pk** (Tensor) - 要搜索的方向。假定方向是下降方向。
- **jac** (function) - 求 x 处的梯度的函数, 其中 x 是一个扁平的 Tensor, 函数返回一个 Tensor。如果要使用自动微分, 则可以传 `None`。
- **gfk** (Tensor) - `value_and_gradient` 作为位置的初始值。默认值: `None`。
- **old_fval** (Tensor) - 同 `gfk`。默认值: `None`。
- **old_old_fval** (Tensor) - 未使用的参数, 仅用于 `scipy` API 合规性。默认值: `None`。
- **c1** (float) - Wolfe 准则常量, 参见 `ref`。默认值: `1e-4`。
- **c2** (float) - 与 `c1` 相同。默认值: `0.9`。
- **maxiter** (int) - 搜索的最大迭代次数。默认值: `20`。

返回:

线搜索的结果。

支持平台:

GPU CPU

样例:

```
>>> import numpy as onp
>>> from mindspore.scipy.optimize import line_search
>>> from mindspore import Tensor
>>> x0 = Tensor(onp.ones(2).astype(onp.float32))
>>> p0 = Tensor(onp.array([-1, -1]).astype(onp.float32))
>>> def func(x):
...     return x[0] ** 2 - x[1] ** 3
>>> res = line_search(func, x0, p0)
>>> print(res.a_k)
1.0
```

16.2.2 mindspore.scipy.optimize.linear_sum_assignment

`mindspore.scipy.optimize.linear_sum_assignment` (*cost_matrix*, *maximize*, *dimension_limit=Tensor(sys.maxsize)*)
解决线性分配问题（指派问题）。

指派问题如下表示：

$$\min \sum_i \sum_j C_{i,j} X_{i,j}$$

其中， C 表示代价矩阵， $X_{i,j} = 1$ 表示列 j 被指派给了行 i 。

参数：

- **cost_matrix** (Tensor) - 二维代价矩阵。其 shape 为 (M, N) 。
- **maximize** (bool) - 为 True 则计算最大权值匹配，否则计算最小权值匹配。
- **dimension_limit** (Tensor, 可选) - 用来限制 `cost_matrix` 第二维的实际大小。默认值：Tensor(sys.maxsize)，表示没有限制。类型为零维 int64 类型 Tensor。

返回：

由 `row_idx` 和 `col_idx` 组成的 tuple。

- **row_idx** (Tensor) - 指派问题的行索引。如果指定了 `dimension_limit`，用 -1 在结尾补齐。其 shape 为 $(N,)$ ，其中 N 为 `cost_matrix` 维度较小值。
- **col_idx** (Tensor) - 指派问题的列索引。如果指定了 `dimension_limit`，用 -1 在结尾补齐。其 shape 为 $(N,)$ ，其中 N 为 `cost_matrix` 维度较小值。

异常：

- **TypeError** - 如果 `cost_matrix` 数据类型不是 float16、float32、float64、int8、int16、int32、int64、uint8、uint16、uint32、uint64、bool 之一。
- **TypeError** - 如果 `maximize` 的类型不是 bool。
- **TypeError** - 如果 `dimension_limit` 的数据类型不是 int64。
- **ValueError** - 如果 `cost_matrix` 维度不为 2。

支持平台：

Ascend CPU

样例:

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor
>>> import mindspore.scipy.optimize.linear_sum_assignment as lsap
>>> cost_matrix = Tensor(np.array([[2, 3, 3], [3, 2, 3], [3, 3, 2]]).astype(ms.float64))
>>> dimension_limit = Tensor(2)
>>> maximize = False
>>> a, b = lsap(cost_matrix, maximize, dimension_limit)
>>> print(a)
[0 1 -1]
>>> print(b)
[0 1 -1]
>>> a, b = lsap(cost_matrix, maximize)
>>> print(a)
[0 1 2]
>>> print(b)
[0 1 2]
```

16.2.3 mindspore.scipy.optimize.minimize

`mindspore.scipy.optimize.minimize` (*func*, *x0*, *args=()*, *method=None*, *jac=None*, *hess=None*, *hessp=None*, *bounds=None*, *constraints=()*, *tol=None*, *callback=None*, *options=None*)

最小化一个或多个变量的标量函数。

此函数的 API 与 SciPy 匹配，但有一些细微的差异：

当 *jac* 为 *None* 时，会使用 MindSpore 的自动微分功能计算 *func* 的反向梯度。*method* 参数是必需的。如果不指定求解器，将触发异常。尚未实现 SciPy 接口中的如下可选参数：“*hess*”、“*hessp*”、“*bounds*”、“*constraints*”、“*tol*”、“*callback*”。由于线搜索实现的差异，优化结果可能与 SciPy 不同。

说明:

- *minimize* 接口当前还不支持多维 Tensor 输入或求微分，但有支持的计划。
- Windows 平台上还不支持 *minimize*。
- *LAGRANGE* 方法仅在 GPU 上支持。

参数:

- **func** (Callable) - 要最小化的目标函数 $func(x, *args) \rightarrow float$ ，其中 *x* 是一个一维数组，其 shape 为 (*n*,)。 *args* 是一个 Tuple，用于指定 *func* 的执行所需的所有参数。当 *jac* 为 *None* 时，*func* 必须能支持微分。
- **x0** (Tensor) - 初始猜测。shape 为 (*n*,) 的实数数组，其中 *n* 是自变量的个数。
- **args** (Tuple) - 传递给目标函数的额外参数。默认值：()。
- **method** (str) - 求解器类型。应为 “*BFGS*” 和 “*LBFGS*”、“*LAGRANGE*” 中的一种。
- **jac** (Callable, 可选) - 计算梯度向量的函数。只支持 “*BFGS*” 和 “*LBFGS*”。如果为 *None*，则将使用 *func* 的反向梯度函数进行梯度计算。如果 *jac* 是可执行的，则应该是能返回梯度向量的函数： $jac(x, *args) \rightarrow array\_like, shape(n,)$ ，其中 *x* 是一个数组，其 shape 为 (*n*,)，*args* 是一个具有固定参数的元组。
- **hess** (Callable, 可选) - 计算 Hessian 矩阵的方法。当前尚未实现。
- **hessp** (Callable, 可选) - 目标函数的 Hessian 乘以任意向量 *p*。当前尚未实现。

- **bounds** (Sequence, 可选) - x 中的每个元素的 (min, max) 对的序列。当前尚未实现。
- **constraints** (Callable, 可选) - 表示不等式的约束, 约束中的每个函数都将 $function < 0$ 表示为不等式约束。
- **tol** (float, 可选) - 异常终止的容差范围。如需更具体的操控, 请使用求解器里专门的选项。默认值: `None`。
- **callback** (Callable, 可选) - 每次迭代后调用的可执行函数。当前尚未实现。
- **options** (Mapping[str, Any], 可选) - 用于保存求解器可选项的字典。所有求解器方法都能支持下述通用选项。默认值: `None`。
 - "history_size" (int) - 用于更新 Hessian 矩阵的逆的缓冲区大小, 仅支持与 `method="LBFGS"` 一起使用。默认值: 20。
 - "maxiter" (int) - 要执行的最大迭代次数。根据方法的不同, 每个迭代可能会使用多个函数进行求值。

以下选项是拉格朗日方法的专有选项:

- "save_tol" (list) - 保存 `tol` 的列表, 长度与 `constrains` 相同。
- "obj_weight" (float) - 目标函数的权重, 通常在 1.0 - 100000.0 之间。
- "lower" (Tensor) - 变量的下限约束, 必须具有与 `x0` 相同的 shape。
- "upper" (Tensor) - 变量的上限约束, 必须具有与 `x0` 相同的 shape。
- "learning_rate" (float) - 每个 Adam 步骤的学习率。
- "coincide_func" (Callable) - 子函数, 表示目标函数和约束之间的公共部分, 用于避免冗余计算。
- "rounds" (int) - 更新拉格朗日乘数的次数。
- "steps" (int) - 每执行 `steps` 次就执行一次 Adam 去更新拉格朗日乘数。
- "log_sw" (bool) - 是否打印每一步的 `loss` 值。

返回:

优化的结果。

支持平台:

GPU CPU

样例:

```
>>> import numpy as onp
>>> from mindspore.scipy.optimize import minimize
>>> from mindspore import Tensor
>>> x0 = Tensor(onp.zeros(2).astype(onp.float32))
>>> def func(p):
...     x, y = p
...     return (x ** 2 + y - 11.) ** 2 + (x + y ** 2 - 7.) ** 2
>>> res = minimize(func, x0, method='BFGS', options=dict(maxiter=None, gtol=1e-6))
>>> print(res.x)
[3. 2.]
>>> l_res = minimize(func, x0, method='LBFGS', options=dict(maxiter=None, gtol=1e-6))
>>> print(l_res.x)
[3. 2.]
```

17.1 介绍

`mindspore.multiprocessing` 提供了创建多进程的能力，其内部实现继承自 Python 原生的 `multiprocessing` 模块，并通过对部分接口进行重载，确保以 `fork` 方式创建多进程时 **MindSpore** 框架的正常使用。

当使用 `mindspore.multiprocessing` 并以 `fork` 方式创建多进程时，框架内部会对线程、锁等资源进行清理和重置，以保障框架功能正常。

- 在 `fork` 发生时，父进程在 `fork` 前会先等待内部线程任务执行完毕，并在当前线程主动持有 **GIL** 锁，以避免 `fork` 后的子进程无法持有 **GIL** 锁。
- 在 `fork` 发生后，父进程会释放主动持有的 **GIL** 锁。
- 在 `fork` 发生后，子进程会释放主动持有的 **GIL** 锁，然后对框架内部的线程等资源进行恢复和清理，并将后端重置为 CPU。

当不使用 `fork` 方式创建多进程时，上述流程不会被触发。此时使用 `mindspore.multiprocessing` 创建多进程的执行流程和使用 Python 原生的 `multiprocessing` 模块的流程完全一致。

由于 `mindspore.multiprocessing` 继承自 Python 原生的 `multiprocessing` 模块，其接口用法和原生模块完全兼容，用户可以直接把 `import multiprocessing` 修改为 `import mindspore.multiprocessing`。因此，此处不对原生接口进行重复描述。接口的详细介绍和使用方式请参考[multiprocessing](#)。

当前仅支持后端为 CPU 的 **Tensor** 在进程间共享。

17.2 使用说明

一个使用 `mindspore.multiprocessing` 创建多进程的样例如下：

```
from mindspore import Tensor, ops
import mindspore.multiprocessing as multiprocessing

def child_process():
    y = ops.log(Tensor(2.0))
    print("ops.log(Tensor(2.0))=", y)

if __name__ == '__main__':
    p = multiprocessing.Process(target=child_process)
    p.start()
    p.join()
```

运行结果如下：

```
ops.log(Tensor(2.0))= 0.6931472
```

一个使用 `mindspore.multiprocessing` 创建进程池的样例如下：

```
from mindspore import Tensor, ops
import mindspore.multiprocessing as multiprocessing

def child_process(x):
    return ops.log(x)

if __name__ == '__main__':
    with multiprocessing.Pool(processes=2) as pool:
        inputs = [Tensor(2.0), Tensor(2.0)]
        outputs = pool.map(child_process, inputs)
        print("ops.log(Tensor(2.0))=", outputs)
```

运行结果如下：

```
ops.log(Tensor(2.0))= [Tensor(shape=[], dtype=Float32, value= 0.693147), Tensor(shape=[], dtype=Float32, value= 0.693147)]
```

当用户在执行计算任务后，再使用 `fork` 方式创建多进程时，若使用的模块不是 `mindspore.multiprocessing`，可能会出现框架线程丢失导致的子进程卡住的问题。改为使用 `mindspore.multiprocessing` 模块的 `fork` 方式创建多进程，可以解决该问题。

仅在 POSIX 系统下（如 Linux 和 macOS）支持 `fork` 方式创建子进程，Windows 下不支持该方式

```
from mindspore import Tensor, ops
# Child process may be stuck when using 'import multiprocessing' .
import mindspore.multiprocessing as multiprocessing

def child_process(q):
    y = ops.log(Tensor(2.0))
    q.put(y)
    return

if __name__ == '__main__':
    multiprocessing.set_start_method("fork", force=True)
    print("parent process:ops.log(Tensor(2.0))=", ops.log(Tensor(2.0)))
    q = multiprocessing.Queue()
```

(下页继续)

(续上页)

```
p = multiprocessing.Process(target=child_process, args=(q,))
p.start()
print("child process:ops.log(Tensor(2.0))=", q.get())
p.join()
```

运行结果如下:

```
parent process:ops.log(Tensor(2.0))= 0.6931472
child process:ops.log(Tensor(2.0))= 0.6931472
```

当后端为 Ascend 时, 进程会独占卡资源。若用户在执行计算任务后创建子进程, 且子进程使用了父进程的资源执行另一个计算任务, 可能会因资源无法访问而报错。修改子进程创建方式为 `fork` 后, 框架会将子进程的后端重置为 CPU, 以避免资源冲突。

```
from mindspore import Tensor, ops, context
import mindspore.multiprocessing as multiprocessing

def child_process(q):
    y = ops.log(Tensor(2.0))
    q.put(y)
    return

if __name__ == '__main__':
    context.set_context(device_target="Ascend")
    # Child process may not be able to acquire resources when start_method is set to 'spawn' .
    multiprocessing.set_start_method("fork", force=True)
    print("parent process:ops.log(Tensor(2.0))=", ops.log(Tensor(2.0)))
    q = multiprocessing.Queue()
    p = multiprocessing.Process(target=child_process, args=(q,))
    p.start()
    print("child process:ops.log(Tensor(2.0))=", q.get())
    p.join()
```

运行结果如下:

```
parent process:ops.log(Tensor(2.0))= 0.6931472
child process:ops.log(Tensor(2.0))= 0.6931472
```

也可以将创建多进程的动作放在执行任何计算任务的前面, 并在子进程里手动修改后端。

```
from mindspore import Tensor, ops, context
import mindspore.multiprocessing as multiprocessing

def child_process(q):
    # Child process may not be able to acquire resources when using the same resources as parent_
    ↪process.
    context.set_context(device_target="CPU")
    y = ops.log(Tensor(2.0))
    q.put(y)
    return

if __name__ == '__main__':
    context.set_context(device_target="Ascend")
    multiprocessing.set_start_method("spawn", force=True)
    q = multiprocessing.Queue()
    p = multiprocessing.Process(target=child_process, args=(q,))
```

(下页继续)

(续上页)

```
p.start()
print("child process:ops.log(Tensor(2.0))=", q.get())
p.join()
# Child process may not be able to acquire resources when this compute task is executed.
↪before the child process is created.
print("parent process:ops.log(Tensor(2.0))=", ops.log(Tensor(2.0)))
```

运行结果如下:

```
child process:ops.log(Tensor(2.0))= 0.6931472
parent process:ops.log(Tensor(2.0))= 0.6931472
```

`mindspore.utils.stress_detect()`

用于检测硬件精度是否有故障。常见使用场景为在每个 step 或者保存 checkpoint 的时候，用户调用该接口，查看硬件是否有故障会影响精度。

返回：

int，返回值代表错误类型，0 表示正常；非 0 表示硬件故障。

支持平台：

Ascend

样例：

```
>>> from mindspore.utils import stress_detect
>>> ret = stress_detect()
>>> print(ret)
0
```

`mindspore.utils.dryrun.set_simulation()`

用于设置 dryrun 功能开启。dryrun 功能主要用于模拟大模型的实际运行，开启后可以在不占卡的情况下模拟出显存占用，编译信息等。在 PyNative 场景下开启后，若存在从 device 取值到 host 的场景，会打印出 Python 调用栈日志，告知用户这些值是不准确的。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore.utils import dryrun
>>> import numpy as np
>>> dryrun.set_simulation()
>>> print(os.environ.get('MS_SIMULATION_LEVEL'))
1
```

`mindspore.utils.dryrun.mock(mock_val, *args)`

在网络中若一些 if 判断需要用到实际执行值，虚拟执行无法获取，可以使用该接口返回模拟值。实际执行时可以获取到正确结果，返回执行值。

参数：

- **mock_val** (Union[Value, Tensor]) - 虚拟执行场景下，想要输出的值。
- **args** (Union[Value, function]) - 期望被 mock 掉的值，可以是函数或者变量。

返回：

开启 `dryrun` 功能，返回静态模拟值 (`mock_val`)，否则返回实际执行结果 (`args`)。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore.utils import dryrun
>>> import numpy as np
>>> dryrun.set_simulation()
>>> a = ms.Tensor(np.random.rand(3, 3).astype(np.float32))
>>> if dryrun.mock(True, a[0, 0] > 0.5):
...     print("return mock_val: True.")
return mock_val: True
>>>
>>> import mindspore as ms
>>> from mindspore.utils import dryrun
>>> import numpy as np
>>> a = ms.Tensor(np.ones((3, 3)).astype(np.float32))
>>> if dryrun.mock(False, a[0, 0] > 0.5):
...     print("return real execution: True.")
return real execution: True.
>>>
>>> import mindspore as ms
>>> from mindspore.utils import dryrun
>>> import numpy as np
>>> a = ms.Tensor(np.ones((3, 3)).astype(np.float32))
>>> if dryrun.mock(False, (a > 0.5).any):
...     print("return real execution: True.")
return real execution: True.
```


实验性模块。

19.1 实验性优化器

接口名	概述	支持平台
<code>mindspore.experimental.optim.Optimizer</code>	用于参数更新的优化器基类。	Ascend GPU CPU
<code>mindspore.experimental.optim.Adadelta</code>	Adadelta 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.Adagrad</code>	Adagrad 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.Adam</code>	Adaptive Moment Estimation (Adam) 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.Adamax</code>	Adamax 算法的实现 (基于无穷范数的 Adam 算法)。	Ascend GPU CPU
<code>mindspore.experimental.optim.AdamW</code>	Adaptive Moment Estimation Weight Decay(AdamW) 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.ASGD</code>	Averaged Stochastic Gradient Descent 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.NAdam</code>	NAdam 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.RAdam</code>	RAdam 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.RMSprop</code>	RMSprop 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.Rprop</code>	Rprop 算法的实现。	Ascend GPU CPU
<code>mindspore.experimental.optim.SGD</code>	随机梯度下降算法。	Ascend GPU CPU

19.1.1 mindspore.experimental.optim.Optimizer

class mindspore.experimental.optim.Optimizer (params, defaults)
用于参数更新的优化器基类。

警告：这是一个实验性的优化器模块，需要和 LRScheduler 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **defaults** (dict) - 一个包含了优化器参数默认值的字典（当参数组未指定参数值时，使用此默认值）。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, Tensor, Parameter
>>> from mindspore import ops
>>> from mindspore.experimental import optim
>>>
>>> class MySGD(optim.Optimizer):
...     def __init__(self, params, lr):
...         defaults = dict(lr=lr)
...         super(MySGD, self).__init__(params, defaults)
...
...     def construct(self, gradients):
...         for group_id, group in enumerate(self.param_groups):
...             id = self.group_start_id[group_id]
...             for i, param in enumerate(group["params"]):
...                 next_param = param + gradients[id+i] * group["lr"]
...                 ops.assign(param, next_param)
>>>
>>> net = nn.Dense(8, 2)
>>> data = Tensor(np.random.rand(20, 8).astype(np.float32))
>>> label = Tensor(np.random.rand(20, 2).astype(np.float32))
>>>
>>> optimizer = MySGD(net.trainable_params(), 0.01)
>>> optimizer.add_param_group({"params": Parameter([0.01, 0.02])})
>>>
>>> criterion = nn.MAELoss(reduction="mean")
>>>
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = criterion(logits, label)
...     return loss, logits
>>>
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>>
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     print(loss)
```

(下页继续)

(续上页)

```
>>>
>>> train_step(data, label)
```

add_param_group (param_group)

为 `Optimizer.param_groups` 属性添加一个参数组。

参数:

- **param_group** (dict) - 指定了当前网络参数组的特定的优化器配置。

19.1.2 mindspore.experimental.optim.Adadelta

class mindspore.experimental.optim.**Adadelta** (params, lr=1.0, rho=0.9, eps=1e-6, weight_decay=0.0, *, maximize=False)

Adadelta 算法的实现。

更新公式如下:

input : γ (lr), θ_0 (params), $f(\theta)$ (objective), ρ (decay), λ (weight decay)
initialize : $v_0 \leftarrow 0$ (square avg), $u_0 \leftarrow 0$ (accumulate variables)

for $t = 1$ **to** ... **do**
 $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$
if $\lambda \neq 0$
 $g_t \leftarrow g_t + \lambda \theta_{t-1}$
 $v_t \leftarrow v_{t-1} \rho + g_t^2 (1 - \rho)$
 $\Delta x_t \leftarrow \frac{\sqrt{u_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} g_t$
 $u_t \leftarrow u_{t-1} \rho + \Delta x_t^2 (1 - \rho)$
 $\theta_t \leftarrow \theta_{t-1} - \gamma \Delta x_t$

return θ_t

警告: 这是一个实验性的优化器接口, 需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数:

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值: 1.0。
- **rho** (float, 可选) - 梯度平方平均值的系数。上述公式中的 ρ 。默认值: 0.9。
- **eps** (float, 可选) - 加在分母上的值, 以确保数值稳定。上述公式中的 ϵ 。默认值: 1e-6。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值: 0.0。

关键字参数:

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值: False。

输入:

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常:

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - ϵ 小于等于 0。
- **ValueError** - ρ 范围不在 [0, 1] 之间。
- **ValueError** - weight_decay 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.Adadelta(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.3 mindspore.experimental.optim.Adagrad

class mindspore.experimental.optim.**Adagrad** (*params, lr=1e-2, lr_decay=0.0, weight_decay=0.0, initial_accumulator_value=0.0, eps=1e-10, *, maximize=False*)

Adagrad 算法的实现。

更新公式如下：

input : γ (lr), θ_0 (params), $f(\theta)$ (objective), λ (weight decay),
 τ (initial accumulator value), η (lr decay)
initialize : $\text{state\_sum}_0 \leftarrow 0$

for $t = 1$ **to** ... **do**
 $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$
 $\tilde{\gamma} \leftarrow \gamma / (1 + (t - 1)\eta)$
if $\lambda \neq 0$
 $g_t \leftarrow g_t + \lambda \theta_{t-1}$
 $\text{state\_sum}_t \leftarrow \text{state\_sum}_{t-1} + g_t^2$
 $\theta_t \leftarrow \theta_{t-1} - \tilde{\gamma} \frac{g_t}{\sqrt{\text{state\_sum}_t + \epsilon}}$

return θ_t

警告：这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-2。
- **lr_decay** (float, 可选) - 学习率衰减系数。默认值：0.。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：0.。
- **initial_accumulator_value** (float, 可选) - 初始累加器的值。默认值：0.。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值：1e-10。

关键字参数：

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常：

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - 学习率衰减系数小于 0。
- **ValueError** - *weight_decay* 小于 0。
- **ValueError** - *initial_accumulator_value* 小于 0。
- **ValueError** - *eps* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.Adagrad(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.4 mindspore.experimental.optim.Adam

class mindspore.experimental.optim.**Adam** (*params*, *lr*=1e-3, *betas*=(0.9, 0.999), *eps*=1e-8, *weight_decay*=0.0, *amsgrad*=False, *, *maximize*=False)

Adaptive Moment Estimation (Adam) 算法的实现。

更新公式如下：

```

input :  $\gamma$  (lr),  $\beta_1, \beta_2$  (betas),  $\theta_0$  (params),  $f(\theta)$  (objective)
          $\lambda$  (weight decay), amsgrad, maximize
initialize :  $m_0 \leftarrow 0$  (first moment),  $v_0 \leftarrow 0$  (second moment),  $\hat{v}_0^{max} \leftarrow 0$ 


---


for  $t = 1$  to ... do
    if maximize :
         $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ 
    else
         $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ 
    if  $\lambda \neq 0$ 
         $g_t \leftarrow g_t + \lambda \theta_{t-1}$ 
     $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
     $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
     $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
     $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ 
    if amsgrad
         $\widehat{v}_t^{max} \leftarrow \max(\widehat{v}_t^{max}, \widehat{v}_t)$ 
         $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t^{max}} + \epsilon)$ 
    else
         $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ 


---


return  $\theta_t$ 

```

警告：该优化器接口的实现公式与论文中的实现并不完全一致。若要使用完全一致的接口，建议使用 `mindspore.mint.optim.Adam`，它目前仅支持 Ascend。这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-3。
- **betas** (Tuple[float, float], 可选) - 动量矩阵的指数衰减率。参数范围 (0.0, 1.0)。默认值：(0.9, 0.999)。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值：1e-8。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：0.0。
- **amsgrad** (bool, 可选) - 是否使用 AMSGrad 算法。默认值：False。

关键字参数：

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常:

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *eps* 小于 0。
- **ValueError** - *betas* 范围不在 [0, 1) 之间。
- **ValueError** - *weight_decay* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.Adam(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.5 mindspore.experimental.optim.Adamax

class mindspore.experimental.optim.**Adamax** (*params*, *lr*=2e-3, *betas*=(0.9, 0.999), *eps*=1e-8, *weight_decay*=0.0, *, *maximize*=False)

Adamax 算法的实现 (基于无穷范数的 Adam 算法)。

更新公式如下:

input : γ (*lr*), β_1, β_2 (*betas*), θ_0 (*params*), $f(\theta)$ (*objective*), λ (*weight decay*), ϵ (*epsilon*)

initialize : $m_0 \leftarrow 0$ (first moment), $u_0 \leftarrow 0$ (infinity norm)

for $t = 1$ **to** ... **do**

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$

if $\lambda \neq 0$

$g_t \leftarrow g_t + \lambda \theta_{t-1}$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$u_t \leftarrow \max(\beta_2 u_{t-1}, |g_t| + \epsilon)$

$\theta_t \leftarrow \theta_{t-1} - \frac{\gamma m_t}{(1 - \beta_1^t) u_t}$

return θ_t

警告：这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值： $2e-3$ 。
- **betas** (Tuple[float, float], 可选) - 梯度及其平方的运行平均值的系数。默认值：(0.9, 0.999)。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值： $1e-8$ 。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：0.0。

关键字参数：

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常：

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *eps* 小于 0。
- **ValueError** - *betas* 范围不在 [0,1) 之间。
- **ValueError** - *weight_decay* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.Adamax(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```


19.1.6 mindspore.experimental.optim.AdamW

class mindspore.experimental.optim.**AdamW** (*params*, *lr*=1e-3, *betas*=(0.9, 0.999), *eps*=1e-8, *weight_decay*=1e-2, *amsgrad*=False, *, *maximize*=False)

Adaptive Moment Estimation Weight Decay(AdamW) 算法的实现。

更新公式如下：

```

input :  $\gamma$ (lr),  $\beta_1, \beta_2$ (betas),  $\theta_0$ (params),  $f(\theta)$ (objective),  $\epsilon$  (epsilon)
          $\lambda$ (weight decay), amsgrad, maximize
initialize :  $m_0 \leftarrow 0$  (first moment),  $v_0 \leftarrow 0$  (second moment),  $\hat{v}_0^{max} \leftarrow 0$ 


---


for  $t = 1$  to ... do
    if maximize :
         $g_t \leftarrow -\nabla_{\theta} f_t(\theta_{t-1})$ 
    else
         $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ 
     $\theta_t \leftarrow \theta_{t-1} - \gamma \lambda \theta_{t-1}$ 
     $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
     $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ 
    if amsgrad
         $\hat{v}_t^{max} \leftarrow \max(\hat{v}_t^{max}, \hat{v}_t)$ 
         $\theta_t \leftarrow \theta_t - \gamma \hat{m}_t / (\sqrt{\hat{v}_t^{max}} + \epsilon)$ 
    else
         $\theta_t \leftarrow \theta_t - \gamma \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ 


---


return  $\theta_t$ 

```

警告： 这是一个实验性的优化器接口，需要和 [LRScheduler](#) 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-3。
- **betas** (Tuple[float, float], 可选) - 动量矩阵的指数衰减率。参数范围 (0.0, 1.0)。默认值：(0.9, 0.999)。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值：1e-8。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：1e-2。
- **amsgrad** (bool, 可选) - 是否使用 AMSGrad 算法。默认值：False。

关键字参数：

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor], 可选) - 网络权重的梯度。

异常：

- **ValueError** - 学习率不是 int、float 或 Tensor。

- **ValueError** - 学习率小于 0。
- **ValueError** - *eps* 小于 0。
- **ValueError** - *betas* 范围不在 [0, 1) 之间。
- **ValueError** - *weight_decay* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.AdamW(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.7 mindspore.experimental.optim.ASGD

class mindspore.experimental.optim.ASGD (*params*, *lr=1e-2*, *lambd=1e-4*, *alpha=0.75*, *t0=1e6*, *weight_decay=0.0*, *maximize=False*)

Averaged Stochastic Gradient Descent 算法的实现。

警告：这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-2。
- **lambd** (float, 可选) - 衰减项。默认值：1e-4。
- **alpha** (float, 可选) - eta 更新的幂。默认值：0.75。
- **t0** (float, 可选) - 开始计算平均的时刻。默认值：1e6。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：0.0。
- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常:

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *weight_decay* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.ASGD(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.8 mindspore.experimental.optim.NAdam

class mindspore.experimental.optim.NAdam(*params*, *lr*=2e-3, *betas*=(0.9, 0.999), *eps*=1e-8, *weight_decay*=0.0, *momentum_decay*=4e-3)

NAdam 算法的实现。

警告: 这是一个实验性的优化器接口, 需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数:

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值: 2e-3。
- **betas** (Tuple[float, float], 可选) - 梯度及其平方的运行平均值的系数。默认值: (0.9, 0.999)。
- **eps** (float, 可选) - 加在分母上的值, 以确保数值稳定。必须大于 0。默认值: 1e-8。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值: 0.0。
- **momentum_decay** (float, 可选) - 动量衰减系数。默认值: 4e-3。

输入:

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常:

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *eps* 小于 0。
- **ValueError** - *weight_decay* 小于 0。
- **ValueError** - *momentum_decay* 小于 0。
- **ValueError** - *betas* 内元素取值范围不在 [0, 1) 之间。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.NAdam(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.9 mindspore.experimental.optim.RAdam

class mindspore.experimental.optim.**RAdam** (*params, lr=1e-3, betas=(0.9, 0.999), eps=1e-8, weight_decay=0.0*)
RAdam 算法的实现。

更新公式如下：

```

input :  $\gamma$  (lr),  $\beta_1, \beta_2$  (betas),  $\theta_0$  (params),  $f(\theta)$  (objective),  $\lambda$  (weightdecay),
         $\epsilon$  (epsilon)
initialize :  $m_0 \leftarrow 0$  ( first moment),  $v_0 \leftarrow 0$  ( second moment),
         $\rho_\infty \leftarrow 2/(1 - \beta_2) - 1$ 

```

```

for  $t = 1$  to ... do
     $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ 
    if  $\lambda \neq 0$ 
         $g_t \leftarrow g_t + \lambda \theta_{t-1}$ 
         $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$ 
         $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ 
         $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ 
         $\rho_t \leftarrow \rho_\infty - 2t\beta_2^t / (1 - \beta_2^t)$ 
        if  $\rho_t > 5$ 
             $l_t \leftarrow \frac{\sqrt{(1-\beta_2^t)}}{\sqrt{v_t + \epsilon}}$ 
             $r_t \leftarrow \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}}$ 
             $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m}_t r_t l_t$ 
        else
             $\theta_t \leftarrow \theta_{t-1} - \gamma \widehat{m}_t$ 

```

```

return  $\theta_t$ 

```

警告： 这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-3。
- **betas** (Tuple[float, float], 可选) - 梯度及其平方的运行平均值的系数。默认值：(0.9, 0.999)。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值：1e-8。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：0.0。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常：

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *eps* 小于 0。
- **ValueError** - *weight_decay* 小于 0。
- **ValueError** - *betas* 内元素的取值范围不在 [0, 1) 之间。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.RAdam(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss

```

19.1.10 mindspore.experimental.optim.RMSprop

class mindspore.experimental.optim.**RMSprop** (*params, lr=1e-2, alpha=0.99, eps=1e-8, weight_decay=0.0, momentum=0.0, centered=False, maximize=False*)

RMSprop 算法的实现。

警告：这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-2。
- **alpha** (float, 可选) - 平滑常数。默认值：0.99。
- **eps** (float, 可选) - 加在分母上的值，以确保数值稳定。必须大于 0。默认值：1e-8。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)。默认值：0.0。
- **momentum** (float, 可选) - 动量系数。默认值：0.0。
- **centered** (bool, 可选) - 如果为 True，则计算 centered RMSprop，梯度通过其方差进行归一化。默认值：False。
- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常：

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *momentum* 小于 0。
- **ValueError** - *alpha* 小于 0。
- **ValueError** - *eps* 小于 0。

- **ValueError** - *weight_decay* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.RMSprop(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.11 mindspore.experimental.optim.Rprop

class mindspore.experimental.optim.**Rprop** (*params, lr=1e-2, etas=(0.5, 1.2), step_sizes=(1e-6, 50), *, maximize=False*)

Rprop 算法的实现。

警告：这是一个实验性的优化器接口，需要和 **LRScheduler** 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor], 可选) - 学习率。默认值：1e-2。
- **etas** (Tuple[float, float], 可选) - (etaminus, etaplus)，进行增大和减小的因子。默认值：(0.5, 1.2)。
- **step_sizes** (Tuple[float, float], 可选) - 设定的最小步长和最大步长。默认值：(1e-6, 50)。

关键字参数：

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常：

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *etas[1]* 小于等于 1。

- **ValueError** - *etas[0]* 不在 0-1 之间。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.Rprop(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.12 mindspore.experimental.optim.SGD

class mindspore.experimental.optim.SGD(*params, lr, momentum=0, dampening=0, weight_decay=0.0, nesterov=False, *, maximize=False*)

随机梯度下降算法。

$$v_{t+1} = u * v_t + gradient * (1 - dampening)$$

如果 *nesterov* 为 *True*：

$$p_{t+1} = p_t - lr * (gradient + u * v_{t+1})$$

如果 *nesterov* 为 *False*：

$$p_{t+1} = p_t - lr * v_{t+1}$$

需要注意的是，对于训练的第一步 $v_{t+1} = gradient$ 。

p、*v* 和 *u* 分别表示 *parameters*、*accum* 和 *momentum*。

警告： 这是一个实验性的优化器接口，需要和 `LRScheduler` 下的动态学习率接口配合使用。

参数：

- **params** (Union[list(Parameter), list(dict)]) - 网络参数的列表或指定了参数组的列表。
- **lr** (Union[int, float, Tensor]) - 学习率。
- **momentum** (Union[int, float], 可选) - 动量值。默认值：0。
- **weight_decay** (float, 可选) - 权重衰减 (L2 penalty)，必须大于等于 0。默认值：0.0。

- **dampening** (Union[int, float], 可选) - 动量的阻尼值。默认值：0。
- **nesterov** (bool, 可选) - 启用 Nesterov 动量。如果使用 Nesterov，动量必须为正，阻尼必须等于 0.0。默认值：False。

关键字参数：

- **maximize** (bool, 可选) - 是否根据目标函数最大化网络参数。默认值：False。

输入：

- **gradients** (tuple[Tensor]) - 网络权重的梯度。

异常：

- **ValueError** - 学习率不是 int、float 或 Tensor。
- **ValueError** - 学习率小于 0。
- **ValueError** - *momentum* 和 *weight_decay* 值小于 0.0。
- **ValueError** - *momentum*、*dampening* 和 *weight_decay* 不是 int 或 float。
- **ValueError** - *nesterov* 和 *maximize* 不是布尔类型。
- **ValueError** - *nesterov* 为 True 时，*momentum* 不为正或 *dampening* 不为 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.SGD(net.trainable_params(), lr=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
```

19.1.13 LRScheduler 类

本模块中的动态学习率都是 LRScheduler 的子类，此模块仅与 mindspore.experimental.optim 下的优化器配合使用，使用时将优化器实例传递给 LRScheduler 类。在训练过程中，LRScheduler 子类通过调用 *step* 方法进行学习率的动态改变。

```
import mindspore
from mindspore import nn
from mindspore.experimental import optim
# Define the network structure of LeNet5. Refer to
# https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
```

(下页继续)

(续上页)

```

net = LeNet5()
loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
optimizer = optim.Adam(net.trainable_params(), lr=0.05)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.1)
def forward_fn(data, label):
    logits = net(data)
    loss = loss_fn(logits, label)
    return loss, logits
grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
def train_step(data, label):
    (loss, _), grads = grad_fn(data, label)
    optimizer(grads)
    return loss
for epoch in range(6):
    # Create the dataset taking MNIST as an example. Refer to
    # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py

    for data, label in create_dataset(need_download=False):
        train_step(data, label)
    scheduler.step()

```

接口名	概述	支持平台
<code>mindspore.experimental.optim.lr_scheduler.LRScheduler</code>	动态学习率的基类。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.ConstantLR</code>	将每个参数组的学习率，按照衰减因子 <i>factor</i> 进行衰减，直到 <i>last_epoch</i> 达到 <i>total_iters</i> 。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.CosineAnnealingLR</code>	使用余弦退火，改变优化器参数组的学习率。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.CosineAnnealingWarmRestarts</code>	使用余弦退火热重启，改变优化器参数组的学习率。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.CyclicLR</code>	根据循环学习率策略（CLR）设置每个参数组的学习率。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.ExponentialLR</code>	每个 <i>epoch</i> 呈指数衰减的学习率，即乘以 <i>gamma</i> 。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.LambdaLR</code>	将每个参数组的学习率，设定为初始学习率乘以指定的 <i>lr_lambda</i> 函数。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.LinearLR</code>	线性减小学习率乘法因子，并将每个参数组的学习率按照此乘法因子进行衰减，直到 <i>last_epoch</i> 数达到 <i>total_iters</i> 。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.MultiplicativeLR</code>	将每个参数组当前的学习率，按照传入的 <i>lr_lambda</i> 函数乘以指定的乘法因子。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.MultiStepLR</code>	当 <i>epoch/step</i> 达到 <i>milestones</i> 时，将每个参数组的学习率按照乘法因子 <i>gamma</i> 进行变化。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.PolynomialLR</code>	每个 <i>epoch</i> 通过多项式拟合来调整学习率。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.ReduceLROnPlateau</code>	学习率调整优化器，当指标停止改进时降低学习率。	Ascend GPU CPU
<code>mindspore.experimental.optim.lr_scheduler.SequentialLR</code>	将 <i>schedulers</i> 中的多个学习率调整策略，按照顺序串联起来，在 <i>milestone</i> 时切换到下一个学习率调整策略。	Ascend GPU CPU

下页继续

表 2 - 续上页

<code>mindspore.experimental.optim.lr_scheduler.StepLR</code>	每 <code>step_size</code> 个 epoch 按 <code>gamma</code> 衰减每个参数组的学习率。	Ascend GPU CPU

mindspore.experimental.optim.lr_scheduler.LRScheduler

class mindspore.experimental.optim.lr_scheduler.LRScheduler (optimizer, last_epoch=- 1)
动态学习率的基类。

警告：这是一个实验性的动态学习率模块，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (`mindspore.experimental.optim.Optimizer`) - 优化器实例。
- **last_epoch** (int, 可选) - 当前学习率调整策略的 `step()` 方法的执行次数。默认值：-1。

异常：

- **TypeError** - `optimizer` 不满足类型要求。
- **KeyError** - `last_epoch` 不是 -1 且 'initial_lr' 不在参数组内。
- **ValueError** - `last_epoch` 不是 int 类型。
- **ValueError** - `last_epoch` 小于-1。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>>
>>> class ConstantLR(optim.lr_scheduler.LRScheduler):
...     def __init__(self, optimizer, factor=0.5, total_iters=3, last_epoch=-1):
...         self.factor = factor
...         self.total_iters = total_iters
...         super(ConstantLR, self).__init__(optimizer, last_epoch)
...
...     def get_lr(self):
...         if self.last_epoch == 0:
...             return [lr * self.factor for lr in self._last_lr]
...         if self.last_epoch != self.total_iters:
...             return [lr * 1. for lr in self._last_lr]
...         return [lr / self.factor for lr in self._last_lr]
>>>
>>> net = nn.Dense(8, 2)
>>> optimizer = optim.SGD(net.trainable_params(), 0.01)
>>> scheduler = ConstantLR(optimizer)
>>> for i in range(4):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.005)]
```

(下页继续)

(续上页)

```
[Tensor(shape=[], dtype=Float32, value= 0.005)]
[Tensor(shape=[], dtype=Float32, value= 0.01)]
[Tensor(shape=[], dtype=Float32, value= 0.01)]
```

get_last_lr()

返回当前使用的学习率。

step (*epoch=None*)

按照定义的计算逻辑计算并修改学习率。

参数:

- **epoch** (int, 可选) - epoch 数。默认值: None。

mindspore.experimental.optim.lr_scheduler.ConstantLR

class mindspore.experimental.optim.lr_scheduler.**ConstantLR** (*optimizer, factor=1.0 / 3, total_iters=5, last_epoch=- 1*)

将每个参数组的学习率, 按照衰减因子 *factor* 进行衰减, 直到 *last_epoch* 达到 *total_iters*。注意, 这种衰减可能与外部对于学习率的改变同时发生。

警告: 这是一个实验性的动态学习率接口, 需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数:

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **factor** (float, 可选) - 学习率的衰减因子。默认值: 1.0 / 3。
- **total_iters** (int, 可选) - 学习率进行衰减的执行次数, 当 *last_epoch* 数达到 *total_iters*, 恢复学习率。默认值: 5。
- **last_epoch** (int, 可选) - 当前 scheduler 的 *step()* 方法的执行次数。默认值: -1。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> net = nn.Dense(2, 3)
>>> optimizer = optim.Adam(net.trainable_params(), 0.05)
>>> # Assuming optimizer uses lr = 0.05 for all groups
>>> # lr = 0.025 if epoch < 4
>>> # lr = 0.05 if epoch >= 4
>>> scheduler = optim.lr_scheduler.ConstantLR(optimizer, factor=0.5, total_iters=4)
>>> for i in range(6):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.025)]
[Tensor(shape=[], dtype=Float32, value= 0.025)]
[Tensor(shape=[], dtype=Float32, value= 0.025)]
```

(下页继续)

(续上页)

```
[Tensor(shape=[], dtype=Float32, value= 0.05)]
[Tensor(shape=[], dtype=Float32, value= 0.05)]
[Tensor(shape=[], dtype=Float32, value= 0.05)]
```

`mindspore.experimental.optim.lr_scheduler.CosineAnnealingLR`

class `mindspore.experimental.optim.lr_scheduler.CosineAnnealingLR` (*optimizer*, *T_max*, *eta_min*=0.0, *last_epoch*=-1)

使用余弦退火，改变优化器参数组的学习率。下述公式中， η_{max} 为初始学习率， η_{min} 为学习率变化的最小值， T_{max} 为余弦函数的半周期， T_{cur} 为当前周期内的迭代数， η_t 为当前学习率。

$$\begin{aligned}\eta_t &= \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), \\ T_{cur} &\neq (2k+1)T_{max}; \\ \eta_{t+1} &= \eta_t + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 - \cos\left(\frac{1}{T_{max}}\pi\right)\right), \\ T_{cur} &= (2k+1)T_{max}.\end{aligned}$$

详情请查看 [SGDR: Stochastic Gradient Descent with Warm Restarts](#)。

警告：这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (`mindspore.experimental.optim.Optimizer`) - 优化器实例。
- **T_max** (int) - 余弦函数的半周期。
- **eta_min** (float, 可选) - 学习率的最小值。默认值：0.0。
- **last_epoch** (int, 可选) - 当前 scheduler 的 `step()` 方法的执行次数。默认值：-1。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore.experimental import optim
>>> from mindspore import nn
>>> net = nn.Dense(3, 2)
>>> optimizer = optim.SGD(net.trainable_params(), lr=0.1, momentum=0.9)
>>> scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=2)
>>>
>>> for i in range(6):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.05)]
[Tensor(shape=[], dtype=Float32, value= 0)]
[Tensor(shape=[], dtype=Float32, value= 0.05)]
[Tensor(shape=[], dtype=Float32, value= 0.1)]
[Tensor(shape=[], dtype=Float32, value= 0.05)]
[Tensor(shape=[], dtype=Float32, value= 0)]
```

mindspore.experimental.optim.lr_scheduler.CosineAnnealingWarmRestarts

class mindspore.experimental.optim.lr_scheduler.**CosineAnnealingWarmRestarts** (*optimizer*, *T_0*,
T_mult=1,
eta_min=0,
last_epoch=- 1)

使用余弦退火热重启, 改变优化器参数组的学习率。

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + \cos \left(\frac{T_{cur}}{T_i} \pi \right) \right)$$

其中, η_{max} 为初始学习率, η_{min} 为学习率变化的最小值, η_t 为当前学习率, T_0 为初始周期, T_i 为当前周期, 即 SGDR 两次热重启之间的迭代数, T_{cur} 为当前周期内的迭代数。

当 $T_{cur} = T_i$ 时, $\eta_t = \eta_{min}$; 热重启后 $T_{cur} = 0$ 时, $\eta_t = \eta_{max}$ 。

详情请查看 [SGDR: Stochastic Gradient Descent with Warm Restarts](#)。

警告: 这是一个实验性的动态学习率接口, 需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数:

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **T_0** (int) - 余弦函数的初始周期数。
- **T_mult** (int, 可选) - 迭代中对 T_i 进行增长的乘法系数。默认值: 1。
- **eta_min** (Union(float, int), 可选) - 学习率的最小值。默认值: 0。
- **last_epoch** (int, 可选) - 当前学习率调整策略的 *step()* 方法的执行次数。默认值: -1。

异常:

- **ValueError** - T_0 小于等于 0 或不是 int 类型。
- **ValueError** - T_{mult} 小于等于 1 或不是 int 类型。
- **ValueError** - eta_{min} 不是 int 或 float 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore.experimental import optim
>>> from mindspore import nn
>>> net = nn.Dense(3, 2)
>>> optimizer = optim.SGD(net.trainable_params(), lr=0.1, momentum=0.9)
>>> scheduler = optim.lr_scheduler.CosineAnnealingWarmRestarts(optimizer, 2)
>>> iters = 3
>>> for epoch in range(2):
...     for i in range(iters):
...         scheduler.step(epoch + i / iters)
...         current_lr = scheduler.get_last_lr()
...         print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.1)]
[Tensor(shape=[], dtype=Float32, value= 0.0933013)]
[Tensor(shape=[], dtype=Float32, value= 0.075)]
```

(下页继续)

(续上页)

```
[Tensor(shape=[], dtype=Float32, value= 0.05)]
[Tensor(shape=[], dtype=Float32, value= 0.025)]
[Tensor(shape=[], dtype=Float32, value= 0.00669873)]
```

step (*epoch=None*)

按照定义的计算逻辑计算并修改学习率。

参数：

- **epoch** (int, 可选) - epoch 数。默认值：None。

mindspore.experimental.optim.lr_scheduler.CyclicLR

class mindspore.experimental.optim.lr_scheduler.**CyclicLR** (*optimizer, base_lr, max_lr, step_size_up=2000, step_size_down=None, mode='triangular', gamma=1., scale_fn=None, scale_mode='cycle', last_epoch=-1*)

根据循环学习率策略（CLR）设置每个参数组的学习率。该策略以恒定频率在两个边界之间循环学习率值，详情请参考论文 [Cyclical Learning Rates for Training Neural Networks](#)。两个边界之间的距离可以在每次迭代或每个周期的基础上缩放。

正如论文中提出的，该类（对学习率变化幅度）有三个内置计算策略：

- "triangular"：没有幅度缩放的基本三角循环。
- "triangular2"：每个循环将初始幅度缩放一半的基本三角循环。
- "exp_range"：在每个迭代中按照 $\text{gamma}^{\text{cycle iterations}}$ 缩放初始幅度。

警告：这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **base_lr** (Union(float, list)) - 初始学习率，也是优化器参数组中学习率的下界值。
- **max_lr** (Union(float, list)) - 每个参数组的学习率上界值。在功能上，(`max_lr - base_lr`) 定义了学习率周期变化的幅度。周期内，当前的学习率的计算方式为 `base_lr` 与振幅乘以缩放系数的和。
- **step_size_up** (int, 可选) - 递增半周期内的训练迭代次数。默认值：2000。
- **step_size_down** (int, 可选) - 递减半周期内的训练迭代次数。如果 `step_size_down` 为 None，则设置为 `step_size_up` 的值。默认值：None。
- **mode** (str, 可选) - "triangular", "triangular2" 或 "exp_range"。对应的计算策略详见上文，如果 `scale_fn` 不是 None，则此参数无效。默认值："triangular"。
- **gamma** (float, 可选) - "exp_range" 模式下的常量，计算方式为 $\text{gamma}^{(\text{cycle iterations})}$ 。默认值：1.0。
- **scale_fn** (function, 可选) - 由单个参数的 `lambda` 匿名函数定义的自定义扩展策略，其中对所有的 $x \geq 0$, $0 \leq \text{scale\_fn}(x) \leq 1$ 。如果设定了此参数，则 `mode` 设定值将被忽略。默认值：None。
- **scale_mode** (str, 可选) - 'cycle' 或 'iterations'。定义 `scale_fn` 是按周期数还是周期内的迭代次数（当前周期内训练迭代的次数）。若传入不合法输入，将默认使用 'iterations' 模式。默认值：'cycle'。
- **last_epoch** (int, 可选) - 当前 scheduler 的 `step()` 方法的执行次数。默认值：-1。

异常：

- **ValueError** - *base_lr* 为 list 或 tuple 时, 长度不等于参数组数目。
- **ValueError** - *max_lr* 为 list 或 tuple 时, 长度不等于参数组数目。
- **ValueError** - *mode* 不是 ["triangular", "triangular2", "exp_range"] 且 *scale_fn* 为 None。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore.experimental import optim
>>> from mindspore import nn
>>> net = nn.Dense(3, 2)
>>> optimizer = optim.SGD(net.trainable_params(), lr=0.1, momentum=0.9)
>>> scheduler = optim.lr_scheduler.CyclicLR(optimizer, base_lr=0.01, max_lr=0.1)
>>> expect_list = [[0.010045], [0.01009], [0.010135], [0.01018], [0.010225]]
>>>
>>> for i in range(5):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.010045)]
[Tensor(shape=[], dtype=Float32, value= 0.01009)]
[Tensor(shape=[], dtype=Float32, value= 0.010135)]
[Tensor(shape=[], dtype=Float32, value= 0.01018)]
[Tensor(shape=[], dtype=Float32, value= 0.010225)]
```

`mindspore.experimental.optim.lr_scheduler.ExponentialLR`

class `mindspore.experimental.optim.lr_scheduler.ExponentialLR` (*optimizer, gamma, last_epoch=-1*)
每个 epoch 呈指数衰减的学习率, 即乘以 *gamma*。注意, 这种衰减可能与外部对于学习率的改变同时发生。

警告: 这是一个实验性的动态学习率接口, 需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数:

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **gamma** (float) - 学习率衰减的乘法因子。
- **last_epoch** (int, 可选) - 最后一个 epoch 的索引。默认值: -1。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(16 * 5 * 5, 120)
...     def construct(self, x):
```

(下页继续)

(续上页)

```

...         return self.fc(x)
>>> net = Net()
>>> optimizer = optim.Adam(net.trainable_params(), 0.01)
>>> scheduler = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.5)
>>> for i in range(3):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.005)]
[Tensor(shape=[], dtype=Float32, value= 0.0025)]
[Tensor(shape=[], dtype=Float32, value= 0.00125)]

```

mindspore.experimental.optim.lr_scheduler.LambdaLR

class mindspore.experimental.optim.lr_scheduler.LambdaLR(optimizer, lr_lambda, last_epoch=-1)

将每个参数组的学习率，设定为初始学习率乘以指定的 *lr_lambda* 函数。当 *last_epoch = -1* 时，将学习率设置成初始学习率。

警告： 这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **lr_lambda** (Union(function, list)) - 一个关于 *last_epoch* 的匿名函数，或类似函数的列表，列表中每个函数对应 *optimizer.param_groups* 中的每个参数组。
- **last_epoch** (int, 可选) - 当前 scheduler 的 *step()* 方法的执行次数。默认值：-1。

异常：

- **ValueError** - *lr_lambda* 的长度不等于参数组数目。

支持平台：

Ascend GPU CPU

样例：

```

>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> net = nn.Dense(2, 3)
>>> optimizer = optim.Adam(net.trainable_params(), 0.01)
>>> lmbda = lambda epoch: 0.9 ** epoch
>>> scheduler = optim.lr_scheduler.LambdaLR(optimizer, lr_lambda=[lmbda])
>>> for i in range(3):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.009)]
[Tensor(shape=[], dtype=Float32, value= 0.0081)]
[Tensor(shape=[], dtype=Float32, value= 0.00729)]

```

mindspore.experimental.optim.lr_scheduler.LinearLR

class mindspore.experimental.optim.lr_scheduler.**LinearLR** (*optimizer, start_factor=1.0 / 3, end_factor=1.0, total_iters=5, last_epoch=- 1*)

线性减小学习率乘法因子，并将每个参数组的学习率按照此乘法因子进行衰减，直到 *last_epoch* 数达到 *total_iters*。注意，这种衰减可能与外部对于学习率的改变同时发生。

警告：这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **start_factor** (float, 可选) - 初始的乘法因子值，后续向 *end_factor* 进行线性变化。默认值：1.0 / 3。
- **end_factor** (float, 可选) - 线性变化过程结束时的乘法因子值。默认值：1.0。
- **total_iters** (int, 可选) - 迭代的次数。默认值：5。
- **last_epoch** (int, 可选) - 当前 scheduler 的 *step()* 方法的执行次数。默认值：-1。

异常：

- **ValueError** - *start_factor* 不在 (0, 1] 范围内。
- **ValueError** - *end_factor* 不在 [0, 1] 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.Adam(net.trainable_params(), lr=0.05)
>>> # Assuming optimizer uses lr = 0.05 for all groups
>>> # lr = 0.025     if epoch == 0
>>> # lr = 0.03125   if epoch == 1
>>> # lr = 0.0375    if epoch == 2
>>> # lr = 0.04375   if epoch == 3
>>> # lr = 0.05      if epoch >= 4
>>> scheduler = optim.lr_scheduler.LinearLR(optimizer, start_factor=0.5, total_iters=4)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _) = grad_fn(data, label)
...     optimizer(grads)
...     return loss
>>> for epoch in range(5):
...     # Create the dataset taking MNIST as an example. Refer to
```

(下页继续)

(续上页)

```
... # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
... for data, label in create_dataset():
...     train_step(data, label)
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
```

`mindspore.experimental.optim.lr_scheduler.MultiplicativeLR`

class `mindspore.experimental.optim.lr_scheduler.MultiplicativeLR` (*optimizer, lr_lambda, last_epoch=-1*)

将每个参数组当前的学习率，按照传入的 *lr_lambda* 函数乘以指定的乘法因子。当 *last_epoch = -1* 时，将学习率设置成初始学习率。

警告：这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **lr_lambda** (`Union(function, list)`) - 一个关于 epoch/step 的乘法函数，或类似函数的列表，列表中每个函数对应 *optimizer.param_groups* 中的每个参数组。
- **last_epoch** (`int`, 可选) - 当前 scheduler 的 *step()* 方法的执行次数。默认值：-1。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> net = nn.Dense(2, 3)
>>> optimizer = optim.Adam(net.trainable_params(), 0.01)
>>> lambda = lambda epoch: 0.95
>>> scheduler = optim.lr_scheduler.MultiplicativeLR(optimizer, lr_lambda=lambda)
>>> for i in range(3):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.0095)]
[Tensor(shape=[], dtype=Float32, value= 0.009025)]
[Tensor(shape=[], dtype=Float32, value= 0.00857375)]
```

mindspore.experimental.optim.lr_scheduler.MultiStepLR

class mindspore.experimental.optim.lr_scheduler.**MultiStepLR** (*optimizer, milestones, gamma=0.1, last_epoch=-1*)

当 epoch/step 达到 *milestones* 时, 将每个参数组的学习率按照乘法因子 *gamma* 进行变化。注意, 这种衰减可能与外部对于学习率的改变同时发生。

警告: 这是一个实验性的动态学习率接口, 需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数:

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **milestones** (list) - 阈值列表, 当 *last_epoch* 数达到阈值时, 将学习率乘以 *gamma*。
- **gamma** (float, 可选) - 学习率的乘法因子。默认值: 0.1。
- **last_epoch** (int, 可选) - 当前 scheduler 的 *step()* 方法的执行次数。默认值: -1。

异常:

- **TypeError** - *milestones* 不是列表。
- **TypeError** - *milestones* 的元素不是 int 类型。
- **TypeError** - *gamma* 不是 float 类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> net = nn.Dense(2, 3)
>>> optimizer = optim.Adam(net.trainable_params(), 0.05)
>>> # Assuming optimizer uses lr = 0.05 for all groups
>>> # lr = 0.05      if epoch < 2
>>> # lr = 0.005     if 2 <= epoch < 4
>>> # lr = 0.0005    if epoch >= 4
>>> scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[2,4], gamma=0.1)
>>> for i in range(6):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.05)]
[Tensor(shape=[], dtype=Float32, value= 0.005)]
[Tensor(shape=[], dtype=Float32, value= 0.005)]
[Tensor(shape=[], dtype=Float32, value= 0.0005)]
[Tensor(shape=[], dtype=Float32, value= 0.0005)]
[Tensor(shape=[], dtype=Float32, value= 0.0005)]
```

`mindspore.experimental.optim.lr_scheduler.PolynomialLR`

class `mindspore.experimental.optim.lr_scheduler.PolynomialLR` (*optimizer*, *total_iters*=5, *power*=1.0, *last_epoch*=-1)

每个 `epoch` 通过多项式拟合来调整学习率。当 `epoch` 大于等于 `total_iters` 时，学习率设置为 0。注意，这种衰减可能与外部对于学习率的改变同时发生。

学习率计算的多项式公式如下：

$$factor = \left(\frac{1.0 - \frac{last\_epoch}{total\_iters}}{1.0 - \frac{last\_epoch - 1.0}{total\_iters}} \right)^{power}$$

$$lr = lr \times factor$$

警告：这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (`mindspore.experimental.optim.Optimizer`) - 优化器实例。
- **total_iters** (int, 可选) - 通过多项式拟合调整学习率的迭代次数。默认值：5。
- **power** (float, 可选) - 多项式的幂。默认值：1.0。
- **last_epoch** (int, 可选) - 最后一个 `epoch` 的索引。默认值：-1。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.fc = nn.Dense(16 * 5 * 5, 120)
...     def construct(self, x):
...         return self.fc(x)
>>> net = Net()
>>> optimizer = optim.Adam(net.trainable_params(), 0.01)
>>> scheduler = optim.lr_scheduler.PolynomialLR(optimizer)
>>> for i in range(6):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.008)]
[Tensor(shape=[], dtype=Float32, value= 0.006)]
[Tensor(shape=[], dtype=Float32, value= 0.004)]
[Tensor(shape=[], dtype=Float32, value= 0.002)]
[Tensor(shape=[], dtype=Float32, value= 0)]
[Tensor(shape=[], dtype=Float32, value= 0)]
```

mindspore.experimental.optim.lr_scheduler.ReduceLROnPlateau

```
class mindspore.experimental.optim.lr_scheduler.ReduceLROnPlateau (optimizer, mode='min',
                                                                    factor=0.1, patience=10,
                                                                    threshold=1e-4,
                                                                    threshold_mode='rel',
                                                                    cooldown=0, min_lr=0,
                                                                    eps=1e-8)
```

学习率调整优化器，当指标停止改进时降低学习率。训练中学习停滞情况下，模型通常会受益于将学习率降低 2~10 倍。

调度程序在执行过程中读取指标 *metrics*，如果指标在 *patience* 个周期内没有得到改进，则通过 *step* 方法调整学习率。

警告：这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **mode** (str, 可选) - 触发模式，当监控指标在最低点/最高点时触发降低学习率。在 'min' 模式下，当监控指标不再下降，降低学习率；在 'max' 模式下，当监控指标不再上升，降低学习率。默认值：'min'。
- **factor** (float, 可选) - 学习率衰减因子。默认值：0.1。
- **patience** (int, 可选) - 监控指标无改善情况下等待的 epoch 数。例如，如果 *patience* = 2，则前 2 个无改善的 epoch 将被忽略，从第 3 个 epoch 开始降低学习率。默认值：10。
- **threshold** (float, 可选) - 衡量指标变好的最小阈值。默认值：1e-4。
- **threshold_mode** (str, 可选) - 衡量指标变好的模式，可取值 'rel' 或 'abs'。默认值：'rel'。

假设 *best* 代表当前性能指标的最值。

– 在 'rel' 模式下，指标与比例形式的 *threshold* 比较：

- * 当 *mode* 为 'max'，若超过 $best * (1 + threshold)$ 则认为指标变好。
- * 当 *mode* 为 'min'，若低于 $best * (1 - threshold)$ 则认为指标变好。

– 在 'abs' 模式下，指标与绝对值形式的 *threshold* 比较：

- * 当 *mode* 为 'max'，若超过 $best + threshold$ 则认为指标变好。
- * 当 *mode* 为 'min'，若低于 $best - threshold$ 则认为指标变好。

- **cooldown** (int, 可选) - 在降低学习率后恢复正常运行之前要等待的 epoch 数。默认值：0。
- **min_lr** (Union(float, list), 可选) - 标量或标量列表，所有参数组或每个组的学习率最小值。默认值：0。
- **eps** (float, 可选) - 应用于学习率的最小衰减。如果学习率变化的差异小于 *eps*，则忽略更新。默认值：1e-8。

异常：

- **ValueError** - *factor* 大于等于 1.0。
- **TypeError** - *optimizer* 不是 *Optimizer*。
- **ValueError** - *min_lr* 为 list 或 tuple 时，其长度不等于参数组数目。
- **ValueError** - *mode* 不是 'min' 或 'max'。
- **ValueError** - *threshold_mode* 不是 'rel' 或 'abs'。

支持平台：

Ascend GPU CPU

样例:

```
>>> from mindspore.experimental import optim
>>> from mindspore import nn
>>> net = nn.Dense(3, 2)
>>> optimizer = optim.Adam(net.trainable_params(), 0.1)
>>> scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', patience=0)
>>> metrics = [1, 1.5, 1.8, 0.4, 0.5]
>>> for i in range(5):
...     scheduler.step(metrics[i])
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.1)]
[Tensor(shape=[], dtype=Float32, value= 0.01)]
[Tensor(shape=[], dtype=Float32, value= 0.001)]
[Tensor(shape=[], dtype=Float32, value= 0.001)]
[Tensor(shape=[], dtype=Float32, value= 0.0001)]
```

get_last_lr()

返回当前使用的学习率。

in_cooldown()

返回是否在 *cooldown* 时期。

step(metrics)

按照定义的计算逻辑计算并修改学习率。

参数:

- **metrics** (Union(int, float)) - 评估指标值。

mindspore.experimental.optim.lr_scheduler.SequentialLR

class mindspore.experimental.optim.lr_scheduler.**SequentialLR** (*optimizer, schedulers, milestones, last_epoch=-1*)

将 *schedulers* 中的多个学习率调整策略, 按照顺序串联起来, 在 *milestone* 时切换到下一个学习率调整策略。

警告: 这是一个实验性的动态学习率接口, 需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数:

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **schedulers** (list[*mindspore.experimental.optim.lr_scheduler.LRScheduler*]) - 被顺序执行的学习率调度器列表。
- **milestones** (list) - 里程碑节点的整数列表, 设定了每个 epoch 哪个学习率调整策略被调用。
- **last_epoch** (int, 可选) - 当前学习率调整策略的 *step()* 方法的执行次数。默认值: -1。

异常:

- **ValueError** - *schedulers* 中的 *optimizer* 与传入的 *optimizer* 不同。
- **ValueError** - *schedulers* 中的 *optimizer* 与 *schedulers[0].optimizer* 不同。
- **ValueError** - *milestones* 的长度不等于 *schedulers* 的长度减 1。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore.experimental import optim
>>> from mindspore import nn
>>> net = nn.Dense(3, 2)
>>> optimizer = optim.Adam(net.trainable_params(), 0.1)
>>> scheduler1 = optim.lr_scheduler.ConstantLR(optimizer, factor=0.1, total_iters=2)
>>> scheduler2 = optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.9)
>>> scheduler = optim.lr_scheduler.SequentialLR(optimizer, schedulers=[scheduler1,
↪scheduler2], milestones=[2])
>>> for i in range(6):
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()
...     print(current_lr)
[Tensor(shape=[], dtype=Float32, value= 0.01)]
[Tensor(shape=[], dtype=Float32, value= 0.1)]
[Tensor(shape=[], dtype=Float32, value= 0.09)]
[Tensor(shape=[], dtype=Float32, value= 0.081)]
[Tensor(shape=[], dtype=Float32, value= 0.0729)]
[Tensor(shape=[], dtype=Float32, value= 0.06561)]
```

get_last_lr()

返回当前使用的学习率。

step()

按照定义的计算逻辑计算并修改学习率。

mindspore.experimental.optim.lr_scheduler.StepLR

class mindspore.experimental.optim.lr_scheduler.**StepLR** (*optimizer, step_size, gamma=0.1, last_epoch=-1*)
每 *step_size* 个 epoch 按 *gamma* 衰减每个参数组的学习率。*StepLR* 对于学习率的衰减可能与外部对于学习率的改变同时发生。

警告：这是一个实验性的动态学习率接口，需要和 `mindspore.experimental.optim` 下的接口配合使用。

参数：

- **optimizer** (*mindspore.experimental.optim.Optimizer*) - 优化器实例。
- **step_size** (int) - 学习率衰减的周期。
- **gamma** (float, 可选) - 学习率衰减的乘法因子。默认值: 0.1。
- **last_epoch** (int, 可选) - 当前 scheduler 的 *step()* 方法的执行次数。默认值: -1。

支持平台：

Ascend GPU CPU

样例：


```

>>> import mindspore
>>> from mindspore import nn
>>> from mindspore.experimental import optim
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> optimizer = optim.Adam(net.trainable_params(), lr=0.05)
>>> # Assuming optimizer uses lr = 0.05 for all groups
>>> # lr = 0.05      if epoch < 2
>>> # lr = 0.005     if 2 <= epoch < 4
>>> # lr = 0.0005    if 4 <= epoch < 6
>>> scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.1)
>>> def forward_fn(data, label):
...     logits = net(data)
...     loss = loss_fn(logits, label)
...     return loss, logits
>>> grad_fn = mindspore.value_and_grad(forward_fn, None, optimizer.parameters, has_aux=True)
>>> def train_step(data, label):
...     (loss, _), grads = grad_fn(data, label)
...     optimizer(grads)
...     return loss
>>> for epoch in range(6):
...     # Create the dataset taking MNIST as an example. Refer to
...     # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
...     for data, label in create_dataset():
...         train_step(data, label)
...     scheduler.step()
...     current_lr = scheduler.get_last_lr()

```

19.2 实验性 EmbeddingService

`mindspore.experimental.es.
EmbeddingService`

ES (EmbeddingService) 功能可支持大小表 Embedding 的模型训练和推理，为训练和推理提供统一的 Embedding 管理、存储和计算的能力。

`mindspore.experimental.es.
EsEmbeddingLookup`

提供大表特征值 keys 的查询功能。

`mindspore.experimental.es.
ESEmbeddingSmallTableLookup`

提供小表特征值 keys 的查询功能。

19.2.1 mindspore.experimental.es.EmbeddingService

class mindspore.experimental.es.EmbeddingService

ES (EmbeddingService) 功能可支持大小表 Embedding 的模型训练和推理，为训练和推理提供统一的 Embedding 管理、存储和计算的能力。其中大表指 vocab_size 超过 10 万的数据量，推荐用户存储在 PS (Parameter Server) 上的表；小表指 vocab_size 小于 10 万的数据量，推荐用户存储在 device 上的表。

目前 ES 特性只能创建一个 EmbeddingService 对象的实例。

警告：这是一个实验性的 EmbeddingService 接口。

说明：此接口需要先调用 `mindspore.communication.init()` 完成动态组网后才能生效。

异常：

- **ValueError** - 实例化对象的时候，未设置 ESCLUSTER_CONFIG_PATH 环境变量会报错。
- **ValueError** - ESCLUSTER_CONFIG_PATH 路径对应的配置文件中，每个 server 参数服务器总数超过 4。
- **ValueError** - ESCLUSTER_CONFIG_PATH 路径对应的配置文件中，参数服务器总数超过 4。

支持平台：

Atlas A2 训练系列产品

completion_key (*completion_key, mask=True*)

为每张大表初始化 completion key 策略。

参数：

- **completion_key** (int) - completion key 取值。
- **mask** (bool, 可选) - 是否更新 completion key，设置为 False 则不更新。默认值为 True。

返回：

CompletionKeyOption 对象。

异常：

- **TypeError** - 如果 completion_key 入参类型不是 int。
- **TypeError** - 如果 mask 入参类型不是 bool。

counter_filter (*filter_freq, default_key=None, default_value=None*)

为每张大表设置准入策略。

说明：此功能仅支持训练模式。如果训练时使用了特征准入，然后执行推理，对于某些 key 查询不到的情况，当前支持使用 default_value 进行设置。

参数：

- **filter_freq** (int) - 特征准入的频率阈值。
- **default_key** (int, 可选) - 出现次数未达到阈值的 key 特征，在查询时返回 default_key 对应的特征值。默认值为 None。
- **default_value** (Union[int, float], 可选) - 出现次数未达到阈值的 value 特征，在查询时返回 embedding_dim 长度的 default_value 作为其特征值。默认值为 None。

返回：

CounterFilter 对象。

异常：

- **TypeError** - 如果 filter_freq 类型不是 int。
- **ValueError** - 如果 filter_freq 小于 0。

- **ValueError** - 如果 *default_key* 和 *default_value* 都是 None。
- **ValueError** - 如果 *default_key* 和 *default_value* 都不是 None。
- **TypeError** - 如果 *default_key* 是 None, 且 *default_value* 不是 int 或 float。
- **TypeError** - 如果 *default_value* 是 None, 且 *default_key* 不是 int。

embedding_ckpt_export (*file_path, trainable_var*)

导出每张大表的 embedding table 和优化器参数, 及小表的 embedding tale。

说明: 此功能只支持 rank 0 执行。针对大表, 在导出之前需要先调用 `mindspore.experimental.es.EmbeddingService.embedding_variable_option()` 接口为每张表设置对应的被动淘汰条件。

参数:

- **file_path** (str) - 存放 ckpt 的地址, 最后一个字符不能为 "/"。
- **trainable_var** (list[parameter]) - 存放所有小表的 parameter 信息。

返回:

EmbeddingComputeVarExport 算子的返回值及小表的导出结果。

embedding_ckpt_import (*file_path*)

导入 *file_path* 下所有的 embedding 文件和 ckpt 文件。

参数:

- **file_path** (str) - 存放 ckpt 的地址, 最后一个字符不能为 "/"。

返回:

EmbeddingComputeVarImport 算子的返回值及小表导入结果。

embedding_evict (*steps_to_live*)

为所有大表使能主动淘汰。

参数:

- **steps_to_live** (int) - 特征 key 设置的淘汰存活步数。

返回:

EmbeddingTableEvict 算子的返回值。

异常:

- **TypeError** - 如果 *steps_to_live* 类型不是 int。
- **ValueError** - 如果 *steps_to_live* 不大于 0。

embedding_init (*name, init_vocabulary_size, embedding_dim, max_feature_count=None, initializer=Uniform(scale=0.01), embedding_type='PS', ev_option=None, multihot_lens=None, optimizer=None, allow_merge=False, optimizer_param=None, mode='train'*)

大小表的初始化接口。

参数:

- **name** (str) - 表名。
- **init_vocabulary_size** (int) - 初始化表的大小。
- **embedding_dim** (int) - 表中插入数据的维度大小。

- **max_feature_count** (int, 可选) - 每次查询的 **keys** 的数量。默认值为 `None`。
- **initializer** (Initializer, 可选) - 表的初始化策略，默认值为 `Uniform(scale=0.01)`。
- **embedding_type** (str, 可选) - 生成 **embedding** 表的类型，可配置参数 `["PS", "data_parallel"]`，`"PS"` 表示初始化大表，`"data_parallel"` 表示初始化小表。默认值为 `"PS"`。
- **ev_option** (EmbeddingVariableOption, 可选) - 大表的一些属性，是 `mindspore.experimental.es.EmbeddingService.embedding_variable_option()` 函数的返回值，为 `EmbeddingVariableOption` 对象。默认值为 `None`。
- **multihot_lens** (int, 可选) - 小表的属性，只有 `allow_merge` 使能之后才用，当前不支持。默认值为 `None`。
- **optimizer** (str, 可选) - 训练场景下大表优化器的类型，每张大表不能共用优化器，当前只支持 `"Adam"`、`"Ftrl"`、`"SGD"` 和 `"RMSProp"`。默认值为 `None`。
- **allow_merge** (bool, 可选) - 小表的属性，表示是否进行小表合并，当前不支持。默认值为 `False`。
- **optimizer_param** (float, 可选) - 用户配置的优化器参数：`initial_accumulator_value`，表示 **moment** 累加器的初始值。默认值为 `None`。
- **mode** (str, 可选) - 网络运行模式，可配置参数 `["train", "predict", "export"]`，`"train"` 表示训练模式，`"predict"` 表示推理模式，`"export"` 表示模型导出模式。默认值为 `"train"`。

返回：

- 如果初始化小表，返回小表的 **dict** 信息。
- 如果初始化大表，返回 `EmbeddingServiceOut` 对象，包含 5 个参数：`table_id_dict`、`es_initializer`、`es_counter_filter`、`es_padding_keys`、`es_completion_keys`。
 - `table_id_dict(dict)`，大表名和 `table_id` 的键值对。
 - `es_initializer(dict)`，`table_id` 和大表属性 `EsInitializer` 对象的键值对。
 - `es_counter_filter(dict)`，`table_id` 和准入条件的键值对。
 - `es_padding_keys(dict)`，`table_id` 和 `padding key` 条件的键值对。
 - `es_completion_keys(dict)`，`table_id` 和 `completion ket` 的键值对。

异常：

- **ValueError** - 如果未设置 `name`，`init_vocabulary_size`，`embedding_dim`，`max_feature_count`。
- **ValueError** - 如果 `name`，`init_vocabulary_size`，`embedding_dim`，`max_feature_count` 类型不匹配。
- **ValueError** - 如果 `init_vocabulary_size`，`embedding_dim`，`max_feature_count` 值小于等于 0，或 `init_vocabulary_size` 值大于 2147483647。
- **ValueError** - 如果大表个数超过 1024。
- **ValueError** - 如果 `optimizer` 不属于 `["adam", "adagrad", "adamw", "ftrl", "sgd", "rmsprop"]`。
- **TypeError** - 如果 `initializer` 不是 `EsInitializer` 对象，或者不属于 `["TruncatedNormal", "Uniform", "Constant"]`。

embedding_table_export (`file_path`, `trainable_var`)

导出每张大表及小表的 **embedding table** 信息。

说明：此功能只支持 **rank 0** 执行。

参数：

- **file_path** (str) - 存放 **table** 表的地址，最后一个字符不能为 `" / "`。

- **trainable_var** (list[parameter]) - 存放所有小表的 parameter 信息。

返回：

EmbeddingTableExport 算子的返回值，及小表的 embedding tale。

embedding_table_import (file_path)

导入 file_path 下所有的 embedding 文件。

参数：

- **file_path** (str) - 存放 table 表的地址，最后一个字符不能为 "/"。

返回：

EmbeddingTableImport 算子的返回值及小表导入结果。

embedding_variable_option (filter_option=None, padding_option=None, evict_option=None, completion_option=None, storage_option=None, feature_freezing_option=None, communication_option=None)

每个大表的所有属性配置合集。

参数：

- **filter_option** (CounterFilter, 可选) - filter 属性。默认值为 None。
- **padding_option** (PaddingParamsOption, 可选) - padding key 属性。默认值为 None。
- **evict_option** (EvictOption, 可选) - evict 属性。默认值为 None。
- **completion_option** (CompletionKeyOption, 可选) - completion key 属性。默认值为 None。
- **storage_option** (None, 可选) - 预留属性，当前不支持。默认值为 None。
- **feature_freezing_option** (None, 可选) - 预留属性，当前不支持。默认值为 None。
- **communication_option** (None, 可选) - 预留属性，当前不支持。默认值为 None。

返回：

EmbeddingVariableOption 对象，作为 `mindspore.experimental.es.EmbeddingService.embedding_init()` 的 `ev_option` 入参。

异常：

- **TypeError** - 如果 `filter_option` 不是 None 且类型不是 CounterFilter。
- **TypeError** - 如果 `padding_option` 不是 None 且类型不是 PaddingParamsOption。
- **TypeError** - 如果 `completion_option` 不是 None 且类型不是 CompletionKeyOption。
- **TypeError** - 如果 `evict_option` 不是 None 且类型不是 EvictOption。

evict_option (steps_to_live)

为每张大表设置被动淘汰策略。

参数：

- **steps_to_live** (int) - 特征 key 设置的淘汰存活步数。

返回：

EvictOption 对象。

异常：

- **TypeError** - 如果 `steps_to_live` 类型不是 int。
- **ValueError** - 如果 `steps_to_live` 不大于 0。

incremental_embedding_table_export (*file_path*)

导出每张大表进行 embedding table 的增量导出。

说明：此功能只支持 rank 0 执行。

参数：

- **file_path** (str) - 增量导出 table 表的存放地址，最后一个字符不能为 "/"。

返回：

EmbeddingTableExport 算子的返回值。

init_table ()

为小表初始化相应的参数。

返回：

dict，所有已初始化的表信息。其中 key 为每张小表的名字，value 为初始化的 parameter。

padding_param (*padding_key, mask=True, mask_zero=False*)

为每张大表初始化 padding key 策略。

参数：

- **padding_key** (int) - padding 的值，必须是真实合法的 hash key。
- **mask** (bool, 可选) - 是否更新 *padding_key*。默认值为 True，表示更新。
- **mask_zero** (bool, 可选) - 当 key 为 0 时，是否更新 padding key 值。默认值为 False，表示不更新。

返回：

PaddingParamsOption 对象。

异常：

- **TypeError** - 如果 *padding_key* 入参类型不是 int。
- **TypeError** - 如果 *mask* 入参类型不是 bool。

19.2.2 mindspore.experimental.es.EsEmbeddingLookup

class mindspore.experimental.es.**EsEmbeddingLookup** (*table_id, es_initializer, embedding_dim, max_key_num, optimizer_mode=None, optimizer_params=None, es_filter=None, es_padding_key=None, es_completion_key=None*)

提供大表特征值 keys 的查询功能。

警告：这是一个实验性的 EmbeddingService 接口。

参数：

- **table_id** (int) - table id 值。
- **es_initializer** (EsInitializer) - table_id 对应大表的 EsInitializer 对象，推理时可为 None。
- **embedding_dim** (int) - table_id 对应大表的插入数据的维度大小。

- **max_key_num** (int) - 查询的 keys 的数量。
- **optimizer_mode** (str) - 优化器类型。默认值为 `None`。
- **optimizer_params** (tuple[float]) - 优化器参数。默认值为 `None`。
- **es_filter** (CounterFilter) - table_id 对应大表的特征准入属性。默认值为 `None`。
- **es_padding_key** (PaddingParamsOption) - table_id 对应大表的 padding_key 属性。默认值为 `None`。
- **es_completion_key** (CompletionKeyOption) - table_id 对应大表的 completion_key 属性。默认值为 `None`。

输入：

- **keys** (Tensor) - 大表特征 keys。
- **actual_keys_input** (Tensor) - 大表特征 keys 的所有唯一元素组成的 Tensor。
- **unique_indices** (Tensor) - 大表特征 keys 每个元素在 actual_keys_input 中的索引值。
- **key_count** (Tensor) - actual_keys_input 每个元素在大表特征 keys 的计数值。

支持平台：

Atlas A2 训练系列产品

19.2.3 mindspore.experimental.es.ESEmbeddingSmallTableLookup

class mindspore.experimental.es.ESEmbeddingSmallTableLookup (*name, rank_id, rank_size, small_table_to_variable*)

提供小表特征值 keys 的查询功能。

警告：这是一个实验性的 EmbeddingService 接口。

参数：

- **name** (str) - 小表名字。
- **rank_id** (int) - 查询小表 key 时的 rank id。
- **rank_size** (int) - 查询小表 key 时的 rank size。
- **small_table_to_variable** (dict[str, parameter]) - 存储小表信息的 dict：key 为小表名称，value 为小表对应的参数信息。

输入：

- **ids_list** (Tensor) - 小表的每个特征 keys。

支持平台：

Atlas A2 训练系列产品

可用于 Cell 的构造函数的算子。
动态 shape 的支持情况详见 [算子动态 shape 支持情况](#)。
bfloat16 数据类型的支持情况详见 [支持列表](#)。
算子级并行过程各算子的使用约束详见 [算子级并行使用约束](#)。
模块导入方法如下：

```
import mindspore.ops as ops
```

MindSpore 中 `mindspore.ops.primitive` 接口与上一版本相比，新增、删除和支持平台的变化信息请参考 [mindspore.ops.primitive API 接口变更](#)。

20.1 算子原语

<code>mindspore.ops.Primitive</code>	Primitive 是 Python 中算子原语的基类。
<code>mindspore.ops.PrimitiveWithCheck</code>	PrimitiveWithCheck 是 Python 中原语的基类，定义了检查算子输入参数的函数，但是使用了 C++ 源码中注册的推理方法。
<code>mindspore.ops.PrimitiveWithInfer</code>	PrimitiveWithInfer 是 Python 中的原语基类，在 python 中定义了跟踪推理的函数。

20.1.1 mindspore.ops.Primitive

class mindspore.ops.Primitive(name)

Primitive 是 Python 中算子原语的基类。

参数:

- **name** (str) - 当前 Primitive 的名称。

样例:

```
>>> from mindspore.ops import prim_attr_register, Primitive
>>> add = Primitive('add')
>>>
>>> # or work with prim_attr_register:
>>> # init a Primitive class with attr1 and attr2
>>> class Add(Primitive):
...     @prim_attr_register
...     def __init__(self, attr1, attr2):
...         '''init for add'''
...         # check attr1 and attr2 or do some initializations
...         # init a Primitive obj with attr1=1 and attr2=2
>>> add = Add(attr1=1, attr2=2)
```

add_prim_attr(name, value)

添加 Primitive 的属性。

参数:

- **name** (str) - 属性名称。
- **value** (Any) - 属性值。

样例:

```
>>> from mindspore import ops
>>> a = ops.Add()
>>> a = a.add_prim_attr("attr",1)
>>> out = a.attrs["attr"]
>>> print(out)
1
```

check_elim(*args)

检查是否能在编译阶段计算 Primitive，如果能，则可以消除。有需要的子类可以重写该方法。

参数:

- **args** (Primitive args) - 与当前 Primitive 的参数相同。

返回:

由两个元素组成的元组。第一个元素是指是否能在编译阶段计算 Primitive，第二个元素是计算结果。

样例:

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.ops import prim_attr_register, Primitive
>>> class AddN(Primitive):
...     @prim_attr_register
```

(下页继续)

(续上页)

```

...     def __init__(self):
...         self.init_prim_io_names(inputs=["inputs"], outputs=["sum"])
...     def check_elim(self, inputs):
...         if len(inputs) != 1:
...             return (False, None)
...         if isinstance(inputs[0], Tensor):
...             return (True, inputs[0])
...
>>> addn = AddN()
>>> input_x = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> output = addn.check_elim((input_x,))
>>> print(output)
(True, Tensor(shape=[3], dtype=Float32, value= [ 1.00000000e+00,  2.00000000e+00,  3.
↪00000000e+00]))

```

del_prim_attr(name)
删除 Primitive 的属性。

参数:

- **name** (str) - 属性名称。

样例:

```

>>> from mindspore import ops
>>> a = ops.Add()
>>> a = a.add_prim_attr("attr", 1)
>>> a = a.del_prim_attr("attr")
>>> print(a.attrs)
{}

```

init_prim_io_names(inputs, outputs)
初始化 Tensor 或属性的输入输出的名称。

参数:

- **inputs** (list[str]) - 输入名称的列表。
- **outputs** (list[str]) - 输出名称的列表。

样例:

```

>>> from mindspore import ops
>>> a = ops.Add()
>>> a.init_prim_io_names(["x", "y"], ["sum"])
>>> print(a.input_names)
['x', 'y']
>>> print(a.output_names)
['sum']

```

place(role, rank_id)

设置 Primitive 算子标签。让此算子在对对应进程上执行。每个进程的标签都由进程角色 *role* 和 *rank_id* 组成，因此，对算子设置标签代表着算子在哪个进程执行，从而执行分布式训练等任务。

说明:

- 此接口只在成功调用 *mindspore.communication.init()* 完成动态组网后才能生效。

参数:

- **role** (str) - 算子执行所在进程的角色。只支持'MS_WORKER'。
- **rank_id** (int) - 算子执行所在进程的 ID。在相同角色进程间, *rank_id* 是唯一的。

样例:

```
>>> from mindspore import context
>>> from mindspore import ops
>>> context.set_context(mode=context.GRAPH_MODE)
>>> matmul = ops.MatMul()
>>> matmul.place('MS_WORKER', 0)
```

recompute (*mode=True*)

设置 Primitive 的重计算属性。

如果有一个被设置了重计算属性的 Primitive, 并且其结果在计算导数的时候被使用, 那么不会保存该 Primitive 在前向网络中的中间计算结果, 而是在自动微分的时候重新进行计算。

说明:

- 如果计算涉及随机化或全局变量, 则暂无法保证等效性。
- 在 PyNative 模式下不支持。

参数:

- **mode** (bool) - Primitive 是否设置了重计算。默认值: True。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor, ops, nn
>>> class NetRecompute(nn.Cell):
...     def __init__(self):
...         super(NetRecompute, self).__init__()
...         self.relu = ops.ReLU().recompute()
...         self.sqrt = ops.Sqrt()
...     def construct(self, x):
...         out = self.relu(x)
...         return self.sqrt(out)
...
>>> class GradNet(nn.Cell):
...     def __init__(self, network):
...         super(GradNet, self).__init__()
...         self.network = network
...         self.grad = ops.GradOperation()
...     def construct(self, x):
...         g_out = self.grad(self.network)(x)
...         return g_out
...
>>> x = Tensor(np.array([-1, 1]).astype(np.float32))
>>> net = NetRecompute()
>>> grad = GradNet(net)
>>> a = grad(x)
```

(下页继续)

(续上页)

```
>>> print(a)
[0. 0.5]
```

set_device(device_target)

设置 Primitive 执行后端。

参数:

- **device_target** (str) - 后端名称, 支持 CPU、GPU、Ascend。

样例:

```
>>> from mindspore import ops
>>> a = ops.Add()
>>> a = a.set_device("GPU")
>>> print(a.primitive_target)
GPU
```

set_prim_instance_name(instance_name)

设置 Primitive 算子的实例的名称。

说明: 当用户定义 Primitive 算子时, 默认调用它。

参数:

- **instance_name** (str) - 用户设置的 Primitive 算子的实例的名称。

样例:

```
>>> from mindspore import ops
>>> a = ops.Add()
>>> a = a.set_prim_instance_name("add")
>>> print(a.instance_name)
add
```

set_stage(stage)

将 stage 的 ID 添加到 Primitive 属性中。

说明: 仅在半自动并行模式下有效。在其他并行模式下, 请将其设置为 0。

参数:

- **stage** (int) - 当前 stage 的 ID。

样例:

```
>>> from mindspore import ops
>>> add = ops.Add()
>>> print(add.set_stage(0))
Prim[Add]<stage=0>
```

shard(in_strategy=None, out_strategy=None)

将切分策略添加到 Primitive 属性中。

说明：仅在半自动并行或自动并行模式下有效。在其他并行模式中，将忽略此处设置的策略。

参数：

- **in_strategy** (tuple) - 描述算子输入的切分策略。默认值：None。
- **out_strategy** (tuple) - 描述算子输出的切分策略，仅针对某些算子，如 **MatMul**。默认值：None。

样例：

```
>>> from mindspore import ops
>>> add = ops.Add()
>>> print(add.shard(((1, 1), (1, 1))))
Prim[Add]<in_strategy=((1, 1), (1, 1)), out_strategy=None>
>>> # using layout
>>> from mindspore import Layout
>>> layout = Layout((2, 2, 2), ("dp", "sp", "mp"))
>>> layout_tuple = (layout("dp", "sp"), layout("sp", "mp"))
>>> from mindspore import ops
>>> matmul = ops.MatMul()
>>> print(matmul.shard(layout_tuple))
Prim[MatMul]<in_layout=({'device_matrix': (2, 2, 2), 'tensor_map': (2, 1)},
{'device_matrix': (2, 2, 2), 'tensor_map': (1, 0)})>
>>> # using layout with None
>>> from mindspore import Layout
>>> layout = Layout((2, 2, 2), ("dp", "sp", "mp"))
>>> layout_tuple = (layout("dp", "sp"), layout("sp", "None")) # "None" means the axis_
↳ would not be split
>>> from mindspore import ops
>>> matmul = ops.MatMul()
>>> print(matmul.shard(layout_tuple))
Prim[MatMul]<in_layout=({'device_matrix': (2, 2, 2), 'tensor_map': (2, 1)},
{'device_matrix': (2, 2, 2), 'tensor_map': (1, -1)})>
```

property update_parameter

判断此 Primitive 是否会更新参数的值。

20.1.2 mindspore.ops.PrimitiveWithCheck

class mindspore.ops.PrimitiveWithCheck (name)

PrimitiveWithCheck 是 Python 中原语的基类，定义了检查算子输入参数的函数，但是使用了 C++ 源码中注册的推理方法。

可以重写三个方法来定义 Primitive 的检查逻辑：__check__()、check_shape() 和 check_dtype()。如果在 Primitive 中定义了 __check__()，则 __check__() 的优先级最高。

如果未定义 __check__()，则可以定义 check_shape() 和 check_dtype() 来描述形状和类型的检查逻辑。可以定义 infer_value() 方法（如 PrimitiveWithInfer），用于常量传播。

了解更多如何自定义算子，请查看 [自定义算子](#)。

参数：

- **name** (str) - 当前 Primitive 的名称。

支持平台：

Ascend GPU CPU

样例:

```
>>> from mindspore import dtype as mstype
>>> from mindspore.ops import prim_attr_register, PrimitiveWithCheck
>>> # init a Primitive class with check
>>> class Flatten(PrimitiveWithCheck):
...     @prim_attr_register
...     def __init__(self):
...         pass
...     def check_shape(self, input_x):
...         Validator.check_int(len(input_x), 1, validator.GE, 'input_x rank', self.name)
...
...     def check_dtype(self, input_x):
...         Validator.check_subclass("input_x", input_x, mstype.tensor_type, self.name)
...
>>> # init a Primitive obj
>>> add = Flatten()
```

check_dtype (*args)

检查输入参数的数据类型。

参数:

- **args** (*mindspore.dtype*) - 输入的数据类型。

返回:

None。

check_shape (*args)

检查输入参数的 shape。

说明: Scalar 的 shape 是一个空元组。

参数:

- **args** (tuple(int)) - 输入 Tensor 的 shape。

返回:

None。

20.1.3 mindspore.ops.PrimitiveWithInfer

class mindspore.ops.PrimitiveWithInfer (name)

PrimitiveWithInfer 是 Python 中的原语基类，在 python 中定义了跟踪推理的函数。

可以重写四个方法来定义 Primitive 的推断逻辑: `__infer__()`、`infer_shape()`、`infer_dtype()` 和 `infer_value()`。如果在 Primitive 中定义了 `__infer__()`，则 `__infer__()` 的优先级最高。

如果未定义 `__infer__()`，则可以定义 `infer_shape()` 和 `infer_dtype()` 来描述 shape 和类型的推断逻辑。`infer_value()` 用于常量传播。

关于如何自定义算子，请查看 [自定义算子](#)。

参数:

- **name** (str) - 当前 Primitive 的名称。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore.ops import prim_attr_register, PrimitiveWithInfer
>>> # init a Primitive class with infer
>>> class Add(PrimitiveWithInfer):
...     @prim_attr_register
...     def __init__(self):
...         pass
...
...     def infer_shape(self, x, y):
...         return x # output shape same as first input 'x'
...
...     def infer_dtype(self, x, y):
...         return x # output type same as first input 'x'
...
>>> # init a Primitive obj
>>> add = Add()
```

20.2 神经网络层算子

20.2.1 神经网络

接口名	概述	支持平台
<code>mindspore.ops.AvgPool</code>	对输入的多维数据进行二维平均池化运算。	Ascend GPU CPU
<code>mindspore.ops.AvgPool3D</code>	对输入的多维数据进行三维的平均池化运算。	Ascend GPU CPU
<code>mindspore.ops.BatchNorm</code>	对输入数据进行归一化 (Batch Normalization) 和更新参数。	Ascend GPU CPU
<code>mindspore.ops.Conv2D</code>	二维卷积层。	Ascend GPU CPU
<code>mindspore.ops.Conv2DTranspose</code>	计算二维转置卷积，也称为反卷积，实际不是真正的反卷积。	Ascend GPU CPU
<code>mindspore.ops.Conv3D</code>	三维卷积层。	Ascend GPU CPU
<code>mindspore.ops.Conv3DTranspose</code>	计算三维转置卷积，也称为反卷积（实际不是真正的反卷积）。	Ascend GPU CPU
<code>mindspore.ops.CTCGreedyDecoder</code>	对输入中给定的 logits 执行贪婪解码。	Ascend GPU CPU
<code>mindspore.ops.Dense</code>	更多详情请查看: <code>mindspore.ops.dense()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Dropout</code>	Dropout 是一种正则化手段，通过在训练中以 $1 - keep\_prob$ 的概率随机将神经元输出设置为 0，起到减少神经元相关性的作用，避免过拟合。	Ascend GPU CPU
<code>mindspore.ops.Dropout2D</code>	在训练期间，根据概率 $1 - keep\_prob$ ，随机地将一些通道设置为 0，且服从伯努利分布。	Ascend GPU CPU
<code>mindspore.ops.Dropout3D</code>	在训练期间，以服从伯努利分布的概率 $1 - keep\_prob$ 随机将输入 Tensor 的某些通道归零。	Ascend GPU CPU
<code>mindspore.ops.DynamicGRUV2</code>	为输入序列应用一个单层 GRU(gated recurrent unit)。	Ascend

下页继续

表 2 - 续上页

<code>mindspore.ops.DynamicRNN</code>	将循环神经网络应用到输入上。	Ascend
<code>mindspore.ops.EmbeddingLookup</code>	根据指定的索引, 返回输入 Tensor 的切片。	Ascend GPU CPU
<code>mindspore.ops.Flatten</code>	扁平化 (Flatten) 输入 Tensor , 不改变 0 轴的 size 。	Ascend GPU CPU
<code>mindspore.ops.FractionalMaxPool3DWithFixedKsize</code>	此运算对由多个输入平面组成的输入信号进行 3D 分数最大池化。	Ascend GPU CPU
<code>mindspore.ops.GridSampler2D</code>	此操作使用基于流场网格的插值对 2D <i>input_x</i> 进行采样, 该插值通常由 <code>mindspore.ops.affine_grid()</code> 生成。	Ascend GPU CPU
<code>mindspore.ops.GridSampler3D</code>	给定一个输入和一个网格, 使用网格中的输入值和像素位置计算输出。	Ascend GPU CPU
<code>mindspore.ops.LayerNorm</code>	在输入 Tensor 上应用层归一化 (Layer Normalization)。	Ascend GPU CPU
<code>mindspore.ops.LRN</code>	局部响应归一化操作 LRN (Local Response Normalization)。	GPU CPU
<code>mindspore.ops.LSTM</code>	对输入执行长短期记忆 (LSTM) 网络。	GPU CPU
<code>mindspore.ops.MaxPool</code>	对输入的多维数据进行二维的最大池化运算。	Ascend GPU CPU
<code>mindspore.ops.MaxPool3D</code>	对输入的多维数据进行三维的最大池化运算。	Ascend GPU CPU
<code>mindspore.ops.MaxPool3DWithArgmax</code>	三维最大值池化, 返回最大值结果及其索引值。	Ascend GPU CPU
<code>mindspore.ops.MaxUnpool2D</code>	MaxPool2D 的逆过程。	Ascend GPU CPU
<code>mindspore.ops.MaxUnpool3D</code>	<code>mindspore.ops.MaxPool3D</code> 的逆过程。	Ascend GPU CPU
<code>mindspore.ops.MirrorPad</code>	通过指定的填充模式和大小对输入 Tensor 进行填充。	Ascend GPU CPU
<code>mindspore.ops.Pad</code>	根据参数 <i>paddings</i> 对输入进行填充。	Ascend GPU CPU
<code>mindspore.ops.Padding</code>	将输入 Tensor 的最后一个维度从 1 扩展到 <i>pad_dim_size</i> , 其填充值为 0。	Ascend GPU CPU
<code>mindspore.ops.ResizeBicubic</code>	使用双三次插值调整图像大小到指定的大小。	Ascend GPU CPU
<code>mindspore.ops.ResizeNearestNeighbor</code>	使用最近邻插值算法调整输入 Tensor 为指定大小。	Ascend GPU CPU
<code>mindspore.ops.UpsampleNearest3D</code>	执行最近邻上采样操作。	Ascend GPU CPU
<code>mindspore.ops.UpsampleTrilinear3D</code>	输入为五维度 Tensor , 跨其中三维执行三线性插值上调采样。	Ascend GPU CPU

mindspore.ops.AvgPool

class mindspore.ops.**AvgPool** (*kernel_size=1, strides=1, pad_mode='VALID', data_format='NCHW'*)

对输入的多维数据进行二维平均池化运算。

更多参考详见 `mindspore.ops.avg_pool2d()`。

参数:

- **kernel_size** (`Union[int, tuple[int]]`, 可选) - 指定池化核尺寸大小, 可以是单个整数表示池化核高度和宽度, 或者整数 `tuple` 分别表示池化核高度和宽度。默认值: 1。
- **strides** (`Union[int, tuple[int]]`, 可选) - 池化操作的移动步长, 可以是单个整数表示高度和宽度方向的移动步长, 或者整数 `tuple` 分别表示高度和宽度方向的移动步长。默认值: 1。
- **pad_mode** (`str`, 可选) - 指定填充模式, 填充值为 0。可选值为 "SAME" 或 "VALID"。默认值: "VALID"。
 - "SAME": 在输入的四周围填充, 使得当 *stride* 为 1 时, 输入和输出的 **shape** 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在底部/右侧。

- "VALID": 不对输入进行填充, 返回输出可能的最大高度和宽度, 不能构成一个完整 stride 的额外的像素将被丢弃。

- **data_format** (str, 可选) - 指定输入和输出的数据格式。取值为 'NHWC' 或 'NCHW'。默认值: 'NCHW'。

输入:

- **x** (Tensor) - 输入 shape 为 $(N, C_{in}, H_{in}, W_{in})$ 的 Tensor。支持的数据类型: float16、float32、float64。

输出:

Tensor, shape 为 $(N, C_{out}, H_{out}, W_{out})$ 。

异常:

- **TypeError** - *kernel_size* 或 *strides* 既不是 int 也不是 tuple。
- **TypeError** - *x* 的数据类型不是 float16、float32 或 float64。
- **ValueError** - *pad_mode* 既不是 'VALID', 也不是 'SAME', 不区分大小写。
- **ValueError** - *data_format* 既不是 'NCHW' 也不是 'NHWC'。
- **ValueError** - *kernel_size* 或 *strides* 小于 1。
- **ValueError** - *x* 的 shape 长度不等于 4。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, nn
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.avgpool_op = ops.AvgPool(pad_mode='VALID', kernel_size=2, strides=1)
...
...     def construct(self, x):
...         result = self.avgpool_op(x)
...         return result
...
>>> x = Tensor(np.arange(1 * 3 * 3 * 4).reshape(1, 3, 3, 4), mindspore.float32)
>>> net = Net()
>>> output = net(x)
>>> print(output)
[[[ [ 2.5  3.5  4.5]
    [ 6.5  7.5  8.5]]
  [14.5 15.5 16.5]
  [18.5 19.5 20.5]]
 [26.5 27.5 28.5]
 [30.5 31.5 32.5]]]
```

mindspore.ops.AvgPool3D

```
class mindspore.ops.AvgPool3D(kernel_size=1, strides=1, pad_mode='valid', pad=0, ceil_mode=False,
                               count_include_pad=True, divisor_override=0, data_format='NCDHW')
```

对输入的多维数据进行三维的平均池化运算。

一般情况下，当输入 shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 时，AvgPool3D 会在 (D_{in}, H_{in}, W_{in}) 维度上输出区域平均值。给定 $kernel\_size$ 为 $ks = (d_{ker}, h_{ker}, w_{ker})$ ，给定 $stride$ 为 $s = (s_0, s_1, s_2)$ ，运算如下：

$$\text{output}(N_i, C_j, d, h, w) = \frac{1}{d_{ker} * h_{ker} * w_{ker}} \sum_{l=0}^{d_{ker}-1} \sum_{m=0}^{h_{ker}-1} \sum_{n=0}^{w_{ker}-1} \text{input}(N_i, C_j, s_0 \times d + l, s_1 \times h + m, s_2 \times w + n)$$

说明：该接口暂不支持 Atlas A2 训练系列产品。

参数：

- **kernel_size** (Union[int, tuple[int]], 可选) - 指定池化核尺寸大小。可以是一个整数，其值同时对应深度、高度和宽度；或者是一个 tuple，含 3 个整数，分别对应深度、高度和宽度。默认值：1，取值范围为 [1, 255]。
- **strides** (Union[int, tuple[int]], 可选) - 池化操作的移动步长。可以是一个整数，其值同时对应移动深度、高度和宽度；或者是一个 tuple，含 3 个整数，分别对应移动深度、高度和宽度。默认值：1，取值范围为 [1, 63]。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："valid"。
 - "same"：在输入的深度、高度和宽度维度进行填充，使得当 $stride$ 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在前方/底部/右侧。如果设置了此模式， pad 必须为 0。
 - "valid"：不对输入进行填充，返回输出可能的最大深度、高度和宽度，不能构成一个完整 $stride$ 的额外的像素将被丢弃。如果设置了此模式， pad 必须为 0。
 - "pad"：对输入填充指定的量。在这种模式下，在输入的深度、高度和宽度方向上填充的量由 pad 参数指定。如果设置此模式， pad 必须大于或等于 0。
- **pad** (Union(int, tuple[int], list[int]), 可选) - 池化填充方式。默认值：0。如果 pad 是一个整数，则头部、尾部、顶部、底部、左边和右边的填充都是相同的，等于 pad 。如果 pad 是六个 integer 的 tuple，则头部、尾部、顶部、底部、左边和右边的填充分别等于填充 $pad[0]$ 、 $pad[1]$ 、 $pad[2]$ 、 $pad[3]$ 、 $pad[4]$ 和 $pad[5]$ 。
- **ceil_mode** (bool, 可选) - 是否使用 ceil 函数计算输出高度和宽度。默认值：False。
- **count_include_pad** (bool, 可选) - 如果为 True，平均计算将包括零填充。默认值：True。
- **divisor_override** (int, 可选) - 如果指定了该值，它将在平均计算中用作除数，否则将使用 $kernel\_size$ 作为除数。默认值：0。
- **data_format** (str, 可选) - 输入和输出的数据格式。目前仅支持 'NCDHW'。默认值："NCDHW"。

输入：

- **x** (Tensor) - shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 的 Tensor。数据类型为 float16、float32 和 float64。

输出：

Tensor，其 shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ ，数据类型与 x 相同。

异常：

- **TypeError** - $kernel\_size$ 、 $strides$ 或 pad 既不是 int 也不是 tuple。
- **TypeError** - $ceil\_mode$ 或 $count\_include\_pad$ 不是 bool。

- **TypeError** - *pad_mode* 或 *data_format* 不是 string。
- **TypeError** - *divisor_override* 不是 int。
- **ValueError** - *kernel_size* 或 *strides* 中的数字不是正数。
- **ValueError** - *kernel_size* 或 *strides* 是长度不等于 3 的 tuple。
- **ValueError** - *pad_mode* 不是 'same', 'valid', 或 'pad'。
- **ValueError** - *pad* 是长度不等于 6 的 tuple。
- **ValueError** - *pad* 的元素小于 0。
- **ValueError** - *pad_mode* 不等于 'pad' 且 *pad* 不等于 0 或 (0, 0, 0, 0, 0)。
- **ValueError** - *data_format* 不是 'NCDHW'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> x = Tensor(np.arange(1 * 2 * 2 * 2 * 3).reshape((1, 2, 2, 2, 3)), mindspore.float16)
>>> avg_pool3d = ops.AvgPool3D(kernel_size=2, strides=1, pad_mode="valid")
>>> output = avg_pool3d(x)
>>> print(output)
[[[[[ 5.  6.]]]
  [[17. 18.]]]]]
```

mindspore.ops.BatchNorm

class mindspore.ops.**BatchNorm** (*is_training=False, epsilon=1e-5, momentum=0.1, data_format='NCHW'*)

对输入数据进行归一化 (Batch Normalization) 和更新参数。

批量归一化广泛应用于卷积神经网络中。此运算对输入应用归一化，避免内部协变量偏移，详见论文 [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#)。使用 mini-batch 数据和学习参数进行训练，学习的参数见如下公式中，

$$y = \frac{x - \text{mean}}{\sqrt{\text{variance} + \epsilon}} * \gamma + \beta$$

其中， γ 为 *scale*， β 为 *bias*， ϵ 为 *epsilon*，*mean* 为 *x* 的均值，*variance* 为 *x* 的方差。

警告：

- 如果该运算用于推理，并且输出 "reserve_space_1" 和 "reserve_space_2" 可用，则 "reserve_space_1" 的值与 "mean" 相同，"reserve_space_2" 的值与 "variance" 相同。
- 对于 Ascend 310，由于平方根指令，结果精度未能达到 1%。

参数：

- **is_training** (bool, 可选) - 如果 *is_training* 为 True，则在训练期间计算 *mean* 和 *variance*。如果 *is_training* 为 False，则在推理期间从 checkpoint 加载。默认值：False。

- **epsilon** (float, 可选) - 添加到分母上的值, 以确保数值稳定性。默认值: $1e-5$, 取值范围为 (0, 1]。
- **momentum** (float, 可选) - 动态均值和动态方差所使用的动量。(例如 $new\_running\_mean = (1 - momentum) * running\_mean + momentum * current\_mean$)。动量值必须为 [0, 1]。默认值: 0.1。
- **data_format** (str, 可选) - 输入数据格式, 可选值有: 'NHWC' 或 'NCHW', 'NHWC' 仅在 GPU 上支持。默认值: 'NCHW'。

输入:

- **input_x** (Tensor) - 数据输入, shape 为 (N, C) 的 Tensor, 数据类型为 float16 或 float32。
- **scale** (Union[Parameter, Tensor]) - 输入 Scalar, shape 为 $(C,)$ 的 Tensor 或 Parameter, 数据类型为 float16 或 float32。
- **bias** (Union[Parameter, Tensor]) - 输入偏置项, shape 为 $(C,)$ 的 Tensor 或 Parameter, 具有与 *scale* 相同的数据类型。
- **mean** (Union[Parameter, Tensor]) - 输入均值, shape 为 $(C,)$ 的 Tensor 或 Parameter, 具有与 *scale* 相同的数据类型。
- **variance** (Union[Parameter, Tensor]) - 输入方差, shape 为 $(C,)$ 的 Tensor 或 Parameter, 具有与 *scale* 相同的数据类型。

输出:

5 个 Tensor 组成的 tuple、归一化输入和更新的参数。

- **output_x** (Tensor) - 数据类型和 shape 与输入 *input_x* 相同。shape 为 (N, C) 。
- **batch_mean** (Tensor) - 输入的均值, shape 为 $(C,)$ 的一维 Tensor。
- **batch_variance** (Tensor) - 输入的方差, shape 为 $(C,)$ 的一维 Tensor。
- **reserve_space_1** (Tensor) - 需要计算梯度时, 被重新使用的均值, shape 为 $(C,)$ 的一维 Tensor。
- **reserve_space_2** (Tensor) - 需要计算梯度时, 被重新使用的方差, shape 为 $(C,)$ 的一维 Tensor。

异常:

- **TypeError**: *is_training* 不是 bool。
- **TypeError**: *epsilon* 或 *momentum* 的数据类型不是 float。
- **TypeError**: *data_format* 不是 str。
- **TypeError**: *input_x*、*scale*、*bias*、*mean* 或 *variance* 不是 Tensor。
- **TypeError**: *input_x* 和 *scale* 的数据类型既不是 float16, 也不是 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.ones([2, 2]), mindspore.float32)
>>> scale = Tensor(np.ones([2]), mindspore.float32)
>>> bias = Tensor(np.ones([2]), mindspore.float32)
>>> mean = Tensor(np.ones([2]), mindspore.float32)
>>> variance = Tensor(np.ones([2]), mindspore.float32)
>>> batch_norm = ops.BatchNorm()
```

(下页继续)

(续上页)

```
>>> output = batch_norm(input_x, scale, bias, mean, variance)
>>> print(output[0])
[[1. 1.]
 [1. 1.]]
```

mindspore.ops.Conv2D

class mindspore.ops.Conv2D (out_channel, kernel_size, mode=1, pad_mode='valid', pad=0, stride=1, dilation=1, group=1, data_format='NCHW')

二维卷积层。

对输入 Tensor 计算二维卷积，通常输入的 shape 为 $(N, C_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C 为通道数， H 为特征图的高度， W 为特征图的宽度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中， bias 为输出偏置， ccor 为 [cross-correlation](#) 操作， weight 为卷积核的值， X 为输入的特征图。

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中， $\text{bias}(C_{out_j})$ 为第 j 个输出通道的偏置， $\text{weight}(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。卷积核 shape 为 $(\text{kernel\_size}[0], \text{kernel\_size}[1])$ ，其中 $\text{kernel\_size}[0]$ 和 $\text{kernel\_size}[1]$ 是卷积核的高度和宽度。若考虑到输入输出通道以及 group ，则完整卷积核的 shape 为 $(C_{out}, C_{in}/\text{group}, \text{kernel\_size}[0], \text{kernel\_size}[1])$ ，其中 group 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

说明：在 Ascend 平台上，目前只支持深度卷积场景下的分组卷积运算。也就是说，当 $\text{group} > 1$ 的场景下，必须要满足 $\text{in\_channels} = \text{out\_channels} = \text{group}$ 的约束条件。

参数：

- **out_channel** (int) - 指定输出通道数： C_{out} 。
- **kernel_size** (Union[int, tuple[int]]) - 指定二维卷积核的高度和宽度。可以为单个 int 或一个包含 2 个 int 组成的元组。单个整数表示该值同时适用于内核的高度和宽度。包含 2 个整数的元组表示第一个值用于高度，另一个值用于内核的宽度。
- **mode** (int, 可选) - 指定不同的卷积模式。此值目前未被使用。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："valid"。
 - "same"：在输入的四周填充，使得当 stride 为 1 时，输入和输出的 shape 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在底部/右侧。如果设置了此模式， pad 必须为 0。
 - "valid"：不对输入进行填充，返回输出可能的最大高度和宽度，不能构成一个完整 stride 的额外的像素将被丢弃。如果设置了此模式， pad 必须为 0。

- "pad": 对输入填充指定的量。在这种模式下, 在输入的高度和宽度方向上填充的量由 *pad* 参数指定。如果设置此模式, *pad* 必须大于或等于 0。
- **pad** (Union(int, tuple[int]), 可选) - 当 *pad_mode* 为 "pad" 时, 指定在输入 *x* 的高度和宽度方向上填充的数量。可以为单个 int 或包含四个 int 组成的 tuple。如果 *pad* 是单个 int, 那么上、下、左、右的填充量都等于 *pad*。如果 *pad* 是一个由四个 int 组成的 tuple, 那么上、下、左、右的填充分别等于 *pad*[0]、*pad*[1]、*pad*[2] 和 *pad*[3]。int 值应该要大于或等于 0, 默认值: 0。
- **stride** (Union(int, tuple[int]), 可选) - 卷积核移动的步长。可以为单个 int, 或由两个/四个 int 组成的 tuple。单个 int 表示在高度和宽度方向的移动步长均为该值。两个 int 组成的 tuple 分别表示在高度和宽度方向的移动步长。若为四个 int, N、C 两维度默认为 1, H、W 两维度分别对应高度和宽度上的步长。默认值: 1。
- **dilation** (Union(int, tuple[int]), 可选) - 卷积核膨胀尺寸。可以为单个 int, 或者由两个/四个 int 组成的 tuple。单个 int 表示在高度和宽度方向的膨胀尺寸均为该值。两个 int 组成的 tuple 分别表示在高度和宽度方向的膨胀尺寸。若为四个 int, N、C 两维度 int 默认为 1, H、W 两维度分别对应高度和宽度上的膨胀尺寸。假设 *dilation* = (*d0*, *d1*), 则卷积核在高度方向间隔 *d0* - 1 个元素进行采样, 在宽度方向间隔 *d1* - 1 个元素进行采样。高度和宽度上取值范围分别为 [1, H] 和 [1, W]。默认值: 1。
- **group** (int, 可选) - 分组卷积时在通道上分割输入 *x* 的组数。默认值: 1。
- **data_format** (str, 可选) - 数据格式的可选值有 "NHWC", "NCHW"。默认值: "NCHW"。(目前仅 GPU 支持 NHWC。)

输入:

- **x** (Tensor) - 输入 Tensor, shape 为 (*N*, *C_{in}*, *H_{in}*, *W_{in}*) 或者 (*N*, *H_{in}*, *W_{in}*, *C_{in}*), 具体哪种取决于 *data_format*。
- **weight** (Tensor) - 卷积核的值, 其 shape 应为 (*C_{out}*, *C_{in}*/group, kernel_size[0], kernel_size[1])。

输出:

Tensor, 卷积操作后的值。shape 为 (*N*, *C_{out}*, *H_{out}*, *W_{out}*) 或 (*N*, *H_{out}*, *W_{out}*, *C_{out}*)。要了解不同的填充模式如何影响输出 shape, 请参考 [mindspore.nn.Conv2d](#) 以获取更多详细信息。

异常:

- **TypeError** - *kernel_size*、*stride*、*pad* 或 *dilation* 既不是 int 也不是 tuple。
- **TypeError** - *out_channel* 或 *group* 不是 int。
- **ValueError** - *kernel_size*、*stride* 或 *dilation* 小于 1。
- **ValueError** - *pad_mode* 不是 "same"、"valid" 或 "pad"。
- **ValueError** - *pad* 是一个长度不等于 4 的 tuple。
- **ValueError** - *pad_mode* 不等于 "pad", *pad* 不等于 (0, 0, 0, 0)。
- **ValueError** - *data_format* 既不是 "NCHW", 也不是 "NHWC"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1: All parameters use default values.
>>> x = Tensor(np.ones([10, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> conv2d = ops.Conv2D(out_channel=32, kernel_size=3)
>>> output = conv2d(x, weight)
```

(下页继续)

(续上页)

```

>>> print(output.shape)
(10, 32, 30, 30)
>>> # case 2: pad_mode="pad", other parameters being default.
>>> x = Tensor(np.ones([10, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> conv2d = ops.Conv2D(out_channel=32, kernel_size=3, pad_mode="pad", pad=(4, 10, 4, 10))
>>> output = conv2d(x, weight)
>>> print(output.shape)
(10, 32, 44, 44)
>>> # case 3: stride=(2, 4), other parameters being default.
>>> x = Tensor(np.ones([10, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> conv2d = ops.Conv2D(out_channel=32, kernel_size=3, stride=(2, 4))
>>> output = conv2d(x, weight)
>>> print(output.shape)
(10, 32, 15, 8)
>>> # case 4: dilation=2, other parameters being default.
>>> x = Tensor(np.ones([10, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> conv2d = ops.Conv2D(out_channel=32, kernel_size=3, dilation=2)
>>> output = conv2d(x, weight)
>>> print(output.shape)
(10, 32, 28, 28)
>>> # case 5: group=2, other parameters being default.
>>> x = Tensor(np.ones([10, 64, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> conv2d = ops.Conv2D(out_channel=32, kernel_size=3, group=2)
>>> output = conv2d(x, weight)
>>> print(output.shape)
(10, 32, 30, 30)
>>> # case 6: All parameters are specified.
>>> x = Tensor(np.ones([10, 64, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> conv2d = ops.Conv2D(out_channel=32, kernel_size=3, pad_mode="pad",
...                    pad=(4, 10, 4, 10), stride=(2, 4), dilation=2, group=2)
>>> output = conv2d(x, weight)
>>> print(output.shape)
(10, 32, 21, 11)

```

mindspore.ops.Conv2DTranspose

class mindspore.ops.Conv2DTranspose(out_channel, kernel_size, pad_mode='valid', pad=0, pad_list=None, mode=1, stride=1, dilation=1, group=1, data_format='NCHW')

计算二维转置卷积，也称为反卷积，实际不是真正的反卷积。因为它不能完全的恢复输入矩阵的数据，但能恢复输入矩阵的形状。

参数：

- **out_channel** (int) - 输出的通道数。
- **kernel_size** (Union[int, tuple[int]]) - 卷积核的大小。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same", "valid" 或 "pad"。默认值: "valid"。
 - "same": 在输入的四周填充，使得当 *stride* 为 1 时，输入和输出的 **shape** 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在底部/右侧。如果设置了此

模式, *pad* 必须为 0。

- "valid": 不对输入进行填充, 返回输出可能的最大高度和宽度, 不能构成一个完整 *stride* 的额外的像素将被丢弃。如果设置了此模式, *pad* 必须为 0。
- "pad": 对输入填充指定的量。在这种模式下, 在输入的高度和宽度方向上填充的量由 *pad* 参数指定。如果设置此模式, *pad* 必须大于或等于 0。请参考 [mindspore.nn.Conv2dTranspose](#) 了解更多 *pad_mode* 的使用规则。
- **pad** (Union[int, tuple[int]], 可选) - 指定要填充的填充值。默认值: 0。如果 *pad* 是整数, 则顶部、底部、左侧和右侧的填充都等于 *pad*。如果 *pad* 是四个整数的 tuple, 则顶部、底部、左侧和右侧的填充分别等于 *pad*[0]、*pad*[1]、*pad*[2] 和 *pad*[3]。
- **pad_list** (Union[str, None], 可选) - 卷积填充方式, 如 (顶部、底部、左侧、右侧)。默认值: None, 表示不使用此参数。
- **mode** (int, 可选) - 指定不同的卷积模式。当前未使用该值。默认值: 1。
- **stride** (Union[int, tuple[int]], 可选) - 卷积核移动的步长。默认值: 1。
- **dilation** (Union[int, tuple[int]], 可选) - 卷积核膨胀尺寸。默认值: 1。
- **group** (int, 可选) - 将过滤器拆分为组。默认值: 1。
- **data_format** (str, 可选) - 输入和输出的数据格式。它应该是 "NHWC" 或 "NCHW", 默认值是 "NCHW"。

输入:

- **dout** (Tensor) - 卷积操作的输出的梯度 Tensor。shape: $(N, C_{out}, H_{out}, W_{out})$ 。
- **weight** (Tensor) - 设置卷积核大小为 (K_1, K_2) , 然后 shape 为 $(C_{out}, C_{in}, K_1, K_2)$ 。
- **input_size** (Tensor) - 输入的 shape, shape 的格式为 $(N, C_{in}, H_{in}, W_{in})$ 。

输出:

Tensor, 卷积操作的输入的梯度 Tensor。它的 shape 与输入相同。

异常:

- **TypeError** - 如果 *kernel_size*、*stride*、*pad* 或 *dilation* 既不是 int 也不是 tuple。
- **TypeError** - 如果 *out_channel* 或 *group* 不是 int。
- **ValueError** - 如果 *kernel_size*、*stride* 或 *dilation* 小于 1。
- **ValueError** - 如果 *pad_mode* 不是 "same"、"valid" 或 "pad"。
- **ValueError** - 如果 *padding* 是长度不等于 4 的 tuple。
- **ValueError** - 如果 *pad_mode* 不等于 "pad", *pad* 不等于 (0,0,0,0)。
- **ValueError** - 如果 *data_format* 既不是 "NCHW" 也不是 "NHWC"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> dout = Tensor(np.ones([10, 32, 30, 30]), mindspore.float32)
>>> weight = Tensor(np.ones([32, 32, 3, 3]), mindspore.float32)
>>> x = Tensor(np.ones([10, 32, 32, 32]))
>>> conv2d_transpose_input = ops.Conv2DTranspose(out_channel=32, kernel_size=3)
```

(下页继续)

(续上页)

```
>>> output = conv2d_transpose_input(dout, weight, ops.shape(x))
>>> print(output.shape)
(10, 32, 32, 32)
```

mindspore.ops.Conv3D

class mindspore.ops.Conv3D (out_channel, kernel_size, mode=1, pad_mode='valid', pad=0, stride=1, dilation=1, group=1, data_format='NCDHW')

三维卷积层。

对输入 Tensor 计算三维卷积。通常，输入 Tensor 的 shape 为 $(N, C_{in}, D_{in}, H_{in}, W_{in})$ ，其中 N 为 batch size， C 为通道数， D, H, W 分别为特征图的深度、高度和宽度。

根据以下公式计算输出：

$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{ccor}(\text{weight}(C_{out_j}, k), X(N_i, k))$$

其中， bias 为输出偏置， ccor 为 [cross-correlation](#) 操作， weight 为卷积核的值， X 为输入的特征图。

- i 对应 batch 数，其范围为 $[0, N - 1]$ ，其中 N 为输入 batch。
- j 对应输出通道，其范围为 $[0, C_{out} - 1]$ ，其中 C_{out} 为输出通道数，该值也等于卷积核的个数。
- k 对应输入通道数，其范围为 $[0, C_{in} - 1]$ ，其中 C_{in} 为输入通道数，该值也等于卷积核的通道数。

因此，上面的公式中， $\text{bias}(C_{out_j})$ 为第 j 个输出通道的偏置， $\text{weight}(C_{out_j}, k)$ 表示第 j 个卷积核在第 k 个输入通道的卷积核切片， $X(N_i, k)$ 为特征图第 i 个 batch 第 k 个输入通道的切片。

卷积核 shape 为 $(\text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ ，其中 $\text{kernel\_size}[0]$ 、 $\text{kernel\_size}[1]$ 和 $\text{kernel\_size}[2]$ 分别是卷积核的深度、高度和宽度。若考虑到输入输出通道以及 group ，则完整卷积核的 shape 为 $(C_{out}, C_{in}/\text{group}, \text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])$ ，其中 group 是分组卷积时在通道上分割输入 x 的组数。

想更深入了解卷积层，请参考论文 [Gradient Based Learning Applied to Document Recognition](#)。

说明：

1. 在 Ascend 平台上，目前只支持 $\text{groups} = 1$ 。
2. 在 Ascend 平台上，目前只支持 $\text{dilation} = 1$ 。

参数：

- **out_channel** (int) - 指定输出通道数： C_{out} 。
- **kernel_size** (Union[int, tuple[int]]) - 指定三维卷积核的高度和宽度。可以为单个 int 或一个包含 3 个 int 组成的元组。单个整数表示该值同时适用于内核的深度、高度和宽度。包含 3 个整数的元组表示第一个值用于深度，另两个值用于高度和宽度。
- **mode** (int, 可选) - 指定不同的卷积模式。此值目前未被使用。默认值：1。
- **stride** (Union[int, tuple[int]], 可选) - 卷积核移动的步长，可以为单个 int 或三个 int 组成的 tuple。一个 int 表示在深度、高度和宽度方向的移动步长均为该值。三个 int 组成的 tuple 分别表示在深度、高度和宽度方向的移动步长。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："valid"。

- "same": 在输入的深度、高度和宽度维度进行填充, 使得当 *stride* 为 1 时, 输入和输出的 *shape* 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在前方/底部/右侧。如果设置了此模式, *pad* 必须为 0。
- "valid": 不对输入进行填充, 返回输出可能的最大深度、高度和宽度, 不能构成一个完整 *stride* 的额外的像素将被丢弃。如果设置了此模式, *pad* 必须为 0。
- "pad": 对输入填充指定的量。在这种模式下, 在输入的深度、高度和宽度方向上填充的量由 *pad* 参数指定。如果设置此模式, *pad* 必须大于或等于 0。
- **pad** (Union(int, tuple[int]), 可选) - 指当 *pad_mode* 为 "pad" 时, 指定在输入 *x* 的深度、高度和宽度方向上填充的数量。可以为单个 int 或包含六个 int 组成的 tuple。如果 *pad* 是单个 int, 那么前、后、上、下、左、右的填充量都等于 *pad*。如果 *pad* 是一个由六个 int 组成的 tuple, 那么前、后、上、下、左、右的填充分别等于 *pad*[0]、*pad*[1]、*pad*[2]、*pad*[3]、*pad*[4] 和 *pad*[5]。int 值应该要大于或等于 0, 默认值: 0。
- **dilation** (Union[int, tuple[int]], 可选) - 卷积核膨胀尺寸。可以为单个 int, 或者由三个 int 组成的 tuple。单个 int 表示在深度、高度和宽度方向的膨胀尺寸均为该值。三个 int 组成的 tuple 分别表示在深度、高度和宽度方向的膨胀尺寸。假设 *dilation* = (*d0*, *d1*, *d2*), 则卷积核在深度方向间隔 *d0* - 1 个元素进行采样, 在高度方向间隔 *d1* - 1 个元素进行采样, 在宽度方向间隔 *d2* - 1 个元素进行采样。深度、高度和宽度上取值范围分别为 [1, D]、[1, H] 和 [1, W]。默认值: 1。
- **group** (int, 可选) - 将过滤器拆分的组数, *in_channels* 和 *out_channels* 必须可被 *group* 整除。默认值: 1。
- **data_format** (str, 可选) - 支持的数据模式。目前仅支持 "NCDHW"。

输入:

- **x** (Tensor) - *shape* 为 (*N*, *C_{in}*, *D_{in}*, *H_{in}*, *W_{in}*) 的 Tensor。目前数据类型仅支持 float16 和 float32。
- **weight** (Tensor) - 若 kernel *shape* 为 (*k_d*, *K_h*, *K_w*), 则 weight *shape* 应为 (*C_{out}*, *C_{in}*/*groups*, *k_d*, *K_h*, *K_w*)。目前数据类型仅支持 float16 和 float32。
- **bias** (Tensor) - *shape* 为 *C_{out}* 的 Tensor。如果 *bias* 为 None, 将不会添加偏置。默认值: None。

输出:

Tensor, 卷积后的值。*shape* 为 (*N*, *C_{out}*, *D_{out}*, *H_{out}*, *W_{out}*)。

pad_mode 为 "same" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in}}{\text{stride}[0]} \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in}}{\text{stride}[1]} \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in}}{\text{stride}[2]} \right\rceil \end{aligned}$$

pad_mode 为 "valid" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in} - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1)}{\text{stride}[0]} + 1 \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in} - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1)}{\text{stride}[1]} + 1 \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} - \text{dilation}[2] \times (\text{kernel\_size}[2] - 1)}{\text{stride}[2]} + 1 \right\rceil \end{aligned}$$

pad_mode 为 "pad" 时:

$$\begin{aligned} D_{out} &= \left\lceil \frac{D_{in} + \text{pad}[0] + \text{pad}[1] - (\text{dilation}[0] - 1) \times \text{kernel\_size}[0] - 1}{\text{stride}[0]} + 1 \right\rceil \\ H_{out} &= \left\lceil \frac{H_{in} + \text{pad}[2] + \text{pad}[3] - (\text{dilation}[1] - 1) \times \text{kernel\_size}[1] - 1}{\text{stride}[1]} + 1 \right\rceil \\ W_{out} &= \left\lceil \frac{W_{in} + \text{pad}[4] + \text{pad}[5] - (\text{dilation}[2] - 1) \times \text{kernel\_size}[2] - 1}{\text{stride}[2]} + 1 \right\rceil \end{aligned}$$

异常:

- **TypeError** - *out_channel* 或 *group* 不是 int。
- **TypeError** - *kernel_size*、*stride*、*pad* 或 *dilation* 既不是 int 也不是 Tuple。
- **ValueError** - *out_channel*、*kernel_size*、*stride* 或 *dilation* 小于 1。
- **ValueError** - *pad* 小于 0。
- **ValueError** - *pad_mode* 取值非"same"、"valid" 或"pad"。
- **ValueError** - *pad* 为长度不等于 6 的 Tuple。
- **ValueError** - *pad_mode* 未设定为"pad" 且 *pad* 不等于 (0, 0, 0, 0, 0)。
- **ValueError** - *data_format* 取值非"NCDHW"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1: specify kernel_size with tuple, all parameters use default values.
>>> x = Tensor(np.ones([16, 3, 10, 32, 32]), mindspore.float16)
>>> weight = Tensor(np.ones([32, 3, 4, 3, 3]), mindspore.float16)
>>> conv3d = ops.Conv3D(out_channel=32, kernel_size=(4, 3, 3))
>>> output = conv3d(x, weight)
>>> print(output.shape)
(16, 32, 7, 30, 30)
>>> # case 2: specify kernel_size with int, all parameters use default values.
>>> x = Tensor(np.ones([10, 20, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([40, 20, 3, 3, 3]), mindspore.float32)
>>> conv3d = ops.Conv3D(out_channel=40, kernel_size=3)
>>> output = conv3d(x, weight)
>>> print(output.shape)
(10, 40, 30, 30, 30)
>>> # case 3: stride=(1, 2, 3), other parameters being default.
>>> x = Tensor(np.ones([10, 20, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([40, 20, 3, 3, 3]), mindspore.float32)
>>> conv3d = ops.Conv3D(out_channel=40, kernel_size=3, stride=(1, 2, 3))
>>> output = conv3d(x, weight)
>>> print(output.shape)
(10, 40, 30, 15, 10)
>>> # case 4: pad_mode="pad", other parameters being default.
>>> x = Tensor(np.ones([10, 20, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([40, 20, 3, 3, 3]), mindspore.float32)
>>> conv3d = ops.Conv3D(out_channel=40, kernel_size=3, pad_mode="pad", pad=2)
>>> output = conv3d(x, weight)
>>> print(output.shape)
(10, 40, 34, 34, 34)
>>> # case 5: dilation=(1, 1, 1), other parameters being default.
>>> x = Tensor(np.ones([10, 20, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([40, 20, 3, 3, 3]), mindspore.float32)
>>> conv3d = ops.Conv3D(out_channel=40, kernel_size=3, dilation=(1, 1, 1))
>>> output = conv3d(x, weight)
>>> print(output.shape)
```

(下页继续)

(续上页)

```

(10, 40, 30, 30, 30)
>>> # case 6: group=1, other parameters being default.
>>> x = Tensor(np.ones([10, 20, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([40, 20, 3, 3, 3]), mindspore.float32)
>>> conv3d = ops.Conv3D(out_channel=40, kernel_size=3, group=1)
>>> output = conv3d(x, weight)
>>> print(output.shape)
(10, 40, 30, 30, 30)
>>> # case 7: All parameters are specified.
>>> x = Tensor(np.ones([10, 20, 32, 32, 32]), mindspore.float32)
>>> weight = Tensor(np.ones([40, 20, 3, 3, 3]), mindspore.float32)
>>> conv3d = ops.Conv3D(out_channel=40, kernel_size=3, stride=(1, 2, 3), pad_mode="pad",
...                    pad=2, dilation=(1), group=1)
>>> output = conv3d(x, weight)
>>> print(output.shape)
(10, 40, 34, 17, 12)

```

mindspore.ops.Conv3DTranspose

class mindspore.ops.Conv3DTranspose (in_channel, out_channel, kernel_size, mode=1, stride=1, pad_mode='valid', pad=0, dilation=1, group=1, output_padding=0, data_format='NCDHW')

计算三维转置卷积，也称为反卷积（实际不是真正的反卷积）。

输入的 **shape** 通常为 (N, C, D, H, W) ，其中 N 为 batch size， C 是空间维度， D 、 H 和 W 分别为特征层的深度、高度和宽度。

若 **pad_mode** 被设定为“pad”，则输出的深度，高度和宽度被定义为：

$$\begin{aligned}
 D_{out} &= (D_{in} - 1) \times \text{stride}[0] - 2 \times \text{pad}[0] + \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) + \text{output\_padding}[0] + 1 \\
 H_{out} &= (H_{in} - 1) \times \text{stride}[1] - 2 \times \text{pad}[1] + \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) + \text{output\_padding}[1] + 1 \\
 W_{out} &= (W_{in} - 1) \times \text{stride}[2] - 2 \times \text{pad}[2] + \text{dilation}[2] \times (\text{kernel\_size}[2] - 1) + \text{output\_padding}[2] + 1
 \end{aligned}$$

说明：

- 在 Ascend 平台上，目前只支持 $\text{group} = 1$ 。
- Atlas A2 训练系列产品暂不支持 **output_padding**。

参数：

- **in_channel** (int) - 输入 *dout* 的通道数。
- **out_channel** (int) - 输入 *weight* 的通道数。
- **kernel_size** (Union[int, tuple[int]]) - 指定三维卷积核的深度、高度和宽度。数据类型为 int 或包含三个 int 值的 Tuple。为 int 时表示卷积核的深度、高度和宽度均为该值。包含三个 int 值的 Tuple 分别表示卷积核的深度、高度和宽度。
- **mode** (int, 可选) - 指定不同的卷积模式。此值目前未被使用。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same"，"valid" 或 "pad"。默认值："valid"。
 - "same": 在输入的深度、高度和宽度维度进行填充，使得当 *stride* 为 1 时，输入和输出的 **shape** 一致。待填充的量由算子内部计算，若为偶数，则均匀地填充在四周，若为奇数，多余的填充量将补充在前方/底部/右侧。如果设置了此模式，*pad* 必须为 0。

- "valid": 不对输入进行填充, 返回输出可能的最大深度、高度和宽度。如果不能构成一个完整 stride, 那么额外的像素将被丢弃。如果设置了此模式, *pad* 必须为 0。
- "pad": 对输入填充指定的量。在这种模式下, 在输入的深度、高度和宽度方向上填充的量由 *pad* 参数指定。如果设置此模式, *pad* 必须大于或等于 0。
- **pad** (Union(int, tuple[int]), 可选) - 在输入各维度两侧填充的数量。如果 *pad* 是一个整数, 则前部、后部、顶部、底部, 左边和右边的填充都等于 *pad*。如果 *pad* 是 6 个整数的 Tuple, 则前部、后部、顶部、底部、左边和右边的填充分别等于填充 *pad*[0]、*pad*[1]、*pad*[2]、*pad*[3]、*pad*[4] 和 *pad*[5]。默认值: 0。
- **stride** (Union(int, tuple[int]), 可选) - 三维卷积核的移动步长。数据类型为整型或三个整型的 Tuple。一个整数表示在深度、高度和宽度方向的移动步长均为该值。三个整数的 Tuple 分别表示在深度、高度和宽度方向的移动步长。默认值: 1。
- **dilation** (Union(int, tuple[int]), 可选) - 卷积核膨胀尺寸, 指定应用卷积核的间隔。默认值: 1。
- **group** (int, 可选) - 将过滤器拆分的组数, *in_channels* 和 *out_channels* 必须可被 *group* 整除。默认值: 1。
- **output_padding** (Union(int, tuple[int]), 可选) - 为输出的各个维度添加额外长度。默认值: 0。
- **data_format** (str, 可选) - 支持的数据模式。目前仅支持 "NCDHW"。默认值: "NCDHW"。

输入:

- **dout** (Tensor) - 卷积操作的输出的梯度 Tensor。shape: $(N, C_{in}, D_{out}, H_{out}, W_{out})$ 。支持数据类型:
 - Ascend: float16。
 - GPU/CPU: float16、float32。
- **weight** (Tensor) - 若 kernel shape 为 (K_d, K_h, K_w) , 则 weight shape 应为 $(C_{in}, C_{out} // group, K_d, K_h, K_w)$, 其中 *group* 为算子参数。// 为整数除法操作。其数据类型与 *dout* 一致。
- **bias** (Tensor) - shape 为 C_{out} 的 Tensor。目前仅支持 None。默认值: None。

输出:

卷积操作的输入的梯度 Tensor, shape: $(N, C_{out} // group, D_{out}, H_{out}, W_{out})$, 其中 *group* 为算子参数。

异常:

- **TypeError** - *in_channel*、*out_channel* 或 *group* 不是 int。
- **TypeError** - *kernel_size*、*stride*、*pad*、*dilation* 或 *output_padding* 既不是 int 也不是 Tuple。
- **ValueError** - *in_channel*、*out_channel*、*kernel_size*、*stride* 或 *dilation* 小于 1。
- **ValueError** - *pad* 小于 0。
- **ValueError** - *pad_mode* 取值非 "same"、"valid" 或 "pad"。
- **ValueError** - *pad* 为长度不等于 6 的 Tuple。
- **ValueError** - *pad_mode* 未设定为 "pad" 且 *pad* 不等于 (0, 0, 0, 0, 0, 0)。
- **ValueError** - *data_format* 取值非 "NCDHW"。
- **TypeError** - *dout* 或 *weight* 的数据类型不是 float16 也不是 float32。
- **ValueError** - *bias* 不为 None。*dout* 或 *weight* 的秩不为 5。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> dout = Tensor(np.ones([32, 16, 10, 32, 32]), mindspore.float16)
>>> weight = Tensor(np.ones([16, 3, 4, 6, 2]), mindspore.float16)
>>> conv3d_transpose = ops.Conv3DTranspose(in_channel=16, out_channel=3, kernel_size=(4, 6, 2))
>>> output = conv3d_transpose(dout, weight)
>>> print(output.shape)
(32, 3, 13, 37, 33)
```

mindspore.ops.CTCGreedyDecoder

class mindspore.ops.CTCGreedyDecoder (*merge_repeated=True*)

对输入中给定的 logits 执行贪婪解码。

更多参考详见 `mindspore.ops.ctc_greedy_decoder()`。

说明：在 Ascend 平台上，目前不支持配置 `merge_repeated=False`。

参数：

- **merge_repeated** (bool, 可选) - 返回的结果中是否合并重复的类。默认值：True。

输入：

- **inputs** (Tensor) - shape: (*max_time*, *batch_size*, *num_classes*)，数据类型必须是 float32 或者 float64。*num_classes* 为 *num_labels* + 1 classes，其中 *num_labels* 表示实际标签的个数，空标签默认使用 *num_classes* - 1。
- **sequence_length** (Tensor) - shape: (*batch_size*,)，数据类型必须是 int32，并且 Tensor 中的数值必须小于等于 *max_time*。

输出：

- **decoded_indices** (Tensor) - shape: (*total_decoded_outputs*, 2)，数据类型为 int64。
- **decoded_values** (Tensor) - shape: (*total_decoded_outputs*,)，数据类型为 int64。
- **decoded_shape** (Tensor) - shape: (*batch_size*, *max_decoded_length*)，数据类型为 int64。
- **log_probability** (Tensor) - shape: (*batch_size*, 1)，包含序列的对数概率，其数据类型与 *inputs* 保持一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> inputs = Tensor(np.array([[[[0.6, 0.4, 0.2], [0.8, 0.6, 0.3]],
...                             [[0.0, 0.6, 0.0], [0.5, 0.4, 0.5]]]], mindspore.float32)
>>> sequence_length = Tensor(np.array([2, 2]), mindspore.int32)
>>> decoded_indices, decoded_values, decoded_shape, log_probability = ops.
    ↪CTCGreedyDecoder()(inputs,
```

(下页继续)

(续上页)

```

...
    sequence_length)
>>> print(decoded_indices)
[[0 0]
 [0 1]
 [1 0]]
>>> print(decoded_values)
[0 1 0]
>>> print(decoded_shape)
[2 2]
>>> print(log_probability)
[[-1.2]
 [-1.3]]

```

mindspore.ops.Dense

class mindspore.ops.Dense

```

prim = ops.Dense()
out = prim(input, weight, bias)

```

等价于

```
ops.dense(input, weight, bias)
```

更多详情请查看: [mindspore.ops.dense\(\)](#)。

mindspore.ops.Dropout

class mindspore.ops.Dropout (*keep_prob=0.5, Seed0=0, Seed1=0*)

Dropout 是一种正则化手段, 通过在训练中以 $1 - \text{keep\_prob}$ 的概率随机将神经元输出设置为 0, 起到减少神经元相关性的作用, 避免过拟合。

更多细节请参考[mindspore.ops.dropout\(\)](#)。

警告: Ascend 后端不支持随机数重现功能, *Seed0* 和 *Seed1* 参数不起作用。

参数:

- **keep_prob** (float, 可选) - 输入神经元保留概率, 数值范围在 0 到 1 之间。例如, `keep_prob=0.9`, 删除 10% 的神经元。默认值: 0.5。
- **Seed0** (int, 可选) - 算子层的随机种子, 用于生成随机数。默认值: 0。
- **Seed1** (int, 可选) - 全局的随机种子, 和算子层的随机种子共同决定最终生成的随机数。默认值: 0。

输入:

- **x** (Tensor) - 输入 Tensor, shape 为 $(*, N)$, 数据类型为 float16、float32 或 float64。

输出:

- **output** (Tensor) - shape 和数据类型与 *x* 相同。
- **mask** (Tensor) - 应用于 *x* 的掩码。

- 在 GPU 和 CPU 上, *mask* 具有与 *x* 相同的 *shape* 和数据类型。
- 在 Ascend 上, 为了获得更好的性能, 它被表示为一个具有 `UInt8` 数据类型的一维 `Tensor`。其 *shape* 为 *(byte_counts,)*, 其中 *byte_counts* 为覆盖 *x* 的 *shape* 所需的字节数。通过下面的公式计算 *byte_counts* 大小:

$$\text{byte\_counts} = \text{ceil}(\text{cumprod}(x.\text{shape})/128) * 16$$

若 *x* 的 *shape* 为 (2, 3, 4, 5, 6), 则 *mask* 的 *shape* 为 (96,)。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> dropout = ops.Dropout(keep_prob=0.5)
>>> x = Tensor(np.ones([1, 2, 3, 4, 5]), mindspore.float32)
>>> output, mask = dropout(x)
>>> print(output.shape, mask.shape, mask.dtype)
(1, 2, 3, 4, 5) (16,) UInt8
```

`mindspore.ops.Dropout2D`

class `mindspore.ops.Dropout2D(keep_prob=0.5)`

在训练期间, 根据概率 $1 - \text{keep\_prob}$, 随机地将一些通道设置为 0, 且服从伯努利分布。(对于 *shape* 为 (N, C, H, W) 的四维 `Tensor`, 通道特征图指的是 *shape* 为 (H, W) 的二维特征图。)

`Dropout2D` 可以提高通道特征图之间的独立性。

说明: 保持概率 *keep_prob* 等于 `mindspore.ops.dropout2d()` 中的 $1 - p$ 。

参数:

- **keep_prob** (float, 可选) - 输入通道保留率, 数值范围在 0 到 1 之间, 例如 *keep_prob* = 0.8, 意味着过滤 20% 的通道。默认值: 0.5。

输入:

- **x** (`Tensor`) - *shape* 为 (N, C, H, W) 的四维张量, 其中 *N* 是批次大小, *C* 是通道数, *H* 是特征高度, *W* 是特征宽度。

输出:

- **output** (`Tensor`) - *shape* 和数据类型与 *x* 相同。
- **mask** (`Tensor`) - *shape* 与 *x* 相同, 数据类型为 `bool`。

异常:

- **TypeError** - *x* 不是 `Tensor`。
- **TypeError** - *keep_prob* 的数据类型不是 `float`。
- **ValueError** - *keep_prob* 值不在 $[0.0, 1.0]$ 之间。
- **ValueError** - *x* 的维度不等于 4。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> dropout = ops.Dropout2D(keep_prob=0.5)
>>> x = Tensor(np.ones([2, 1, 2, 3]), mindspore.float32)
>>> output, mask = dropout(x)
>>> print(output.shape)
(2, 1, 2, 3)
```

mindspore.ops.Dropout3D

class mindspore.ops.Dropout3D(keep_prob=0.5)

在训练期间，以服从伯努利分布的概率 $1 - \text{keep\_prob}$ 随机将输入 Tensor 的某些通道归零。(对于形状为 *NCDHW* 的 5D Tensor，其通道特征图指的是后三维 *DHW* 形状的三维特征图)。

说明：保持概率 *keep_prob* 等于 `mindspore.ops.dropout3d()` 中的 $1 - p$ 。

Dropout3D 可以提高 feature map 之间的独立性。

参数：

- **keep_prob** (float, 可选) - 输入通道保留率，数值范围在 0 到 1 之间，例如 *keep_prob* = 0.8，意味着过滤 20% 的通道。默认值：0.5。

输入：

- **x** (Tensor) - shape 为 (*N, C, D, H, W*) 的五维 Tensor。N 代表批次大小，C 代表管道，D 代表特征深度，H 代表特征高度，W 代表特征宽度。

输出：

- **output** (Tensor) - shape 和数据类型与 *x* 相同。
- **mask** (Tensor) - shape 与 *x* 相同，数据类型为 bool。

异常：

- **TypeError** - *keep_prob* 的数据类型不是 float。
- **ValueError** - *keep_prob* 超出 [0.0, 1.0] 范围，或者输入的维度不是 5。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> dropout = ops.Dropout3D(keep_prob=0.5)
>>> x = Tensor(np.ones([2, 1, 2, 1, 2]), mindspore.float32)
>>> output, mask = dropout(x)
```

(下页继续)

(续上页)

```
>>> print(output.shape)
(2, 1, 2, 1, 2)
```

mindspore.ops.DynamicGRUV2

```
class mindspore.ops.DynamicGRUV2 (direction='UNIDIRECTIONAL', cell_depth=1, keep_prob=1.0, cell_clip=- 1.0,
                                   num_proj=0, time_major=True, activation='tanh', gate_order='rzh',
                                   reset_after=True, is_training=True)
```

为输入序列应用一个单层 GRU(gated recurrent unit)。

$$\begin{aligned} r_{t+1} &= \sigma(W_{ir}x_{t+1} + b_{ir} + W_{hr}h_{(t)} + b_{hr}) \\ z_{t+1} &= \sigma(W_{iz}x_{t+1} + b_{iz} + W_{hz}h_{(t)} + b_{hz}) \\ n_{t+1} &= \tanh(W_{in}x_{t+1} + b_{in} + r_{t+1} * (W_{hn}h_{(t)} + b_{hn})) \\ h_{t+1} &= (1 - z_{t+1}) * n_{t+1} + z_{t+1} * h_{(t)} \end{aligned}$$

其中 h_{t+1} 是在时刻 $t+1$ 的隐藏状态, x_{t+1} 是时刻 $t+1$ 的输入, h_t 为时刻 t 的隐藏状态或时刻 0 的初始隐藏状态。 r_{t+1} 、 z_{t+1} 、 n_{t+1} 分别为重置门、更新门和当前候选集。 W , b 为可学习权重和偏置。 σ 是 sigmoid 激活函数, $*$ 为 Hadamard 乘积。

参数:

- **direction** (str, 可选) - 指定 GRU 方向, str 类型。默认值: "UNIDIRECTIONAL"。目前仅支持 "UNIDIRECTIONAL"。
- **cell_depth** (int, 可选) - GRU 单元深度。默认值: 1。
- **keep_prob** (float, 可选) - Dropout 保留概率。默认值: 1.0。
- **cell_clip** (float, 可选) - 输出裁剪率。默认值: -1.0。
- **num_proj** (int, 可选) - 投影维度。默认值: 0。
- **time_major** (bool, 可选) - 如为 True, 则指定输入的第一维度为序列长度 *num_step*, 如为 False 则第一维度为 *batch_size*。默认值: True。
- **activation** (str, 可选) - 字符串, 指定 activation 类型。默认值: "tanh"。目前仅支持取值 "tanh"。
- **gate_order** (str, 可选) - 字符串, 指定 weight 和 bias 中门的排列顺序, 可选值为 "rzh" 或 "zrh"。默认值: "rzh"。"rzh" 代表顺序为: 重置门、更新门、隐藏门。"zrh" 代表顺序为: 更新门、重置门、隐藏门。
- **reset_after** (bool, 可选) - 是否在矩阵乘法后使用重置门。默认值: True。
- **is_training** (bool, 可选) - 是否为训练模式。默认值: True。

输入:

- **x** (Tensor) - 输入词序列。shape: (num_step, batch_size, input_size)。数据类型支持 float16。
- **weight_input** (Tensor) - 权重 $W_{\{ir, iz, in\}}$ 。shape: (input_size, $3 \times \text{hidden\_size}$)。数据类型支持 float16。
- **weight_hidden** (Tensor) - 权重 $W_{\{hr, hz, hn\}}$ 。shape: (hidden_size, $3 \times \text{hidden\_size}$)。数据类型支持 float16。
- **bias_input** (Tensor) - 偏差 $b_{\{ir, iz, in\}}$ 。shape: ($3 \times \text{hidden\_size}$), 或 None。与输入 *init_h* 的数据类型相同。
- **bias_hidden** (Tensor) - 偏差 $b_{\{hr, hz, hn\}}$ 。shape: ($3 \times \text{hidden\_size}$), 或 None。与输入 *init_h* 的数据类型相同。
- **seq_length** (Tensor) - 每个 batch 中序列的长度。shape: (batch_size)。目前仅支持 None。
- **init_h** (Tensor) - 初始隐藏状态。shape: (batch_size, hidden_size)。数据类型支持 float16 和 float32。

输出:

- **y** (Tensor) - Tensor, 与 *bias_type* 数据类型相同。shape 如下:
 - (*num_step*, *batch_size*, *min(hidden_size, num_proj)*), 如果 *num_proj* 大于 0,
 - (*num_step*, *batch_size*, *hidden_size*), 如果 *num_proj* 等于 0。
- **output_h** (Tensor) - Tensor, shape: (*num_step*, *batch_size*, *hidden_size*)。与 *bias_type* 数据类型相同。
- **update** (Tensor) - Tensor, shape: (*num_step*, *batch_size*, *hidden_size*)。与 *bias_type* 数据类型相同。
- **reset** (Tensor) - Tensor, shape: (*num_step*, *batch_size*, *hidden_size*)。与 *bias_type* 数据类型相同。
- **new** (Tensor) - Tensor, shape: (*num_step*, *batch_size*, *hidden_size*)。与 *bias_type* 数据类型相同。
- **hidden_new** (Tensor) - Tensor, shape: (*num_step*, *batch_size*, *hidden_size*)。与 *bias_type* 数据类型相同。关于 *bias_type*:
 - 如果 *bias_input* 和 *bias_hidden* 均为 *None*, 则 *bias_type* 为 *init_h* 的数据类型。
 - 如果 *bias_input* 不为 *None*, 则 *bias_type* 为 *bias_input* 的数据类型。
 - 如果 *bias_input* 为 *None* 而 *bias_hidden* 不为 *None*, 则 *bias_type* 为 *bias_hidden* 的数据类型。

异常:

- **TypeError** - *direction*、*activation* 或 *gate_order* 不是 str。
- **TypeError** - *cell_depth* 或 *num_proj* 不是 int 类型。
- **TypeError** - *keep_prob* 或 *cell_clip* 不是 float 类型。
- **TypeError** - *time_major*、*reset_after* 或 *is_training* 不是 bool 类型。
- **TypeError** - *x*、*weight_input*、*weight_hidden*、*bias_input*、*bias_hidden*、*seq_length* 或 *ini_h* 不是 Tensor。
- **TypeError** - *x*、*weight_input* 或 *weight_hidden* 的数据类型非 float16。
- **TypeError** - *init_h* 数据类型非 float16 或 float32。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.rand(2, 8, 64).astype(np.float16))
>>> weight_i = Tensor(np.random.rand(64, 48).astype(np.float16))
>>> weight_h = Tensor(np.random.rand(16, 48).astype(np.float16))
>>> bias_i = Tensor(np.random.rand(48).astype(np.float16))
>>> bias_h = Tensor(np.random.rand(48).astype(np.float16))
>>> init_h = Tensor(np.random.rand(8, 16).astype(np.float16))
>>> dynamic_gru_v2 = ops.DynamicGRUV2()
>>> output = dynamic_gru_v2(x, weight_i, weight_h, bias_i, bias_h, None, init_h)
>>> print(output[0].shape)
(2, 8, 16)
```

mindspore.ops.DynamicRNN

```
class mindspore.ops.DynamicRNN (cell_type='LSTM', direction='UNIDIRECTIONAL', cell_depth=1, use_peephole=False,
                                keep_prob=1.0, cell_clip=-1.0, num_proj=0, time_major=True, activation='tanh',
                                forget_bias=0.0, is_training=True)
```

将循环神经网络应用到输入上。当前仅支持 LSTM。

$$\begin{aligned}
 i_{t+1} &= \sigma(W_{ix}x_{t+1} + b_{ix} + W_{ih}h_{(t)} + b_{ih}) \\
 f_{t+1} &= \sigma(W_{fx}x_{t+1} + b_{fx} + W_{fh}h_{(t)} + b_{fh}) \\
 \tilde{c}_{t+1} &= \tanh(W_{cx}x_{t+1} + b_{cx} + W_{ch}h_{(t)} + b_{ch}) \\
 o_{t+1} &= \sigma(W_{ox}x_{t+1} + b_{ox} + W_{oh}h_{(t)} + b_{oh}) \\
 c_{t+1} &= f_{t+1} * c_{(t)} + i_t * \tilde{c}_{t+1} \\
 h_{t+1} &= o_{t+1} * \tanh(c_{t+1})
 \end{aligned}$$

其中, h_{t+1} 是在 $t+1$ 时刻的隐藏状态。 x_{t+1} 是在 $t+1$ 时刻的输入。 h_t 是在 t 时刻的隐藏状态或在 0 时刻的初始隐藏状态。 σ 是 sigmoid 函数, $*$ 是 Hadamard 积。 W, b 是公式中输出和输入之间的可学习权重。

例如, W_{ix}, b_{ix} 是把 x 转换为 i 的权重和偏置。

参数:

- **cell_type** (str, 可选) - 指定 Cell 类型。当前仅支持 'LSTM'。默认值: 'LSTM'。
- **direction** (str, 可选) - 指定单向或双向。默认值: 'UNIDIRECTIONAL'。当前仅支持 'UNIDIRECTIONAL'。
- **cell_depth** (int, 可选) - 指定 cell 的层数。默认值: 1。
- **use_peephole** (bool, 可选) - 是否使用“peephole connections”。默认值: False。
- **keep_prob** (float, 可选) - 指定保留率, 即每个元素被保留的概率。1.0 表示所有元素全部保留。默认值: 1.0。
- **cell_clip** (float, 可选) - 将 Cell 裁剪到指定的值, 负值表示禁用。默认值: -1.0。
- **num_proj** (int, 可选) - 投影矩阵的输出维数。默认值: 0。
- **time_major** (bool, 可选) - 指定输入 x 的数据排列格式。如果为 True, 格式为 $(num\_step, batch\_size, input\_size)$, 如果为 False, 格式为: $(batch\_size, num\_step, input\_size)$ 。默认值: True。当前仅支持 True。
- **activation** (str, 可选) - 指定激活函数。默认值: 'tanh'。当前仅支持 'tanh'。
- **forget_bias** (float, 可选) - 指定遗忘门的偏置。默认值: 0.0。
- **is_training** (bool, 可选) - 指定是否开启训练。默认值: True。

输入:

- **x** (Tensor) - 输入的词汇。shape 为 $(num\_step, batch\_size, input\_size)$ 的 Tensor。数据类型必须为 float16。
- **w** (Tensor) - 输入的权重。shape 为 $(input\_size + hidden\_size, 4 * hidden\_size)$ 的 Tensor。数据类型必须为 float16。
- **b** (Tensor) - 输入的偏置。shape 为 $(4 * hidden\_size)$ 的 Tensor。数据类型必须为 float16。
- **seq_length** (Tensor) - 每个批次中句子的真实长度。shape 为 $(batch\_size,)$ 的 Tensor。当前仅支持 None。
- **init_h** (Tensor) - 在初始时刻的隐藏状态。shape 为 $(1, batch\_size, hidden\_size)$ 的 Tensor。数据类型必须为 float16。
- **init_c** (Tensor) - 在初始时刻的 Cell 状态。shape 为 $(1, batch\_size, hidden\_size)$ 的 Tensor。数据类型必须为 float16。

输出:

- **y** (Tensor) - 所有时刻输出层的输出向量, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型与输入 b 相同。
- **output_h** (Tensor) - 所有时刻输出层的输出向量, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型为 float16。
- **output_c** (Tensor) - 所有时刻的 Cell 状态的输出向量, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型与输入 b 相同。
- **i** (Tensor) - 更新输入门的权重, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型与输入 b 相同。
- **j** (Tensor) - 更新单元状态的权重, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型与输入 b 相同。
- **f** (Tensor) - 更新遗忘门的权重, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型输入 b 相同。
- **o** (Tensor) - 更新输出门的权重, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型与输入 b 相同。
- **tanhct** (Tensor) - 更新 tanh 的权重, shape 为 $(num\_step, batch\_size, hidden\_size)$ 的 Tensor。数据类型与输入 b 相同。

异常:

- **TypeError** - *cell_type*、*direction* 或 *activation* 不是 str。
- **TypeError** - *cell_Deep* 或 *num_proj* 不是 int。
- **TypeError** - *keep_prob*、*cell_clip* 或 *forget_bias* 不是 float。
- **TypeError** - *use_peehpol*、*time_major* 或 *is_training* 不是 bool。
- **TypeError** - *x*、*w*、*b*、*seq_length*、*init_h* 或 *init_c* 不是 Tensor。
- **TypeError** - *x*、*w*、*init_h* 或 *nit_c* 的数据类型不是 float16。
- **TypeError** - *b* 的数据类型既不是 float16 也不是 float32。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.rand(2, 16, 64).astype(np.float16))
>>> w = Tensor(np.random.rand(96, 128).astype(np.float16))
>>> b = Tensor(np.random.rand(128).astype(np.float16))
>>> init_h = Tensor(np.random.rand(1, 16, 32).astype(np.float16))
>>> init_c = Tensor(np.random.rand(1, 16, 32).astype(np.float16))
>>> dynamic_rnn = ops.DynamicRNN()
>>> output = dynamic_rnn(x, w, b, None, init_h, init_c)
>>> print(output[0].shape)
(2, 16, 32)
```

mindspore.ops.EmbeddingLookup

class mindspore.ops.EmbeddingLookup

根据指定的索引，返回输入 Tensor 的切片。

此算子在 $axis = 0$ 上的运行与 GatherV2 的功能相似，只是多一个 *offset* 输入。

输入：

- **input_params** (Tensor) - Tensor 切片，shape 为 $(x_1, x_2, \dots, x_R)$ ，当前只支持二维。
- **input_indices** (Tensor) - 指定输入 Tensor 元素的索引，shape 为 $(y_1, y_2, \dots, y_S)$ 。当取值超出 *input_params* 在该维度的最大长度时，超出部分将返回 0 值。不支持负值，否则结果将未定义。其数据类型为 int32 或 int64。
- **offset** (int) - 指定 *input_params* 切片的偏移值。实际索引等于 *input_indices* 减去 *offset*。

输出：

Tensor，shape 为 $(z_1, z_2, \dots, z_N)$ 的 Tensor。数据类型与 *input_params* 相同。

异常：

- **TypeError** - *input_indices* 的数据类型不是 int。
- **ValueError** - *input_params* 的 shape 长度大于 2。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_params = Tensor(np.array([[8, 9], [10, 11], [12, 13], [14, 15]]), mindspore.float32)
>>> input_indices = Tensor(np.array([[5, 2], [8, 5]]), mindspore.int32)
>>> offset = 4
>>> output = ops.EmbeddingLookup()(input_params, input_indices, offset)
>>> print(output)
[[[10. 11.]
  [ 0.  0.]]
 [[ 0.  0.]
  [10. 11.]]]
```

mindspore.ops.Flatten

class mindspore.ops.Flatten

扁平化 (Flatten) 输入 Tensor，不改变 0 轴的 size。

更多参考详见 *mindspore.ops.flatten()*。

输入：

- **input_x** (Tensor) - 待扁平化的 Tensor，其 shape 为 $(N, \dots)$ ， N 表示 batch size。

输出：

Tensor，输出 shape 为 (N, X) 的 Tensor，其中 X 是余下维度的乘积。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.ones(shape=[1, 2, 3, 4]), mindspore.float32)
>>> flatten = ops.Flatten()
>>> output = flatten(input_x)
>>> print(output.shape)
(1, 24)
```

mindspore.ops.FractionalMaxPool3DWithFixedKsize

class mindspore.ops.FractionalMaxPool3DWithFixedKsize (*ksize, output_shape, data_format='NCDHW'*)

此运算对由多个输入平面组成的输入信号进行 3D 分数最大池化。最大池化操作在 (k_D, k_H, k_W) 区域中进行。其步长是随机的, 大小由 *output_shape* 决定。

输出特征的数量等于输入平面的数量。

详细内容请参考论文 [Fractional MaxPooling by Ben Graham](#)。

输入和输出的数据格式可以是“NCDHW”和“NDHWC”, N 是批量大小, C 是通道数, D 是特征深度, H 是特征高度, W 是特征宽度。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **ksize** (Union[float, tuple]) - 池化窗口大小。可以是一个三元组, 表示 shape 为 (k_D, k_H, k_W) , 或者是一个单独的 K 表示 shape 为 (K, K, K) 。
- **output_shape** (Union[int, tuple]) - 目标输出 shape, 可以是一个三元组, 表示 shape 为 $(D_{out}, H_{out}, W_{out})$, 或者是一个单独的 S 表示 shape 为 (S, S, S) 。
- **data_format** (str, 可选) - 可选的数据格式值, 当前支持 "NCDHW" 和 "NDHWC", 默认为 "NCDHW"。

输入:

- **x** (Tensor) - 输入 Tensor, 是一个 4-D 或者 5-D 的 Tensor。数据类型为: float16、float32、double、int32、int64。支持的 shape $(N, C, D_{in}, H_{in}, W_{in})$ 或者 $(N, D_{in}, H_{in}, W_{in}, C)$ 。
- **random_samples** (Tensor) - 随机步长, 是一个 3-D Tensor, 数据类型为: float16、float32、double, 值在 $(0, 1)$ 之间。支持的 shape 为 $(N, C, 3)$ 。

输出:

- **y** (Tensor) - 一个 Tensor, 与 x 具有相同的 dtype, shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 或者 $(N, D_{out}, H_{out}, W_{out}, C)$ 。
- **argmax** (Tensor) - 一个 Tensor, 输出的索引值。与 y 具有相同的 shape, dtype 为 int32 或者 int64。

异常:

- **TypeError** - 如果 *input_x* 不是一个 4-D 或者 5-D 的 Tensor。
- **TypeError** - 如果 *random_samples* 不是一个 3-D 的 Tensor。
- **TypeError** - 如果 x 的数据类型不是: float16、float32、double、int32、int64。
- **TypeError** - 如果 *random_samples* 的数据类型不是: float16、float32、double。
- **TypeError** - 如果 *argmax* 不是 int32、int64。

- **ValueError** - 如果 `output_shape` 不是一个元组, 并且 `output_shape` 长度不是 3。
- **ValueError** - 如果 `ksize` 是一个元组, 并且 `ksize` 长度不是 3。
- **ValueError** - 如果 `output_shape` 或者 `ksize` 中的数值不是正数。
- **ValueError** - 如果 `data_format` 不是 'NCDHW', 也不是 'NDHWC'。
- **ValueError** - 如果 `input_x` 和 `random_samples` 的第一维大小不相等。
- **ValueError** - 如果 `input_x` 和 `random_samples` 的第二维大小不相等。
- **ValueError** - 如果 `random_samples` 的第三维大小不是 3。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]))
...     .reshape([1, 1, 2, 2, 4]), mstype.float32)
>>> random_samples = Tensor(np.array([0.7, 0.7, 0.7]).reshape([1, 1, 3]), mstype.float32)
>>> ksize = (1, 1, 1)
>>> output_shape = (1, 1, 2)
>>> net = ops.FractionalMaxPool3DWithFixedKsize(ksize = ksize, output_shape = output_shape)
>>> output, argmax = net(x, random_samples)
>>> print(output)
[[[[[13. 16.]]]]]
>>> print(argmax)
[[[[[12 15]]]]]
```

`mindspore.ops.GridSampler2D`

class `mindspore.ops.GridSampler2D` (`interpolation_mode='bilinear'`, `padding_mode='zeros'`, `align_corners=False`)

此操作使用基于流场网格的插值对 2D `input_x` 进行采样, 该插值通常由 `mindspore.ops.affine_grid()` 生成。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多参考详见 `mindspore.ops.grid_sample()`。

参数:

- **interpolation_mode** (str, 可选) - 指定插值方法的可选字符串。可选值为: 'bilinear'、'nearest', 默认值: 'bilinear'。
 - 'nearest': 最近邻插值。每个输出像素的值为最近的输入像素的值。这种方法简单快速, 但可能导致块状或像素化的输出。
 - 'bilinear': 双线性插值。每个输出像素是最接近的四个输入像素的加权平均值, 使用双线性插值计算。与最近邻插值相比, 此方法产生更平滑的结果。
- **padding_mode** (str, 可选) - 指定填充模式的可选字符串。可选值为: 'zeros'、'border' 或者 'reflection', 默认值: 'zeros'。当采样 `grid` 超出输入 `Tensor` 的边界时, 各种填充模式效果如下:
 - 'zeros': 使用零填充输入 `Tensor`。

- 'border': 使用 Tensor 边缘上像素的值填充输入 Tensor。
- 'reflection': 通过反射 Tensor 边界处的像素值, 并将反射值沿着 Tensor 的边界向外扩展来填充输入 Tensor。
- **align_corners** (bool, 可选) - 一个可选 bool。如果为 True, 输入和输出 Tensor 的角像素是对齐的, 如果为 False, 则不使用角像素对齐。默认值: False。

输入:

- **input_x** (Tensor) - 一个 4D 的 Tensor, shape 为 (N, C, H_{in}, W_{in}) 。支持数据类型:
 - Ascend: float16、float32。
 - GPU/CPU: float16、float32、float64。
- **grid** (Tensor) - 一个 4D 的 Tensor, dtype 和 *input_x* 相同, shape 为 $(N, H_{out}, W_{out}, 2)$, 用于指定由输入空间维度归一化的采样像素位置。

输出:

一个 4-D 的 Tensor, dtype 和 *input_x* 相同, shape 为 (N, C, H_{out}, W_{out}) 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> gridsampler = ops.GridSampler2D(interpolation_mode='bilinear', padding_mode='zeros',
    ↪align_corners=True)
>>> input_x = Tensor(np.arange(16).reshape((2, 2, 2, 2)).astype(np.float32))
>>> grid = Tensor(np.arange(-9, 9, 0.5).reshape((2, 3, 3, 2)).astype(np.float32))
>>> output = gridsampler(input_x, grid)
>>> print(output)
[[[ [ 0.    0.    0.   ]
    [ 0.    0.    0.   ]
    [ 0.    0.    0.5  ]
    [ 0.    0.    0.   ]
    [ 0.    0.    0.   ]
    [ 0.    1.5   4.5  ]]]
 [[ [10.    8.25   1.375]
    [ 0.    0.    0.   ]
    [ 0.    0.    0.   ]
    [14.    11.25   1.875]
    [ 0.    0.    0.   ]
    [ 0.    0.    0.   ]]]]
```

mindspore.ops.GridSampler3D

class mindspore.ops.**GridSampler3D** (*interpolation_mode='bilinear', padding_mode='zeros', align_corners=False*)
 给定一个输入和一个网格, 使用网格中的输入值和像素位置计算输出。只支持体积 (5-D) 的输入。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多参考详见 `mindspore.ops.grid_sample()`。

参数:

- **interpolation_mode** (str, 可选) - 指定插值方法。可选方法为 'bilinear'、'nearest' 或 'bicubic'。默认值: 'bilinear'。
 - 'nearest': 最近邻插值。每个输出像素的值为最近的输入像素的值。这种方法简单快速, 但可能导致块状或像素化的输出。
 - 'bilinear': 双线性插值。每个输出像素是最接近的四个输入像素的加权平均值。与最近邻插值相比, 此方法产生更平滑的结果。
- **padding_mode** (str, 可选) - 指定填充模式的可选字符串。可选值为: 'zeros'、'border' 或者 'reflection', 默认值: 'zeros'。当采样 **grid** 超出输入 **Tensor** 的边界时, 各种填充模式效果如下:
 - 'zeros': 使用零填充输入 **Tensor**。
 - 'border': 使用 **Tensor** 边缘上像素的值填充输入 **Tensor**。
 - 'reflection': 通过反射 **Tensor** 边界处的像素值, 并将反射值沿着 **Tensor** 的边界向外扩展来填充输入 **Tensor**。
- **align_corners** (bool, 可选) - 指定对齐方式。如果设置成 **True**, -1 和 1 将被视为引用输入角像素的中心点。如果设置为 **False**, -1 和 1 将被视为引用输入角像素的角点, 使采样不受分辨率影响。默认值为 **False**。

输入:

- **input_x** (**Tensor**) - 5-D 输入 **Tensor**, shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 。数据类型为 float16、float32 或 float64。
- **grid** (**Tensor**) - 5-D **Tensor**, shape 为 $(N, D_{out}, H_{out}, W_{out}, 3)$ 。数据类型与 **input_x** 保持一致。

输出:

Tensor, 数据类型与 **input_x** 相同, shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> gridsampler = ops.GridSampler3D(interpolation_mode='bilinear', padding_mode='zeros',
    ↪align_corners=True)
>>> input_x = Tensor(np.arange(32).reshape((2, 2, 2, 2, 2)).astype(np.float32))
>>> grid = Tensor(np.arange(-0.2, 1, 0.1).reshape((2, 2, 1, 1, 3)).astype(np.float32))
>>> output = gridsampler(input_x, grid)
>>> print(output)
[[[[[ 3.3      ]]
    [[ 4.35     ]]
    [[11.300001]]
    [[12.349999]]]]
 [[[[21.4      ]]
    [[22.449999]]
    [[29.4      ]]
    [[30.449999]]]]]
```

mindspore.ops.LayerNorm

class mindspore.ops.LayerNorm(begin_norm_axis=1, begin_params_axis=1, epsilon=1e-7)

在输入 Tensor 上应用层归一化 (Layer Normalization)。

此算子将在给定的轴上对输入进行层归一化。Layer Normalization 描述了 LayerNorm。

$$y = \frac{x - \text{mean}}{\sqrt{\text{variance} + \epsilon}} * \gamma + \beta$$

其中 γ 是 Scalar, β 是偏置项, ϵ 是精度值。

参数:

- **begin_norm_axis** (int) - 指定 *input_x* 需进行层归一化的起始维度, 其值必须在 $[-1, \text{rank}(\text{input}_x)]$ 范围内。默认值: 1。
- **begin_params_axis** (int) - 指定输入参数 (*gamma*, *beta*) 需进行层归一化的开始轴, 其值必须在 $[-1, \text{rank}(\text{input}_x)]$ 范围内。默认值: 1。注: 在 Ascend 平台, *begin_params_axis* 和 *begin_norm_axis* 的值需相等。
- **epsilon** (float) - 添加到分母中的值 (ϵ), 以确保数据稳定性。默认值: $1e-7$ 。

输入:

- **input_x** (Tensor) - LayerNorm 的输入, shape 为 $(N, \dots)$ 的 Tensor。支持的数据类型: float16、float32、float64。
- **gamma** (Tensor) - 可学习参数 γ , shape 为 *input_x_shape*[*begin_params_axis*:] 的 Tensor。支持的数据类型: float16、float32、float64。
- **beta** (Tensor) - 可学习参数 β , shape 为 *input_x_shape*[*begin_params_axis*:] 的 Tensor。支持的数据类型: float16、float32、float64。

输出:

tuple[Tensor], 3 个 Tensor 组成的 tuple, 层归一化输入和更新后的参数。

- **output_x** (Tensor) - 层归一化输入, 数据类型和 shape 与 *input_x* 相同。
- **mean** (Tensor) - 输入的均值, 其 shape 的前 *begin_norm_axis* 维与 *input_x* 相同, 其余维度为 1。假设输入 *input_x* 的 shape 为 $(x_1, x_2, \dots, x_R)$, 输出 *mean* 的 shape 为 $(x_1, \dots, x_{\text{begin\_norm\_axis}}, 1, \dots, 1)$ (当 *begin_norm_axis*=0 时, *mean* shape 为 $(1, \dots, 1)$)。
- **rstd** (Tensor) - 输入的标准差的倒数, shape 同 *mean* 一致。

异常:

- **TypeError** - *begin_norm_axis* 或 *begin_params_axis* 不是 int。
- **TypeError** - *epsilon* 不是 float。
- **TypeError** - *input_x*、*gamma* 或 *beta* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[1, 2, 3], [1, 2, 3]]), mindspore.float32)
>>> gamma = Tensor(np.ones([3]), mindspore.float32)
>>> beta = Tensor(np.ones([3]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> layer_norm = ops.LayerNorm()
>>> output, _, _ = layer_norm(input_x, gamma, beta)
>>> print(output)
[[-0.2247448  1.          2.2247448]
 [-0.2247448  1.          2.2247448]]
```

mindspore.ops.LRN

class mindspore.ops.LRN (*depth_radius=5, bias=1.0, alpha=1.0, beta=0.5, norm_region='ACROSS_CHANNELS'*)
局部响应归一化操作 LRN(Local Response Normalization)。

警告：LRN 在 Ascend 平台已废弃，存在潜在精度问题。建议使用其他归一化方法，如 `mindspore.ops.BatchNorm` 代替 LRN。

$$b_c = a_c \left(k + \frac{\alpha}{n} \sum_{c'=\max(0, c-n/2)}^{\min(N-1, c+n/2)} a_{c'}^2 \right)^{-\beta}$$

其中 a_c 表示特征图中 c 对应的具体像素值； $n/2$ 为参数 `depth_radius`； k 为参数 `bias`； α 为参数 `alpha`； β 为参数 `beta`。

参数：

- **depth_radius** (int) - 一维归一化窗口的半宽。默认值：5。
- **bias** (float) - 偏移量（通常为正以避免除零问题）。默认值：1.0。
- **alpha** (float) - 比例系数，通常为正。默认值：1.0。
- **beta** (float) - 指数。默认值：0.5。
- **norm_region** (str) - 指定归一化区域。可选值："ACROSS_CHANNELS"。默认值："ACROSS_CHANNELS"。

输入：

- **x** (Tensor) - 数据类型为 float16 或 float32 的四维 Tensor。

输出：

Tensor，与 x 的 shape 和数据类型相同。

异常：

- **TypeError** - `depth_radius` 不是 int 类型。
- **TypeError** - `bias`、`alpha` 或 `beta` 不是 float 类型。
- **TypeError** - `norm_region` 不是 str。
- **TypeError** - `x` 不是 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[[0.1], [0.2]],
...                      [[0.3], [0.4]]]]), mindspore.float32)
>>> lrn = ops.LRN()
>>> output = lrn(x)
>>> print(output)
[[[0.09534626]
  [0.1825742 ]]
 [0.2860388 ]
 [0.3651484 ]]]
```

mindspore.ops.LSTM

class mindspore.ops.LSTM(*input_size, hidden_size, num_layers, has_bias, bidirectional, dropout, proj_size=0*)
对输入执行长短期记忆 (LSTM) 网络。

有关详细信息, 请参见 [mindspore.nn.LSTM](#)。

参数:

- **input_size** (int) - 输入的大小。
- **hidden_size** (int) - 隐藏状态大小。
- **num_layers** (int) - LSTM 的网络层数, 当前 CPU 只支持 1。
- **has_bias** (bool) - Cell 是否有偏置 b_{ih} 和 b_{hh} , 当前 CPU 只支持 *False*。
- **bidirectional** (bool) - 是否为双向 LSTM, 当前 CPU 只支持 *False*。
- **dropout** (float) - 指的是除第一层外每层输入时的 dropout 概率。dropout 的范围为 [0.0, 1.0]。
- **proj_size** (int) - 如果大于 0, 则会使用对应大小的投影, 当前只支持 CPU 后端。默认值: 0。

输入:

- **input** (Tensor) - shape 为 (*seq_len, batch_size, input_size*) 或 (*batch_size, seq_len, input_size*) 的 Tensor。
- **h** (tuple) - shape 为 (*num_directions * num_layers, batch_size, real_hidden_size*) 的 Tensor。
- **c** (tuple) - shape 为 (*num_directions * num_layers, batch_size, hidden_size*) 的 Tensor。
- **w** (Tensor) - 权重。

如果 $proj\_size > 0$, $real\_hidden\_size = proj\_size$, 否则 $real\_hidden\_size = hidden\_size$ 。

输出:

tuple, tuple 包含 (*output, h_n, c_n, reserve, state*)。

- **output** (Tensor) - shape 为 (*seq_len, batch_size, num_directions * real_hidden_size*) 的 Tensor。
- **h_n** (Tensor) - shape 为 (*num_directions * num_layers, batch_size, real_hidden_size*) 的 Tensor。
- **c_n** (Tensor) - shape 为 (*num_directions * num_layers, batch_size, hidden_size*) 的 Tensor。
- **reserve** (Tensor) - shape 为 (*r, 1*)。
- **state** (Tensor) - 使用随机数生成状态及其 shape 为 (*s, 1*)。

异常:

- **TypeError** - 如果 *input_size*, *hidden_size* 或 *num_layers* 不是 int。

- **TypeError** - 如果 *has_bias* 或 *bidirectional* 不是 bool。
- **TypeError** - 如果 *dropout* 不是 float。
- **ValueError** - 如果 *dropout* 不在范围 [0.0, 1.0] 内。
- **ValueError** - 如果 *proj_size* 不在范围 [0, *hidden_size*] 内。

支持平台：

GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_size = 10
>>> hidden_size = 2
>>> num_layers = 1
>>> seq_len = 5
>>> batch_size = 2
>>>
>>> net = ops.LSTM(input_size, hidden_size, num_layers, True, False, 0.0)
>>> input_tensor = Tensor(np.ones([seq_len, batch_size, input_size]).astype(np.float32))
>>> h0 = Tensor(np.ones([num_layers, batch_size, hidden_size]).astype(np.float32))
>>> c0 = Tensor(np.ones([num_layers, batch_size, hidden_size]).astype(np.float32))
>>> w = Tensor(np.ones([112, 1, 1]).astype(np.float32))
>>> output, hn, cn, _, _ = net(input_tensor, h0, c0, w)
>>> print(output)
[[[0.9640267  0.9640267 ]
  [0.9640267  0.9640267 ]]
 [0.9950539  0.9950539 ]
 [0.9950539  0.9950539 ]]
 [0.99932843 0.99932843]
 [0.99932843 0.99932843]]
 [0.9999084  0.9999084 ]
 [0.9999084  0.9999084 ]]
 [0.9999869  0.9999869 ]
 [0.9999869  0.9999869 ]]]
```

mindspore.ops.MaxPool

class mindspore.ops.**MaxPool** (*kernel_size=1, strides=1, pad_mode='valid', data_format='NCHW'*)

对输入的多维数据进行二维的最大池化运算。

在一个输入 Tensor 上应用 2D max pooling，可被视为 2D 输入平面。

通常，输入的 shape 为 $(N_{in}, C_{in}, H_{in}, W_{in})$ ，MaxPool 在 (H_{in}, W_{in}) 维度输出区域最大值。给定 *kernel_size* 为 $ks = (h_{ker}, w_{ker})$ 和 *stride* $s = (s_0, s_1)$ ，运算如下：

$$\text{output}(N_i, C_j, h, w) = \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} \text{input}(N_i, C_j, s_0 \times h + m, s_1 \times w + n)$$

参数：

- **kernel_size** (Union[int, tuple[int]]) - 指定池化核尺寸大小。由一个整数或者是两个整数组成的 tuple，表示高和宽。默认值：1。
- **strides** (Union[int, tuple[int]]) - 池化操作的移动步长，由一个整数或者是两个整数组成的 tuple，表示高和宽上的移动步长。默认值：1。
- **pad_mode** (str, 可选) - 指定填充模式，填充值为 0。可选值为 "same" 或 "valid"。默认值："valid"。

- "same": 在输入的四周填充, 使得当 *stride* 为 1 时, 输入和输出的 *shape* 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在底部/右侧。
- "valid": 不对输入进行填充, 返回输出可能的最大高度和宽度, 不能构成一个完整 *stride* 的额外的像素将被丢弃。
- **data_format** (str) - 输入和输出的数据格式。可选值为 "NHWC" 或 "NCHW"。默认值: "NCHW"。

输入:

- **x** (Tensor) - *shape* 为 $(N, C_{in}, H_{in}, W_{in})$ 的 Tensor。支持数据类型:
 - CPU: float16、float32、float64。
 - GPU/Ascend: float16、float32。

输出:

Tensor, *shape* 为 $(N, C_{out}, H_{out}, W_{out})$ 。

异常:

- **TypeError** - *kernel_size* 或 *strides* 既不是 int 也不是 tuple。
- **ValueError** - *pad_mode* 既不是 "valid" 也不是 "same" (不区分大小写)。
- **ValueError** - *data_format* 既不是 "NCHW" 也不是 "NHWC"。
- **ValueError** - *kernel_size* 或 *strides* 小于 1。
- **ValueError** - *input* 的 *shape* 长度不等于 4。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.arange(1 * 3 * 3 * 4).reshape((1, 3, 3, 4)), mindspore.float32)
>>> maxpool_op = ops.MaxPool(pad_mode="VALID", kernel_size=2, strides=1)
>>> output = maxpool_op(x)
>>> print(output)
[[[ [ 5.  6.  7.]
      [ 9. 10. 11.]
      [17. 18. 19.]
      [21. 22. 23.]
      [29. 30. 31.]
      [33. 34. 35.]]]]]
```

mindspore.ops.MaxPool3D

class mindspore.ops.**MaxPool3D** (*kernel_size=1, strides=1, pad_mode='VALID', pad_list=0, ceil_mode=None, data_format='NCDHW'*)

对输入的多维数据进行三维的最大池化运算。

一般, 输入 *shape* 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor, 输出 (D_{in}, H_{in}, W_{in}) 维上的区域最大值。给定 *kernel_size* 为 $ks = (d_{ker}, h_{ker}, w_{ker})$ 和 *stride* 为 $s = (s_0, s_1, s_2)$, 运算如下:

$$\text{output}(N_i, C_j, d, h, w) = \max_{l=0, \dots, d_{ker}-1} \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} \text{input}(N_i, C_j, s_0 \times d + l, s_1 \times h + m, s_2 \times w + n)$$

说明:

- Atlas 训练系列产品暂不支持此算子原语。

参数:

- **kernel_size** (Union[int, tuple[int]]) - 指定池化核尺寸大小。整数类型, 表示池化核深度、高和宽, 或者是三个整数组成的元组, 表示深、高和宽。默认值: 1。
- **strides** (Union[int, tuple[int]]) - 池化操作的移动步长, 整数类型, 表示深、高和宽的移动步长, 或者是三个整数组成的元组, 表示深、高和宽移动步长。默认值: 1。
- **pad_mode** (str, 可选) - 指定填充模式, 填充值为 0。可选值为 "SAME", "VALID" 或 "PAD"。默认值: "VALID"。
 - "SAME": 在输入的深度、高度和宽度维度进行填充, 使得当 *stride* 为 1 时, 输入和输出的 *shape* 一致。待填充的量由算子内部计算, 若为偶数, 则均匀地填充在四周, 若为奇数, 多余的填充量将补充在前方/底部/右侧。如果设置了此模式, *pad_list* 必须为 0。
 - "VALID": 不对输入进行填充, 返回输出可能的最大深度、高度和宽度, 不能构成一个完整 *stride* 的额外的像素将被丢弃。如果设置了此模式, *pad_list* 必须为 0。
 - "PAD": 对输入填充指定的量。在这种模式下, 在输入的深度、高度和宽度方向上填充的量由 *pad_list* 参数指定。如果设置此模式, *pad_list* 必须大于或等于 0。
- **pad_list** (Union(int, tuple[int])) - 池化填充方式。默认值: 0。如果 *pad* 是一个整数, 则头尾部、顶部、底部、左边和右边的填充都是相同的, 等于 *pad*。如果 *pad* 是六个整数的 tuple, 则头尾部、顶部、底部、左边和右边的填充分别等于填充 *pad*[0]、*pad*[1]、*pad*[2]、*pad*[3]、*pad*[4] 和 *pad*[5]。
- **ceil_mode** (Union[bool, None]) - 是否使用 *ceil* 函数计算输出高度和宽度。默认值: None。
- **data_format** (str) - 输入和输出的数据格式。目前仅支持 "NCDHW"。默认值: "NCDHW"。

输入:

- **x** (Tensor) - *shape* 为 $(N, C, D_{in}, H_{in}, W_{in})$ 的 Tensor。数据类型为 float16 或 float32。

输出:

Tensor, *shape* 为 $(N, C, D_{out}, H_{out}, W_{out})$ 。数据类型与 *x* 相同。

异常:

- **TypeError** - *kernel_size* 或 *strides* 既不是 int 也不是元组。
- **TypeError** - *pad_mode* 或 *data_format* 不是 str。
- **ValueError** - *kernel_size* 或 *strides* 不是正数。
- **ValueError** - *pad_mode* 不是 "SAME"、"VALID"、或 "PAD"。
- **ValueError** - *pad_mode* 取值为 "SAME" 或 "VALID", *ceil_mode* 取值不是 None。
- **ValueError** - *kernel_size* 或 *strides* 是长度不等于 3 的元组。
- **ValueError** - *data_format* 不是 "NCDHW"。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.arange(1 * 2 * 2 * 2 * 3).reshape((1, 2, 2, 2, 3)), mindspore.float32)
>>> max_pool3d = ops.MaxPool3D(kernel_size=2, strides=1, pad_mode="VALID")
>>> output = max_pool3d(x)
>>> print(output)
[[[[[10. 11.]]]
  [[22. 23.]]]]]
```

mindspore.ops.MaxPool3DWithArgmax

class mindspore.ops.MaxPool3DWithArgmax (*ksize, strides, pads, dilation=(1, 1, 1), ceil_mode=False, data_format='NCDHW', argmax_type=mstype.int64*)

三维最大值池化，返回最大值结果及其索引值。

输入是 shape 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor，输出 $(D_{out}, H_{out}, W_{out})$ 维度中的最大值。给定 *ksize* $ks = (d_{ker}, h_{ker}, w_{ker})$ ，和 *strides* $s = (s_0, s_1, s_2)$ ，运算如下：

$$\text{output}(N_i, C_j, d, h, w) = \max_{l=0, \dots, d_{ker}-1} \max_{m=0, \dots, h_{ker}-1} \max_{n=0, \dots, w_{ker}-1} \text{input}(N_i, C_j, s_0 \times d + l, s_1 \times h + m, s_2 \times w + n)$$

输出是 shape 为 $(N_{out}, C_{out}, D_{out}, H_{out}, W_{out})$ 的 Tensor，其深度、高度和宽度的计算公式如下：

$$\begin{aligned} D_{out} &= \frac{D_{in} + 2 \times \text{pads}[0] - \text{dilation}[0] \times (\text{ksize}[0] - 1) - 1}{\text{stride}[0]} + 1 \\ H_{out} &= \frac{H_{in} + 2 \times \text{pads}[1] - \text{dilation}[1] \times (\text{ksize}[1] - 1) - 1}{\text{stride}[1]} + 1 \\ W_{out} &= \frac{W_{in} + 2 \times \text{pads}[2] - \text{dilation}[2] \times (\text{ksize}[2] - 1) - 1}{\text{stride}[2]} + 1 \end{aligned}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **ksize** (Union[int, tuple[int]]) - 池化核尺寸大小。可以是一个整数表示池化核的深度，高度和宽度，或者包含三个整数的 tuple，分别表示池化核的深度，高度和宽度。
- **strides** (Union[int, tuple[int]]) - 池化操作的移动步长。可以是一个整数表示在深度，高度和宽度方向的移动步长，或者包含三个整数的 tuple，分别表示在深度，高度和宽度方向的移动步长。
- **pads** (Union[int, tuple[int]]) - 池化填充长度。可以是一个整数表示在深度，高度和宽度方向的填充长度，或者包含三个整数的 tuple，分别表示在深度，高度和宽度方向的填充长度。
- **dilation** (Union[int, tuple[int]]) - 控制池化核内元素的间距。默认为 (1, 1, 1)。
- **ceil_mode** (bool) - 是否使用 ceil 代替 floor 来计算输出的 shape。默认为 False。
- **data_format** (str) - 选择输入数据格式，当前仅支持 'NCDHW'。默认为 'NCDHW'。
- **argmax_type** (mindspore.dtype) - 返回的最大值索引的数据类型。默认为 mstype.int64。

输入：

- **x** (Tensor) - shape 为 $(N_{in}, C_{in}, D_{in}, H_{in}, W_{in})$ 的 Tensor。支持数据类型包括 int8、int16、int32、int64、uint8、uint16、uint32、uint64、float16、float32 和 float64。

输出：

包含两个 Tensor 的 tuple，分别表示最大值结果和最大值对应的索引。

- **output** (Tensor) - 输出的池化后的最大值，shape 为 $(N_{out}, C_{out}, D_{out}, H_{out}, W_{out})$ 。其数据类型与 x 相同。
- **argmax** (Tensor) - 输出的最大值对应的索引，数据类型为 int32 或者 int64。

异常：

- **TypeError** - x 不是 Tensor。
- **ValueError** - x 的维度不是 5D。
- **TypeError** - $ksize$ 、 $strides$ 、 $pads$ 、 $dilation$ 不是 int 或者 tuple。
- **ValueError** - $ksize$ 或 $strides$ 的元素值小于 1。
- **ValueError** - $pads$ 的元素值小于 0。
- **ValueError** - $data\_format$ 不是 'NCDHW'。
- **ValueError** - $argmax\_type$ 不是 mstype.int64 或 mstype.int32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.arange(2 * 1 * 2 * 2 * 2).reshape((2, 1, 2, 2, 2)), mindspore.float32)
>>> max_pool3d_with_arg_op = ops.MaxPool3DWithArgmax(ksize=2, strides=1, pads=1)
>>> output_tensor, argmax = max_pool3d_with_arg_op(x)
>>> print(output_tensor.shape)
(2, 1, 3, 3, 3)
>>> print(argmax.shape)
(2, 1, 3, 3, 3)
```

mindspore.ops.MaxUnpool2D

class mindspore.ops.MaxUnpool2D ($ksize, strides=0, pads=0, output\_shape=(), data\_format='NCHW'$)
MaxPool2D 的逆过程。

由于 MaxPool2D 会丢失非最大值，因此它不是完全可逆的。MaxUnpool2D 将 MaxPool2D 的输出作为输入，包括最大值的索引，并计算部分逆，其中所有非最大值都被设置为零。例如，输入的 shape 为 (N, C, H_{in}, W_{in}) ，输出的 shape 为 (N, C, H_{out}, W_{out}) ，则计算过程如下所示：

$$\begin{aligned} H_{out} &= (H_{in} - 1) \times strides[0] - 2 \times pads[0] + ksize[0] \\ W_{out} &= (W_{in} - 1) \times strides[1] - 2 \times pads[1] + ksize[1] \end{aligned}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **ksize** (Union[int, tuple[int]]) - 用于取最大值的内核大小，是一个整数，表示内核的高度和宽度，或一个元组包含两个整数，分别表示内核的高度和宽度。

- **strides** (Union[int, tuple[int]], 可选) - 内核移动的步长, 如果 *strides* 为 0 或 (0, 0), 那么 *strides* 等于 *ksize*。默认值: 0。
 - 一个整数, 则表示移动的高度和宽度都是 *strides*。
 - 两个整数的元组, 分别表示高度和宽度方向的移动步长。
- **pads** (Union[int, tuple[int]], 可选) - 指定填充量。默认值: 0。
 - 如果 *pads* 是一个整数, 则高度和宽度方向的填充量相同, 都等于 *pads*。
 - 如果 *pads* 是含两个整数的元组, 高度和宽度方向的填充量分别等于 *pads[0]* 和 *pads[1]*。
- **output_shape** (tuple[int], 可选) - 一个可选的输入, 指定目标输出的尺寸。默认值: ()。
 - 如果 *output_shape* == (), 则输出的 *shape* 由 *ksize*、*strides* 和 *pads* 计算得到。
 - 如果 *output_shape* != (), 则 *output_shape* 必须为 (N, C, H, W) 或 (N, H, W, C) , 同时 *output_shape* 必须属于 $[(N, C, H_{out} - strides[0], W_{out} - strides[1]), (N, C, H_{out} + strides[0], W_{out} + strides[1])]$ 。
- **data_format** (str, 可选) - 可选的数据格式。当前支持 "NCHW" 和 "NHWC"。默认值: "NCHW"。

输入:

- **x** (Tensor) - 求逆的输入 Tensor。其 *shape* 为 (N, C, H_{in}, W_{in}) 或 (N, H_{in}, W_{in}, C) 。
- **argmax** (Tensor) - *argmax* 表示最大值的索引。其 *shape* 必须与输入 *x* 相同。*argmax* 的值必须属于 $[0, H_{in} \times W_{in} - 1]$, 数据类型必须为 int32 或 int64。

输出:

Tensor, 其 *shape* 为 (N, C, H_{out}, W_{out}) 或 (N, H_{out}, W_{out}, C) , 与 *x* 的数据类型相同。

异常:

- **TypeError** - 如果 *x* 或 *argmax* 的数据类型不支持。
- **TypeError** - 如果 *ksize*、*strides* 或 *pads* 既不是 int 又不是 tuple。
- **ValueError** - 如果 *strides* (也支持 0 和 (0, 0)) 或 *ksize* 的值不是正数。
- **ValueError** - 如果 *pads* 的值是负数。
- **ValueError** - 如果 *ksize*、*strides* 或 *pads* 不是长度为 2 的 tuple。
- **ValueError** - 如果 *data_format* 不是 str, 同时也不是 "NCHW" 或 "NHWC"。
- **ValueError** - 如果 *output_shape* 的长度不是 0 或 4。
- **ValueError** - 如果 *output_shape* 不在由 *ksize*、*strides* 和 *pads* 计算出的输出尺寸范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> argmax = Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> maxunpool2d = ops.MaxUnpool2D(ksize=1, strides=1, pads=0)
>>> output = maxunpool2d(x, argmax)
>>> print(output.asnumpy())
[[[0. 1.]
  [8. 9.]]]
```

mindspore.ops.MaxUnpool3D

class mindspore.ops.MaxUnpool3D (*ksize, strides=0, pads=0, output_shape=(), data_format='NCDHW'*)
mindspore.ops.MaxPool3D 的逆过程。

MaxUnpool3D 在计算逆的过程中，保留最大值位置的元素，并将非最大值位置元素设置为 0。通常情况下，输入数据格式为 $(N, C, D_{in}, H_{in}, W_{in})$ ，输出数据的格式为 $(N, C, D_{out}, H_{out}, W_{out})$ ，计算公式如下：

$$\begin{aligned} D_{out} &= (D_{in} - 1) \times strides[0] - 2 \times pads[0] + ksize[0] \\ H_{out} &= (H_{in} - 1) \times strides[1] - 2 \times pads[1] + ksize[1] \\ W_{out} &= (W_{in} - 1) \times strides[2] - 2 \times pads[2] + ksize[2] \end{aligned}$$

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **ksize** (Union[int, tuple[int]]) - 用于取最大值的内核大小。是一个整数，表示内核的深度、高度和宽度；或一个元组包含三个整数，分别表示深度、高度和宽度。
- **strides** (Union[int, tuple[int]])，可选 - 内核移动的步长，默认值：0。
 - 一个整数，则表示移动的深度、高度和宽度都是 *strides*，
 - 三个整数的元组，分别表示深度、高度和宽度方向的移动步长。
 - 如果 *strides* 为 0 或 (0, 0, 0)，那么 *strides* 等于 *ksize*。
- **pads** (Union[int, tuple[int]])，可选 - 指定填充量。默认值：0。
 - 如果 *pads* 是一个整数，则深度、高度和宽度方向的填充数量相同，都等于 *pads*。
 - 如果 *pads* 是含三个整数的元组，则深度、高度和宽度方向的填充数量分别等于 *pads*[0]、*pads*[1] 和 *pads*[2]。
- **output_shape** (tuple[int])，可选 - 指定目标输出的尺寸。默认值：()。
 - 如果 *output_shape* == ()，则输出的 shape 由 *ksize*、*strides* 和 *pads* 根据上面的公式计算得到。
 - 如果 *output_shape* != ()，则 *output_shape* 数据格式为 (N, C, D, H, W) 或 (N, D, H, W, C) ，同时 *output_shape* 必须属于 $[(N, C, D_{out} - strides[0], H_{out} - strides[1], W_{out} - strides[2]), (N, C, D_{out} + strides[0], H_{out} + strides[1], W_{out} + strides[2])]$ 的范围。
- **data_format** (str, 可选) - 可选的数据格式。当前支持 'NCDHW' 和 'NDHWC'。默认值：'NCDHW'。

输入：

- **x** (Tensor) - 求逆的输入 Tensor。其 shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 或 $(N, D_{in}, H_{in}, W_{in}, C)$ 。
- **argmax** (Tensor) - *argmax* 表示最大值的索引。其 shape 必须与输入 *x* 相同。*argmax* 的值必须在 $[0, D_{in} \times H_{in} \times W_{in} - 1]$ 范围内，数据类型必须为 int32 或 int64。

输出：

Tensor，其 shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ 或 $(N, D_{out}, H_{out}, W_{out}, C)$ ，与 *x* 的数据类型相同。

异常：

- **TypeError** - 如果 *x* 或 *argmax* 的数据类型不是 Number。
- **TypeError** - 如果 *ksize*、*strides* 或 *pads* 既不是 int 又不是 tuple。
- **ValueError** - 如果 *strides* 或 *ksize* 的值是负数。

- **ValueError** - 如果 *pads* 的值是负数。
- **ValueError** - 如果 *ksize*、*strides* 或 *pads* 是 tuple, 但长度不是 3。
- **ValueError** - 如果 *data_format* 不是值为 'NCDHW' 或 'NDHWC' 的 str。
- **ValueError** - 如果 *output_shape* 的长度不是 0 或 5。
- **ValueError** - 如果 *output_shape* 不在由 *ksize*、*strides* 和 *pads* 计算出的输出尺寸范围内。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[[0, 1], [8, 9]]]]).astype(np.float32))
>>> argmax = Tensor(np.array([[[[0, 1], [2, 3]]]]).astype(np.int64))
>>> maxunpool3d = ops.MaxUnpool3D(ksize=1, strides=1, pads=0)
>>> output = maxunpool3d(x, argmax)
>>> print(output.asnumpy())
[[[[[0. 1.]
      [8. 9.]]]]]
```

mindspore.ops.MirrorPad

class mindspore.ops.MirrorPad (*mode='REFLECT'*)

通过指定的填充模式和大小对输入 Tensor 进行填充。

参数:

- **mode** (str, 可选) - 指定填充模式的可选字符串。可选值为: 'REFLECT' 和 'SYMMETRIC'。默认值: 'REFLECT'。当采样 *grid* 超出输入 Tensor 的边界时, 各种填充模式效果如下:
 - 'REFLECT': 使用 Tensor 边缘像素的值 (不包含边界像素值) 填充输入 Tensor。例如, 向 [1, 2, 3, 4] 的两边分别填充 2 个元素, 结果为 [3, 2, 1, 2, 3, 4, 3, 2]。
 - 'SYMMETRIC': 使用 Tensor 边缘像素的值 (包含边界像素值) 填充输入 Tensor。例如, 向 [1, 2, 3, 4] 的两边分别填充 2 个元素, 结果为 [2, 1, 1, 2, 3, 4, 4, 3]。

输入:

- **input_x** (Tensor) - shape: (*N*, *), 其中 * 表示任何数量的附加维度。
- **paddings** (Tensor) - shape 为 (*N*, 2) 的矩阵。*N* 为输入 Tensor 的秩。int 类型。对于输入的第 *D* 个维度, *paddings*[*D*, 0] 表示需在输入第 *D* 维头部填充的数量, *paddings*[*D*, 1] 表示需在输入第 *D* 维尾部填充的数量。

输出:

填充后的 Tensor。

- 如果设置 *mode* 为 'REFLECT', 将使用对称轴对称复制的方式进行填充。如果 *input_x* 为 [[1,2,3], [4,5,6], [7,8,9]], *paddings* 为 [[1,1], [2,2]], 则输出为 [[6,5,4,5,6,5,4], [3,2,1,2,3,2,1], [6,5,4,5,6,5,4], [9,8,7,8,9,8,7], [6,5,4,5,6,5,4]]。更直观的理解请参见下面的样例。
- 如果 *mode* 为 'SYMMETRIC', 则填充方法类似于 'REFLECT'。它也会根据对称轴复制, 但是也包括对称轴。如果 *input_x* 为 [[1,2,3],[4,5,6],[7,8,9]], *paddings* 为 [[1,1], [2,2]], 则输出为 [[2,1,1,2,3,3,2], [2,1,1,2,3,3,2], [5,4,4,5,6,6,5], [8,7,7,8,9,9,8], [8,7,7,8,9,9,8]]。更直观的理解请参见下面的样例。

异常:

- **TypeError** - *input_x* 或 *padings* 不是 Tensor。
- **TypeError** - *mode* 不是 str。
- **ValueError** - *padings.size* 不等于 $2 * \text{len}(\text{input\_x})$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, nn, ops
>>> # case1: mode="REFLECT"
>>> class Net(nn.Cell):
...     def __init__(self, mode):
...         super(Net, self).__init__()
...         self.pad = ops.MirrorPad(mode=mode)
...         self.paddings = Tensor([[1, 1], [2, 2]])
...     def construct(self, input_x):
...         return self.pad(input_x, self.paddings)
...
>>> input_x = Tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> pad = Net("REFLECT")
>>> output = pad(input_x)
>>> print(output)
[[6 5 4 5 6 5 4]
 [3 2 1 2 3 2 1]
 [6 5 4 5 6 5 4]
 [9 8 7 8 9 8 7]
 [6 5 4 5 6 5 4]]
>>> # case2: mode="SYMMETRIC"
>>> pad = Net("SYMMETRIC")
>>> output = pad(input_x)
>>> print(output)
[[2 1 1 2 3 3 2]
 [2 1 1 2 3 3 2]
 [5 4 4 5 6 6 5]
 [8 7 7 8 9 9 8]
 [8 7 7 8 9 9 8]]
```

mindspore.ops.Pad

class mindspore.ops.Pad(*paddings*)

根据参数 *paddings* 对输入进行填充。

更多参考详见 [mindspore.ops.pad\(\)](#)。如果 *paddings* 里存在负数值，请使用 [mindspore.ops.pad\(\)](#) 接口。

参数：

- **paddings** (tuple) - 填充大小，其 shape 为 (N, 2)，N 是输入数据的维度，填充的元素为 int 类型。对于 *x* 的第 *D* 个维度，*paddings*[D, 0] 表示输入 Tensor 的第 *D* 维度前面要扩展的大小，*paddings*[D, 1] 表示在输入 Tensor 的第 *D* 个维度后面要扩展的大小。

输入：

- **input_x** (Tensor) - 被填充的 Tensor，其 shape 为 (N, *)，其中 * 为任意数量的额外维度。

输出：

填充后的 Tensor。

异常：

- **TypeError** - *paddings* 不是 tuple。
- **TypeError** - *input_x* 不是 Tensor。
- **ValueError** - *paddings* 的 shape 不是 $(N, 2)$ 。
- **ValueError** - *paddings* 的大小不等于 $2 * \text{len}(\text{input\_x})$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]], mindspore.float32)
>>> pad_op = ops.Pad(((1, 2), (2, 1)))
>>> output = pad_op(input_x)
>>> print(output)
[[ 0.  0.  0.  0.  0.  0. ]
 [ 0.  0. -0.1 0.3  3.6  0. ]
 [ 0.  0.  0.4 0.5 -3.2  0. ]
 [ 0.  0.  0.  0.  0.  0. ]
 [ 0.  0.  0.  0.  0.  0. ]]
```

mindspore.ops.Padding

class mindspore.ops.Padding (*pad_dim_size=8*)

将输入 Tensor 的最后一个维度从 1 扩展到 *pad_dim_size*，其填充值为 0。

更多参考详见 `mindspore.ops.padding()`。

参数：

- **pad_dim_size** (int, 可选) - 指定填充的大小，待扩展的 *x* 的最后一个维度的值，必须为正数。默认值：8。

输入：

- **x** (Tensor) - 输入 Tensor，二维或者更高维 Tensor，*x* 的最后一个维度必须为 1。数据类型为 Number。

输出：

填充后的 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[8], [10]]), mindspore.float32)
>>> pad_dim_size = 4
>>> output = ops.Padding(pad_dim_size)(x)
>>> print(output)
[[ 8.  0.  0.  0.]
 [10.  0.  0.  0.]]
```


mindspore.ops.ResizeBicubic

class mindspore.ops.ResizeBicubic (align_corners=False, half_pixel_centers=False)

使用双三次插值调整图像大小到指定的大小。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **align_corners** (bool, 可选) - 如果为 True，则输入输出图像四个角像素的中心被对齐，同时保留角像素处的值。默认值：False。
- **half_pixel_centers** (bool, 可选) - 是否使用半像素中心对齐。如果设置为 True，那么 *align_corners* 应该设置为 False。默认值：False。

输入：

- **images** (Tensor) - 输入图像为四维的 Tensor，其 shape 为 (batch, channels, height, width)，支持的数据类型有：float16、float32、float64。
- **size** (Union[tuple[int], Tensor[int]]) - tuple 或 1-D Tensor，含有两个元素，分别为 new_height、new_width。推荐使用 tuple[int]。

输出：

四维 Tensor，其 shape 为 (batch, channels, new_height, new_width)，且数据类型与 *images* 一致。

异常：

- **TypeError** - *images* 的数据类型不支持。
- **TypeError** - *align_corners* 不是 bool 类型。
- **TypeError** - *half_pixel_centers* 不是 bool 类型。
- **ValueError** - *images* 的维度不是 4。
- **ValueError** - 当 *size* 是 Tensor 时，其维度不是 1。
- **ValueError** - *size* 所含元素的个数不是 2。
- **ValueError** - *size* 中的元素不全是正数。
- **ValueError** - *align_corners* 和 *half_pixel_centers* 同时为 True。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, nn
>>> class NetResizeBicubic(nn.Cell):
...     def __init__(self):
...         super(NetResizeBicubic, self).__init__()
...         align_corners = False
...         half_pixel_centers = False
...         self.resize = ops.ResizeBicubic(align_corners, half_pixel_centers)
... 
```

(下页继续)

(续上页)

```

...     def construct(self, images, size):
...         return self.resize(images, size)
...
>>> images = Tensor(np.array([1, 2, 3, 4]).reshape(1, 1, 2, 2).astype(np.float32))
>>> size = Tensor([1, 4], mindspore.int32)
>>> resizebicubic = NetResizeBicubic()
>>> output = resizebicubic(images, size)
>>> print(output)
[[[1. 1.5 2. 2.09375]]]]

```

mindspore.ops.ResizeNearestNeighbor

class mindspore.ops.ResizeNearestNeighbor (*size, align_corners=False, half_pixel_centers=False*)

使用最近邻插值算法调整输入 Tensor 为指定大小。在计算分段常数插值时，最近邻插值算法会选择最近点的值，而不考虑相邻点的值。

参数：

- **size** (Union[tuple, list]) - 指定输入 Tensor 的新大小。size 的维度必须为 2。
- **align_corners** (bool, 可选) - 输入和输出 Tensor 的 4 个角像素是否居中对齐。默认值：False。
- **half_pixel_centers** (bool, 可选) - 是否使用半像素中心对齐。如果设置为 True，那么 *align_corners* 应该设置为 False。默认值：False。

输入：

- **input_x** (Tensor) - ResizeNearestNeighbor 的输入，四维的 Tensor，其 shape 为 (N, C, H, W) 。

输出：

Tensor，输出 Tensor 的 shape 为 (N, C, NEW_H, NEW_W) 。数据类型与 *input_x* 相同。

异常：

- **TypeError** - *size* 既不是 tuple，也不是 list。
- **TypeError** - *align_corners* 不是 bool。
- **ValueError** - *size* 的 shape 长度不等于 2。

支持平台：

Ascend GPU CPU

样例：

```

>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input_tensor = Tensor(np.array([[[[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]]]), mindspore.
↳ float32)
>>> size = (2, 2)
>>> output = ops.ResizeNearestNeighbor(size=size)(input_tensor)
>>> print(output)
[[[[-0.1  0.3]
      [ 0.4  0.5]]]]

```

`mindspore.ops.UpsampleNearest3D`

`class mindspore.ops.UpsampleNearest3D`

执行最近邻上采样操作。

此运算符使用指定的 `output_size` 或 `scales` 缩放因子放大输入体积，过程使用最近邻算法。

必须指定 `output_size` 或 `scales` 中的一个值，并且不能同时指定两者。

输入：

- **x** (Tensor) - Shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 的五维 Tensor。支持的数据类型：[uint8, float16, float32, float64]。
- **output_size** (Union[tuple[int], list[int]]) - 指定输出体积大小的元组或 int 列表。默认值：None。
- **scales** (Union[tuple[float], list[float]]) - 指定上采样因子的元组或 float 列表。默认值：None。

输出：

- **y** (Tensor) - 上采样输出。其 shape $(N, C, D_{out}, H_{out}, W_{out})$ ，数据类型与 *x* 相同。

异常：

- **TypeError** - 当 `output_size` 不是 None 并且 `output_size` 不是 list[int] 或 tuple[int]。
- **TypeError** - 当 `scales` 不是 None 并且 `scales` 不是 list[float] 或 tuple[float]。
- **TypeError** - *x* 的数据类型不是 uint8、float16、float32 或 float64。
- **ValueError** - `output_size` 不为 None 时含有负数或 0。
- **ValueError** - `scales` 不为 None 时含有负数或 0。
- **ValueError** - *x* 维度不为 5。
- **ValueError** - `scales` 和 `output_size` 同时被指定或都不被指定。
- **ValueError** - `scales` 被指定时其含有的元素个数不为 3。
- **ValueError** - `output_size` 被指定时其含有的元素个数不为 3。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]))
...     .reshape([1, 1, 2, 2, 4]), mstype.float32)
>>> output_size = [3, 4, 5]
>>> net = ops.UpsampleNearest3D()
>>> output = net(x, output_size, None)
>>> print(output)
[[[[[ 1.  1.  2.  3.  4.]
      [ 1.  1.  2.  3.  4.]
      [ 5.  5.  6.  7.  8.]
      [ 5.  5.  6.  7.  8.]]
  [[ 1.  1.  2.  3.  4.]
    [ 1.  1.  2.  3.  4.]
    [ 5.  5.  6.  7.  8.]
    [ 5.  5.  6.  7.  8.]]
  [[ 9.  9. 10. 11. 12.]
```

(下页继续)

(续上页)

```
[ 9.  9. 10. 11. 12.]
[13. 13. 14. 15. 16.]
[13. 13. 14. 15. 16.]]]]]
```

mindspore.ops.UpsampleTrilinear3D

class mindspore.ops.UpsampleTrilinear3D (*align_corners=False*)

输入为五维度 Tensor，跨其中三维执行三线性插值上调采样。

此运算符使用指定的 *output_size* 或 *scales* 缩放因子放大输入体积，过程使用三线性上调算法。

说明：必须指定 *output_size* 或 *scales* 中的一个值，并且不能同时指定两者。

参数：

- **align_corners** (bool, 可选) - 如果为 True，则输入和输出 Tensor 由其角像素的中心点对齐，保留角像素处的值。如果为 False，则输入和输出 Tensor 由其角像素的角点对齐，对边界外值进行插值时使用边值填充。默认值：False。

输入：

- **x** (Tensor) - Shape 为 $(N, C, D_{in}, H_{in}, W_{in})$ 的五维 Tensor。支持的数据类型：[float16, float32, float64]。
- **output_size** (Union[tuple[int], list[int]]) - 包含 3 个 int 的元组或列表。元素分别为 $(output\_depth, output\_height, output\_width)$ 。默认值：None。
- **scales** (Union[tuple[float], list[float]]) - 包含 3 个 float 的元组或列表。元素分别为 $(scale\_depth, scale\_height, scale\_width)$ 。默认值：None。

输出：

- **y** (Tensor) - 上采样输出。其 shape 为 $(N, C, D_{out}, H_{out}, W_{out})$ ，数据类型与 *x* 相同。

异常：

- **TypeError** - 当 *output_size* 不是 None 并且 *output_size* 不是 list[int] 或 tuple[int]。
- **TypeError** - 当 *scales* 不是 None 并且 *scales* 不是 list[float] 或 tuple[float]。
- **TypeError** - *x* 的数据类型不是 float16、float32 或 float64。
- **TypeError** - *align_corners* 的数据类型不是 bool。
- **ValueError** - *output_size* 不为 None 时含有负数或 0。
- **ValueError** - *scales* 不为 None 时含有负数或 0。
- **ValueError** - *x* 维度不为 5。
- **ValueError** - *scales* 和 *output_size* 同时被指定或都未被指定。
- **ValueError** - *scales* 被指定时其含有的元素个数不为 3。
- **ValueError** - *output_size* 被指定时其含有的元素个数不为 3。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> net = ops.UpsampleTrilinear3D()
>>> in_x = Tensor(input_data=np.random.randn(2, 3, 4, 512, 256))
>>> output_size=[4, 64, 48]
>>> out = net(in_x, output_size, None)
>>> print(out.shape)
(2, 3, 4, 64, 48)
>>>
>>> net = ops.UpsampleTrilinear3D()
>>> in_x = Tensor(np.arange(1, 5, dtype=np.float32).reshape((1, 1, 1, 2, 2)))
>>> output_size=[2, 4, 4]
>>> out = net(in_x, output_size, None)
>>> print(out)
[[[[[1.   1.25 1.75 2.   ]
      [1.5   1.75 2.25 2.5 ]
      [2.5   2.75 3.25 3.5 ]
      [3.    3.25 3.75 4.   ]]
  [[1.   1.25 1.75 2.   ]
      [1.5   1.75 2.25 2.5 ]
      [2.5   2.75 3.25 3.5 ]
      [3.    3.25 3.75 4.   ]]]]]]
```

20.2.2 损失函数

接口名	概述	支持平台
<code>mindspore.ops.BCEWithLogitsLoss</code>	输入经过 <code>sigmoid</code> 激活函数计算后作为预测值， <code>BCEWithLogitsLoss</code> 计算预测值和目标值之间的二值交叉熵损失。	Ascend GPU CPU
<code>mindspore.ops.BinaryCrossEntropy</code>	计算目标值和预测值之间的二值交叉熵损失值。	Ascend GPU CPU
<code>mindspore.ops.CTCLoss</code>	计算 CTC (Connectionist Temporal Classification) 损失和梯度。	Ascend GPU CPU
<code>mindspore.ops.CTCLossV2</code>	计算 CTC (Connectionist Temporal Classification) 损失和梯度。	Ascend GPU CPU
<code>mindspore.ops.KLDivLoss</code>	计算输入 <code>logits</code> 和 <code>labels</code> 的 KL 散度。	Ascend GPU CPU
<code>mindspore.ops.L2Loss</code>	用于计算 L2 范数的一半，但不对结果进行开方操作。	Ascend GPU CPU
<code>mindspore.ops.MultilabelMarginLoss</code>	创建一个损失函数，用于最小化多分类任务的合页损失。	Ascend GPU
<code>mindspore.ops.MultiMarginLoss</code>	创建一个损失函数，用于优化输入和输出之间的多分类合页损失。	Ascend GPU CPU
<code>mindspore.ops.NLLLoss</code>	获取预测值和目标值之间的负对数似然损失。	Ascend GPU CPU
<code>mindspore.ops.RNNTLoss</code>	计算相对于 <code>softmax</code> 输出的 <code>RNNTLoss</code> 及其梯度。	Ascend
<code>mindspore.ops.SigmoidCrossEntropyWithLogits</code>	计算预测值与真实值之间的 <code>sigmoid</code> 交叉熵。	Ascend GPU CPU
<code>mindspore.ops.SmoothL1Loss</code>	计算平滑 L1 损失，该 L1 损失函数有稳健性。	Ascend GPU CPU
<code>mindspore.ops.SoftMarginLoss</code>	更多详情请查看： <code>mindspore.ops.soft_margin_loss()</code> 。	Ascend GPU

下页继续

表 3 - 续上页

<code>mindspore.ops.SoftmaxCrossEntropyWithLogits</code>	使用 one-hot 编码获取预测值和真实之间的 softmax 交叉熵。	Ascend GPU CPU
<code>mindspore.ops.SparseSoftmaxCrossEntropyWithLogits</code>	计算预测值和标签之间的稀疏 softmax 交叉熵。	GPU CPU
<code>mindspore.ops.TripletMarginLoss</code>	三元组损失函数。	GPU

mindspore.ops.BCEWithLogitsLoss

class mindspore.ops.BCEWithLogitsLoss (*reduction='mean'*)

输入经过 **sigmoid** 激活函数计算后作为预测值, *BCEWithLogitsLoss* 计算预测值和目标值之间的二值交叉熵损失。

将输入 *input* 设置为 *X*, 输入 *target* 设置为 *Y*, 输入 *weight* 设置为 *W*, 输出设置为 *L*。则,

$$p_{ij} = \text{sigmoid}(X_{ij}) = \frac{1}{1+e^{-X_{ij}}}$$

$$L_{ij} = -[Y_{ij} \log(p_{ij}) + (1 - Y_{ij}) \log(1 - p_{ij})]$$

i 表示 i^{th} 样例, *j* 表示类别。则,

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'}. \end{cases}$$

ℓ 表示计算损失的方法。有三种方法: 第一种方法是直接提供损失值, 第二种方法是计算所有损失的平均值, 第三种方法是计算所有损失的总和。

该算子会将输出乘以相应的权重。*weight* 表示一个 batch 中的每条数据分配不同的权重, *pos_weight* 为每个类别的正例添加相应的权重。

此外, 它可以通过向正例添加权重来权衡召回率和精度。在多标签分类的情况下, 损失可以描述为:

$$p_{ij,c} = \text{sigmoid}(X_{ij,c}) = \frac{1}{1+e^{-X_{ij,c}}}$$

$$L_{ij,c} = -[P_c Y_{ij,c} * \log(p_{ij,c}) + (1 - Y_{ij,c}) \log(1 - p_{ij,c})]$$

其中 *c* 是类别数目 (*c*>1 表示多标签二元分类, *c*=1 表示单标签二元分类), *n* 是批次中样本的数量, P_c 是第 *c* 类正例的权重。 $P_c > 1$ 增大召回率, $P_c < 1$ 增大精度。

参数:

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的加权平均值。
 - 'sum': 计算输出元素的总和。

输入:

- **input** (Tensor) - 输入预测值, 其 shape 为 (*N*, *), 其中 * 为任意数量的额外维度。数据类型必须为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。
- **target** (Tensor) - 输入目标值, shape 与 *input* 相同。数据类型为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。
- **weight** (Tensor) - 指定每个批次二值交叉熵的权重。支持广播, 使其 shape 与 *input* 的 shape 保持一致。数据类型必须为 float16、float32 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。

- **pos_weight** (Tensor) - 指定正类的权重。是一个长度等于分类数的向量。支持广播，使其 shape 与 *input* 的 shape 保持一致。数据类型必须为 float16、float32 或 bfloat16（仅 Atlas A2 训练系列产品支持）。

输出：

Tensor 或 Scalar，如果 *reduction* 为 'none'，则为 shape 和数据类型与输入 *input* 相同的 Tensor。否则，输出为 Scalar。

异常：

- **TypeError** - 所有的输入都不是 Tensor。
- **TypeError** - 所有输入的数据类型不是 float16、float32 或 bfloat16。
- **TypeError** - *reduction* 的数据类型不是 string。
- **ValueError** - *weight* 或 *pos_weight* 不能广播到 shape 为 *input* 的 Tensor。
- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([[[-0.8, 1.2, 0.7], [-0.1, -0.4, 0.7]]], mindspore.float32)
>>> target = Tensor(np.array([[0.3, 0.8, 1.2], [-0.6, 0.1, 2.2]]), mindspore.float32)
>>> weight = Tensor(np.array([1.0, 1.0, 1.0]), mindspore.float32)
>>> pos_weight = Tensor(np.array([1.0, 1.0, 1.0]), mindspore.float32)
>>> loss = ops.BCEWithLogitsLoss()
>>> output = loss(input, target, weight, pos_weight)
>>> print(output)
0.3463612
```

mindspore.ops.BinaryCrossEntropy

class mindspore.ops.BinaryCrossEntropy (*reduction='mean'*)

计算目标值和预测值之间的二值交叉熵损失值。

将 *logits* 设置为 *x*，*labels* 设置为 *y*，输出为 $\ell(x, y)$ 。则，

$$L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_n [y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n)]$$

其中，*L* 表示所有批次的损失，*l* 表示一个批次的损失，*n* 表示 1-N 范围内的一个批次，*w_n* 表示第 *n* 批二进制交叉熵的权重。则，

$$\ell(x, y) = \begin{cases} L, & \text{if reduction = 'none';} \\ \text{mean}(L), & \text{if reduction = 'mean';} \\ \text{sum}(L), & \text{if reduction = 'sum'.} \end{cases}$$

警告：

- *x* 的值必须在 0 到 1 之间。

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的加权平均值。
 - "sum": 计算输出元素的总和。

输入:

- **logits** (Tensor) - 输入预测值。其 shape 为 $(N, *)$, 其中 $*$ 为任意数量的额外维度。
- **labels** (Tensor) - 输入目标值, 其 shape 和数据类型与 *logits* 相同。
- **weight** (Tensor, 可选) - 每个批次二值交叉熵的权重。且 shape 和数据类型必须与 *logits* 相同。默认值: None。

输出:

Tensor, 与 *logits* 有相同的数据类型。如果 *reduction* 为 'none', 则 shape 与 *logits* 相同。否则, 输出为 Scalar Tensor。

异常:

- **TypeError** - *logits*、*labels* 及 *weight* 的数据类型既不是 float16, 也不是 float32。
- **ValueError** - *reduction* 不为 'none'、'mean' 或 'sum'。
- **ValueError** - *labels* 的 shape 与 *logits* 或 *weight* 不同。
- **TypeError** - *logits*、*labels* 或 *weight* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.binary_cross_entropy = ops.BinaryCrossEntropy()
...     def construct(self, logits, labels, weight):
...         result = self.binary_cross_entropy(logits, labels, weight)
...         return result
...
>>> net = Net()
>>> logits = Tensor(np.array([0.2, 0.7, 0.1]), mindspore.float32)
>>> labels = Tensor(np.array([0., 1., 0.]), mindspore.float32)
>>> weight = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = net(logits, labels, weight)
>>> print(output)
0.38240486
```


mindspore.ops.CTCLoss

class mindspore.ops.CTCLoss (*preprocess_collapse_repeated=False, ctc_merge_repeated=True, ignore_longer_outputs_than_inputs=False*)

计算 CTC (Connectionist Temporal Classification) 损失和梯度。

该接口的底层调用了第三方 `baidu-research::warp-ctc` 的实现。CTC 算法是在 [Connectionist Temporal Classification: Labeling Unsegmented Sequence Data with Recurrent Neural Networks](#) 中提出的。

CTCLoss 计算连续时间序列和目标序列之间的损失。CTCLoss 对输入到目标的概率求和, 产生一个损失值, 该值相对于每个输入节点是可微的。假设输入与目标的对齐是“多对一”的, 这样目标序列的长度必须小于或等于输入的长度。

参数:

- **preprocess_collapse_repeated** (bool, 可选) - 如果为 True, 在 CTC 计算之前将折叠重复标签。默认值: False。
- **ctc_merge_repeated** (bool, 可选) - 如果为 False, 在 CTC 计算过程中, 重复的非空白标签不会被合并, 这些标签将被解释为单独的标签。这是 CTC 的简化版本。默认值: True。
- **ignore_longer_outputs_than_inputs** (bool, 可选) - 如果为 True, 则输出比输入长的序列将被忽略。默认值: False。

输入:

- **x** (Tensor) - 输入 Tensor, 是一个 shape 为 (*max_time, batch_size, num_classes*) 的三维 Tensor。 *num_classes* 表示类别数, 必须是 *num_labels + 1*, *num_labels* 表示实际标签的数量。保留空白标签。默认空白标签为 *num_classes - 1*。数据类型必须为 float16、float32 或 float64。
- **labels_indices** (Tensor) - 标签的索引。 *labels_indices[i, :] = [b, t]* 表示 *labels_values[i]* 存储 (*batch b, time t*) 的 ID。数据类型必须为 int64, 秩必须为 2。
- **labels_values** (Tensor) - 一维 Tensor。这些值与给定的 batch size 和时间相关联。数据类型必须为 int32。 *labels_values[i]* 必须在 *[0, num_classes)* 的范围内。
- **sequence_length** (Tensor) - 包含序列长度的 Tensor, shape 为 (*batch_size,*)。数据类型必须为 int32。Tensor 中的每个值不得大于最大时间。

输出:

- **loss** (Tensor) - 包含对数概率的 Tensor, shape 为 (*batch_size,*)。Tensor 的数据类型与 *x* 相同。
- **gradient** (Tensor) - *loss* 的梯度, shape 和数据类型与 *x* 相同。

异常:

- **TypeError** - *preprocess_collapse_repeated*、*ctc_merge_repeated* 或 *ignore_longer_outputs_than_inputs* 不是 bool。
- **TypeError** - *x*、*labels_indices*、*labels_values* 或 *sequence_length* 不是 Tensor。
- **ValueError** - *labels_indices* 的秩不等于 2。
- **TypeError** - *x* 的数据类型不是 float16、float32 或 float64。
- **TypeError** - *labels_indices* 的数据类型不是 int64。
- **TypeError** - *labels_values* 或 *sequence_length* 的数据类型不是 int32。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[0.3, 0.6, 0.6],
...                      [0.4, 0.3, 0.9]],
...             [[0.9, 0.4, 0.2],
...              [0.9, 0.9, 0.1]]).astype(np.float32))
>>> labels_indices = Tensor(np.array([[0, 0], [1, 0]]), mindspore.int64)
>>> labels_values = Tensor(np.array([2, 2]), mindspore.int32)
>>> sequence_length = Tensor(np.array([2, 2]), mindspore.int32)
>>> ctc_loss = ops.CTCLoss()
>>> loss, gradient = ctc_loss(x, labels_indices, labels_values, sequence_length)
>>> print(loss)
[ 0.79628  0.5995158 ]
>>> print(gradient)
[[[ 0.27029088  0.36485454 -0.6351454 ]
  [ 0.28140804  0.25462854 -0.5360366 ]
  [ 0.47548494  0.2883962  0.04510255 ]
  [ 0.4082751  0.4082751  0.02843709 ]]]
```

mindspore.ops.CTCLossV2

class mindspore.ops.CTCLossV2 (blank=0, reduction='none', zero_infinity=False)

计算 CTC (Connectionist Temporal Classification) 损失和梯度。

CTC 算法是在 [Connectionist Temporal Classification: Labeling Unsegmented Sequence Data with Recurrent Neural Networks](#) 中提出的。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **blank** (int, 可选) - 空白标签。默认值：0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式。目前仅支持 'none'，默认值：'none'。
- **zero_infinity** (bool, 可选) - 在损失无限大的时候，是否将无限损失和相关梯度置为零。默认值：False。

输入：

- **log_probs** (Tensor) - 输入 Tensor，是一个 shape 为 (T, N, C) 的三维 Tensor。 T 表示输入长度， N 表示批大小， C 表示类别数，包含空白标签。支持的数据类型：float32、float64。
- **targets** (Tensor) - 标签序列，是一个 shape 为 (N, S) 的二维 Tensor。 S 表示最大标签长度。支持的数据类型：int32、int64。
- **input_lengths** (Union(Tuple, Tensor)) - 输入的长度。其 shape 为 (N) 。支持的数据类型：int32、int64。
- **target_lengths** (Union(Tuple, Tensor)) - 标签的长度。其 shape 为 (N) 。支持的数据类型：int32、int64。

输出：

- **neg_log_likelihood** (Tensor) - 相对于每个输入节点可微分的损失值。
- **log_alpha** (Tensor) - 输入到目标的可能跟踪概率。

异常：

- **TypeError** - 如果 *zero_infinity* 不是 bool 类型。
- **TypeError** - 如果 *reduction* 不是 string 类型。
- **TypeError** - 如果 *log_probs* 的 dtype 不是 float 类型或 double 类型。
- **TypeError** - 如果 *targets*、*input_lengths* 或 *target_lengths* 的 dtype 不是 int32 类型或 int64 类型。
- **ValueError** - 如果 *log_probs* 的秩不等于 2。
- **ValueError** - 如果 *targets* 的秩不等于 2。
- **ValueError** - 如果 *input_lengths* 的 shape 与批大小 *N* 不匹配。
- **ValueError** - 如果 *targets* 的 shape 与批大小 *N* 不匹配。
- **TypeError** - 如果 *targets*、*input_lengths* 或 *target_lengths* 的类型不同。
- **ValueError** - 如果 *blank* 的数值不在 [0, C) 范围内。
- **RuntimeError** - 如果 *input_lengths* 中任意一个元素值大于 (num_labels|C)。
- **RuntimeError** - 如果任何 *target_lengths[i]* 不在范围 [0, *input_length[i]*] 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> log_probs = Tensor(np.array([[0.3, 0.6, 0.6]],
...                             [[0.9, 0.4, 0.2]]).astype(np.float32))
>>> targets = Tensor(np.array([[0, 1]]), mstype.int32)
>>> input_lengths = Tensor(np.array([2]), mstype.int32)
>>> target_lengths = Tensor(np.array([1]), mstype.int32)
>>> CTCLossV2 = ops.CTCLossV2(blank=0, reduction='none', zero_infinity=False)
>>> neg_log_hood, log_alpha = CTCLossV2(
...     log_probs, targets, input_lengths, target_lengths)
>>> print(neg_log_hood)
[-2.2986124]
>>> print(log_alpha)
[[[0.3      0.3      -inf      -inf      -inf]
  [1.2      1.8931472 1.2      -inf      -inf]]]
```

mindspore.ops.KLDivLoss

class mindspore.ops.KLDivLoss (*reduction='mean'*)

计算输入 *logits* 和 *labels* 的 KL 散度。

对于相同形状的张量 *x* 和 *target*，KLDivLoss 的计算公式如下：

$$L(x, target) = target \cdot (\log target - x)$$

输出

$$\ell(x, target) = \begin{cases} L(x, target), & \text{if reduction = 'none';} \\ \text{mean}(L(x, target)), & \text{if reduction = 'mean';} \\ \text{sum}(L(x, target))/x.\text{shape}[0], & \text{if reduction = 'batchmean';} \\ \text{sum}(L(x, target)), & \text{if reduction = 'sum'}. \end{cases}$$

其中 x 代表 *logits* ; *target* 代表 *labels* ; $\ell(x, target)$ 为 *output* 。

说明:

- 目前 Ascend 平台不支持 float64 数据类型。
- 仅当 *reduction* 设置为 'batchmean' 时输出才与 Kullback-Leibler 散度的数学定义一致。
- 在 Ascend 平台上, *reduction* 的可选值为 'batchmean'、'none' 或 'sum' 。
- 在 GPU 平台上, *reduction* 的可选值为 'mean'、'none' 或 'sum' 。
- 在 CPU 平台上, *reduction* 的可选值为 'mean'、'batchmean'、'none' 或 'sum' 。

参数:

- **reduction** (str, 可选) - 指定输出结果的计算方式。默认值: 'mean' 。
 - 'none': 不应用规约方法。
 - 'mean': 计算输出元素的平均值。
 - 'sum': 计算输出元素的总和。
 - 'batchmean': 计算批次的平均损失, 类似于 mean 模式。

输入:

- **logits** (Tensor) - 数据类型支持 float16、float32 和 float64。
- **labels** (Tensor) - 标签 Tensor, 与 *logits* 的 shape 和数据类型相同。

输出:

Tensor 或标量。如果 *reduction* 为 "none", 则输出为 Tensor 且与 *logits* 的 shape 相同, 否则为标量。

异常:

- **TypeError** - *reduction* 不是 str。
- **TypeError** - *logits* 或 *labels* 不是 Tensor。
- **TypeError** - *logits* 或 *labels* 的数据类型不是支持的类型。
- **ValueError** - *logits* 和 *labels* 的 shape 不一致。
- **RuntimeError** - *logits* 或 *labels* 是标量并且 *reduction* 是 'batchmean' 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.kldiv_loss = ops.KLDivLoss(reduction='sum')
...     def construct(self, logits, labels):
...         result = self.kldiv_loss(logits, labels)
...         return result
```

(下页继续)

(续上页)

```

...
>>> net = Net()
>>> logits = Tensor(np.array([0.2, 0.7, 0.1]), mindspore.float32)
>>> labels = Tensor(np.array([0., 1., 0.]), mindspore.float32)
>>> output = net(logits, labels)
>>> print(output)
-0.7

```

mindspore.ops.L2Loss

class mindspore.ops.L2Loss

用于计算 L2 范数的一半，但不结果进行开方操作。

把输入设为 x ，输出设为 $loss$ 。

$$loss = \frac{\sum x^2}{2}$$

输入：

- **input_x** (Tensor) - 用于计算 L2 范数的 Tensor。数据类型必须为 float16 或 float32。

输出：

Tensor，具有与 *input_x* 相同的数据类型的 Scalar Tensor。

异常：

- **TypeError** - *input_x* 不是 Tensor。
- **TypeError** - *input_x* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([1, 2, 3]), mindspore.float16)
>>> l2_loss = ops.L2Loss()
>>> output = l2_loss(input_x)
>>> print(output)
7.0

```

mindspore.ops.MultilabelMarginLoss

class mindspore.ops.MultilabelMarginLoss (reduction='mean')

创建一个损失函数，用于最小化多分类任务的合页损失。它以一个 2D mini-batch Tensor x 作为输入，以包含目标类索引的 2D Tensor y 作为输出。

更多细节请参考 `mindspore.ops.multilabel_margin_loss()`。

参数：

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。

- 'none': 不应用规约方法。
- 'mean': 计算输出元素的平均值。
- 'sum': 计算输出元素的总和。

输入:

- **x** (Tensor) - 预测值。shape 为 (C) 或 (N, C), 其中 *N* 为批量大小, *C* 为类别数。数据类型必须为: float16 或 float32。
- **target** (Tensor) - 真实标签, shape 与 *input* 相同, 数据类型必须为 int32, 标签目标由-1 填充。

输出:

- **y** (Union[Tensor, Scalar]) - MultilabelMarginLoss 损失。如果 *reduction* 的值为'none', 那么返回 shape 为 (N) 的 Tensor 类型数据。否则返回一个标量。
- **is_target** (Tensor) - 用于反向输入的 Tensor, 其 shape 与 *target* 一致, 数据类型为 int32。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> loss = ops.MultilabelMarginLoss()
>>> x = Tensor(np.array([[0.1, 0.2, 0.4, 0.8], [0.2, 0.3, 0.5, 0.7]]), mindspore.float32)
>>> target = Tensor(np.array([[1, 2, 0, 3], [2, 3, -1, 1]]), mindspore.int32)
>>> output = loss(x, target)
>>> print(output)
(Tensor(shape=[], dtype=Float32, value= 0.325), Tensor(shape=[2, 4], dtype=Int32, value=
[[1, 1, 1, 1], [0, 0, 1, 1]]))
```

mindspore.ops.MultiMarginLoss

class mindspore.ops.**MultiMarginLoss** (*p=1, margin=1.0, reduction='mean'*)

创建一个损失函数, 用于优化输入和输出之间的多分类合页损失。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多参考详见 `mindspore.ops.multi_margin_loss()`。

参数:

- **p** (int, 可选) - 对偶距离的范数度。必须为 1 或 2。默认值: 1。
- **margin** (int, 可选) - 改变对偶距离的参数。默认值: 1.0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的加权平均值。
 - "sum": 计算输出元素的总和。

输入:

- **inputs** (Tensor) - 输入, shape 为 (N, C) 。数据类型只支持 float32、float16 或 float64。
- **target** (Tensor) - 真实标签, shape 为 $(N,)$ 。数据类型只支持 int64。值应为非负值, 且小于 C 。
- **weight** (Tensor, 可选) - 每个类别的缩放权重, shape 为 $(C,)$ 。数据类型只支持 float32、float16 或 float64。

输出:

Tensor, 当 *reduction* 为 "none" 时, shape 为 $(N,)$ 。否则, 为标量。类型与 *inputs* 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.ones(shape=[3, 3]), mindspore.float32)
>>> target = Tensor(np.array([1, 2, 1]), mindspore.int64)
>>> weight = Tensor(np.array([1, 1, 1]), mindspore.float32)
>>> loss = ops.MultiMarginLoss()
>>> output = loss(x, target, weight)
>>> print(output)
0.6666667
```

mindspore.ops.NLLLoss

class mindspore.ops.NLLLoss (*reduction='mean', ignore_index=-100*)

获取预测值和目标值之间的负对数似然损失。

reduction = none 时, 负对数似然损失如下:

$$\ell(x, t) = L = \{l_1, \dots, l_N\}^T, \quad l_n = -w_{t_n} x_{n, t_n}, \quad w_c = \text{weight}[c] \cdot 1$$

其中, x 表示预测值, t 表示目标值, w 表示权重, N 表示 batch size, c 限定范围为 $[0, C-1]$, 表示类索引, 其中 C 表示类的数量。

reduction $\neq$ *none* 时 (默认为 "mean"), 则

$$\ell(x, t) = \begin{cases} \sum_{n=1}^N \frac{1}{\sum_{n=1}^N w_{t_n}} l_n, & \text{if reduction = 'mean'}; \\ \sum_{n=1}^N l_n, & \text{if reduction = 'sum'} \end{cases}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'mean'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的加权平均值。
 - "sum": 计算输出元素的总和。
- **ignore_index** (int, 可选) - 指定标签中需要忽略的值 (一般为填充值), 使其不对梯度产生影响。默认值: -100。

输入：

- **logits** (Tensor) - 输入预测值, shape 为 (N, C) 。数据类型仅支持 float32 或 float16 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。
- **labels** (Tensor) - 输入目标值, shape 为 $(N,)$, 取值范围为 $[0, C - 1]$ 。数据类型仅支持 uint8 或 int32 或 int64。
- **weight** (Tensor) - 指定各类别的权重。shape 为 $(C,)$ 。数据类型仅支持 float32 或 float16 或 bfloat16 (仅 Atlas A2 训练系列产品支持)。要求与 *logits* 的数据类型保持一致。

返回：

由 *loss* 和 *total_weight* 组成的 2 个 Tensor 的 tuple。

- **loss** (Tensor) - 当 *reduction* 为 "none" 且 *logits* 为二维 Tensor 时, *loss* 的 shape 为 $(N,)$, 否则 *loss* 为 scalar。*loss* 的数据类型与 *logits* 相同。
- **total_weight** (Tensor) - *total_weight* 是 scalar, 数据类型与 *weight* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> logits = Tensor(np.array([[0.5488135, 0.71518934],
...                           [0.60276335, 0.5448832],
...                           [0.4236548, 0.6458941]]).astype(np.float32))
>>> labels = Tensor(np.array([0, 0, 0]).astype(np.int32))
>>> weight = Tensor(np.array([0.3834415, 0.79172504]).astype(np.float32))
>>> nll_loss = ops.NLLLoss(reduction="mean")
>>> loss, weight = nll_loss(logits, labels, weight)
>>> print(loss)
-0.52507716
>>> print(weight)
1.1503246
```

mindspore.ops.RNNTLoss

class mindspore.ops.RNNTLoss (*blank_label=0*)

计算相对于 softmax 输出的 RNNTLoss 及其梯度。

参数：

- **blank_label** (int) - 空白标签。默认值：0。

输入：

- **acts** (Tensor) - Tensor, shape 为 (B, T, U, V) 。其中 B 为 batch 维, T 为输入序列长度, U 为目标序列长度, V 为输出维。数据类型必须为 float16 或 float32。
- **labels** (Tensor) - Tensor, shape 为 $(B, U - 1)$ 。数据类型为 int32。
- **input_lengths** (Tensor) - Tensor, shape 为 $(B,)$ 。数据类型为 int32。
- **label_lengths** (Tensor) - Tensor, shape 为 $(B,)$ 。数据类型为 int32。

输出：

- **costs** (Tensor) - Tensor, shape 为 $(B,)$ 。数据类型为 int32。

- **grads** (Tensor) - 具有与 *acts* 相同的 shape 和 dtype。

异常:

- **TypeError** - 如果 *acts*、*labels*、*input_lengths* 或 *label_lengths* 不是 Tensor。
- **TypeError** - 如果 *acts* 的 dtype 既不是 float16 也不是 float32。
- **TypeError** - 如果 *labels*、*input_lengths* 或 *label_lengths* 的 dtype 不是 int32。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> from mindspore import ops, Tensor
>>> B, T, U, V = 1, 2, 3, 5
>>> blank = 0
>>> acts = np.random.random((B, T, U, V)).astype(np.float32)
>>> labels = np.array([[1, 2]]).astype(np.int32)
>>> input_length = np.array([T] * B).astype(np.int32)
>>> label_length = np.array([len(l) for l in labels]).astype(np.int32)
>>> rnnt_loss = ops.RNNTLoss(blank_label=0)
>>> costs, grads = rnnt_loss(Tensor(acts), Tensor(labels), Tensor(input_length),
    ↳Tensor(label_length))
>>> print(costs.shape)
(1,)
>>> print(grads.shape)
(1, 2, 3, 5)
```

mindspore.ops.SigmoidCrossEntropyWithLogits

class mindspore.ops.SigmoidCrossEntropyWithLogits

计算预测值与真实值之间的 sigmoid 交叉熵。

首先计算相互独立的离散分类任务中的分布误差，再计算各个类的交叉熵损失。

将输入 *logits* 设置为 *X*，输入 *label* 为 *Y*，输出为 *loss*。然后，

$$p_{ij} = \text{sigmoid}(X_{ij}) = \frac{1}{1+e^{-x_{ij}}}$$

$$\text{loss}_{ij} = -[Y_{ij} * \ln(p_{ij}) + (1 - Y_{ij})\ln(1 - p_{ij})]$$

输入:

- **logits** (Tensor) - 预测值，任意维度的 Tensor，其 shape 为 $(N,*)$ ，其中 * 为任意数量的额外维度。数据类型必须为 float16 或 float32。
- **label** (Tensor) - 真实值，shape 和数据类型与 *logits* 的相同。

输出:

Tensor，shape 和数据类型与输入 *logits* 相同。

异常:

- **TypeError** - *logits* 或 *label* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> logits = Tensor(np.array([[ -0.8, 1.2, 0.7], [ -0.1, -0.4, 0.7]]).astype(np.float32))
>>> labels = Tensor(np.array([[ 0.3, 0.8, 1.2], [ -0.6, 0.1, 2.2]]).astype(np.float32))
>>> sigmoid = ops.SigmoidCrossEntropyWithLogits()
>>> output = sigmoid(logits, labels)
>>> print(output)
[[ 0.6111007  0.5032824  0.26318604]
 [ 0.58439666  0.5530153 -0.4368139 ]]
```

mindspore.ops.SmoothL1Loss

class mindspore.ops.SmoothL1Loss (beta=1.0, reduction='none')

计算平滑 L1 损失, 该 L1 损失函数有稳健性。

警告: 该 API 在 CPU 上性能较差, 推荐在 Ascend/GPU 上运行。

更多参考详见 `mindspore.ops.smooth_l1_loss()`。

参数:

- **beta** (number, 可选) - 控制损失函数在 L1 损失和 L2 损失间变换的阈值, 默认值: 1.0。
 - Ascend: 该值必须大于等于 0。
 - CPU/GPU: 该值必须大于 0。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式, 可选 'none'、'mean'、'sum', 默认值: 'none'。
 - "none": 不应用规约方法。
 - "mean": 计算输出元素的平均值。
 - "sum": 计算输出元素的总和。

输入:

- **logits** (Tensor) - 任意维度输入 Tensor。支持数据类型:
 - Ascend: float16、float32、bfloat16。
 - CPU/GPU: float16、float32、float64。
- **labels** (Tensor) - 真实值。
 - CPU/Ascend: 与 *logits* 的 shape 相同, *logits* 和 *labels* 遵循隐式类型转换规则, 使数据类型一致。
 - GPU: 与 *logits* 的 shape 和数据类型相同。

输出:

Tensor, 如果 *reduction* 为 'none', 则输出 Tensor 的 shape 与 *input* 的 shape 相同, 否则 shape 为 ()。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> loss = ops.SmoothL1Loss()
>>> logits = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> labels = Tensor(np.array([1, 2, 2]), mindspore.float32)
>>> output = loss(logits, labels)
>>> print(output)
[0.  0.  0.5]
```

mindspore.ops.SoftMarginLoss

class mindspore.ops.SoftMarginLoss (reduction='mean')

```
prim = ops.SoftMarginLoss(reduction)
out = prim(input, target)
```

等价于

```
ops.soft_margin_loss(input, target, reduction)
```

更多详情请查看：[mindspore.ops.soft_margin_loss\(\)](#)。

mindspore.ops.SoftmaxCrossEntropyWithLogits

class mindspore.ops.SoftmaxCrossEntropyWithLogits

使用 one-hot 编码获取预测值和真实之间的 softmax 交叉熵。

SoftmaxCrossEntropyWithLogits 算法的更新公式如下：

$$p_{ij} = \text{softmax}(X_{ij}) = \frac{\exp(x_i)}{\sum_{j=0}^{N-1} \exp(x_j)}$$

$$\text{loss}_{ij} = -\sum_j Y_{ij} * \ln(p_{ij})$$

其中 X 代表 *logits*。 Y 代表 *label*。 loss 代表 *output*。

输入：

- **logits** (Tensor) - 输入预测值，其 shape 为 (N, C) ，数据类型为 float16 或 float32。
- **labels** (Tensor) - 输入真实值，其 shape 为 (N, C) ，数据类型与 *logits* 的相同。

输出：

两个 Tensor(*loss*, *dlogits*) 组成的 tuple，*loss* 的 shape 为 $(N,)$ ，*dlogits* 的 shape 与 *logits* 的相同。

异常：

- **TypeError** - *logits* 或 *labels* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - *logits* 或 *labels* 不是 Tensor。
- **ValueError** - *logits* 的 shape 与 *labels* 的不同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> logits = Tensor([[2, 4, 1, 4, 5], [2, 1, 2, 4, 3]], mindspore.float32)
>>> labels = Tensor([[0, 0, 0, 0, 1], [0, 0, 0, 1, 0]], mindspore.float32)
>>> softmax_cross = ops.SoftmaxCrossEntropyWithLogits()
>>> loss, dlogits = softmax_cross(logits, labels)
>>> print(loss)
[0.5899297 0.52374405]
>>> print(dlogits)
[[ 0.02760027  0.20393994  0.01015357  0.20393994 -0.44563377]
 [ 0.08015892  0.02948882  0.08015892 -0.4077012  0.21789455]]
```

mindspore.ops.SparseSoftmaxCrossEntropyWithLogits

class mindspore.ops.SparseSoftmaxCrossEntropyWithLogits (*is_grad=False*)

计算预测值和标签之间的稀疏 softmax 交叉熵。

将预测值设置为 X ，输入标签设置为 Y ，输出设置为 $loss$ 。计算公式如下：

$$p_{ij} = \text{softmax}(X_{ij}) = \frac{\exp(x_i)}{\sum_{j=0}^{N-1} \exp(x_j)}$$

$$loss_{ij} = \begin{cases} -\ln(p_{ij}), & j = y_i \\ 0, & j \neq y_i \end{cases}$$

$$loss = \sum_{ij} loss_{ij}$$

参数：

- **is_grad** (bool, 可选) - 如果为 True，则返回计算的梯度。默认值：False。

输入：

- **logits** (Tensor) - 输入的预测值，其 shape 为 (N, C) 。数据类型必须为 float16 或 float32。
- **labels** (Tensor) - 输入的标签，其 shape 为 (N) 。数据类型必须为 int32 或 int64。

输出：

Tensor，如果 *is_grad* 为 False，则输出 Tensor 是损失值；如果 *is_grad* 为 True，则输出 Tensor 是输入的梯度，其 shape 与 *logits* 相同。

异常：

- **TypeError** - 如果 *is_grad* 不是 bool。
- **TypeError** - 如果 *logits* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *labels* 的数据类型既不是 int32 也不是 int64。
- **ValueError** - 如果 *logits.shape*[0] != *labels.shape*[0]。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> logits = Tensor([[2, 3, 1, 4, 5], [2, 1, 2, 4, 3]], mindspore.float32)
```

(下页继续)

(续上页)

```

>>> labels = Tensor([0, 1], mindspore.int32)
>>> sparse_softmax_cross = ops.SparseSoftmaxCrossEntropyWithLogits()
>>> loss = sparse_softmax_cross(logits, labels)
>>> print(loss)
3.4878292
>>> sparse_softmax_cross_grad = ops.SparseSoftmaxCrossEntropyWithLogits(is_grad=True)
>>> loss_grad = sparse_softmax_cross_grad(logits, labels)
>>> print(loss_grad)
[[-0.48415753  0.04306427  0.00582811  0.11706084  0.3182043 ]
 [ 0.04007946 -0.4852556  0.04007946  0.2961494  0.10894729]]

```

mindspore.ops.TripletMarginLoss

class mindspore.ops.TripletMarginLoss ($p=2$, $eps=1e-6$, $swap=False$, $reduction='mean'$)

三元组损失函数。

创建一个标准，用于计算输入 Tensor x_1 、 x_2 和 x_3 与大于 0 的 *margin* 之间的三元组损失值。可以用来测量样本之间的相似度。一个三元组包含 a 、 p 和 n （即分别代表 *anchor*、*positive examples* 和 *negative examples*）。所有输入 Tensor 的 *shape* 都应该为 (N, D) 。距离交换在 V. Balntas、E. Riba 等人的论文 [Learning local feature descriptors with triplets and shallow convolutional neural networks](#) 中有详细的阐述。

对于每个小批量样本，损失值为：

$$L(a, p, n) = \max\{d(a_i, p_i) - d(a_i, n_i) + \text{margin}, 0\}$$

其中

$$d(x_i, y_i) = \|x_i - y_i\|_p$$

参数：

- **p** (int, 可选) - 成对距离的范数。默认值：2。
- **eps** (float, 可选) - 默认值：1e-6。
- **swap** (bool, 可选) - 距离交换。默认值：False。
- **reduction** (str, 可选) - 指定应用于输出结果的规约计算方式，可选 'none'、'mean'、'sum'，默认值：'mean'。
 - "none"：不应用规约方法。
 - "mean"：计算输出元素的平均值。
 - "sum"：计算输出元素的总和。

输入：

- **x** (Tensor) - 从训练集随机选取的样本，数据类型为 BasicType。
- **positive** (Tensor) - 与 x 为同一类的样本，数据类型和 *shape* 与 x 一致。
- **negative** (Tensor) - 与 x 为异类的样本，数据类型和 *shape* 与 x 一致。
- **margin** (Tensor) - 拉近 a 和 p 之间的距离，拉远 a 和 n 之间的距离。

输出：

Tensor 或 Scalar，如果 *reduction* 为 "none"，将返回 Tensor，其 *shape* 为 (N) ，否则，将返回 Scalar。

异常：

- **TypeError** - *x*、*positive*、*negative* 或者 *margin* 不是 Tensor。
- **TypeError** - *x*、*positive* 或者 *negative* 的数据类型不是 BasicType。
- **TypeError** - *x*、*positive* 或者 *negative* 的数据类型不一致。
- **TypeError** - *margin* 的数据类型不是 float32。
- **TypeError** - *p* 的数据类型不是 int。
- **TypeError** - *eps* 的数据类型不是 float。
- **TypeError** - *swap* 的数据类型不是 bool。
- **ValueError** - *x*、*positive* 和 *negative* 的维度同时小于等于 1。
- **ValueError** - *x*、*positive* 或 *negative* 的维度大于等于 8。
- **ValueError** - *margin* 的 shape 长度不为 0。
- **ValueError** - *x*、*positive* 和 *negative* 三者之间的 shape 无法广播。
- **ValueError** - *reduction* 不为"mean"、"sum" 或"none"。

支持平台：

GPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> loss = ops.TripletMarginLoss()
>>> x = Tensor(np.array([[0.3, 0.7], [0.5, 0.5]]), mindspore.float32)
>>> positive = Tensor(np.array([[0.4, 0.6], [0.4, 0.6]]), mindspore.float32)
>>> negative = Tensor(np.array([[0.2, 0.9], [0.3, 0.7]]), mindspore.float32)
>>> margin = Tensor(1.0, mindspore.float32)
>>> output = loss(x, positive, negative, margin)
>>> print(output)
0.8881968
```

20.2.3 激活函数

接口名	概述	支持平台
<code>mindspore.ops.CeLU</code>	更多详情请查看: <code>mindspore.ops.celu()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Elu</code>	更多详情请查看: <code>mindspore.ops.elu()</code> 。	Ascend GPU CPU
<code>mindspore.ops.FastGeLU</code>	更多详情请查看: <code>mindspore.ops.fast_gelu()</code> 。	Ascend GPU CPU
<code>mindspore.ops.GeLU</code>	高斯误差线性单元激活函数 (Gaussian Error Linear Units activation function)。	Ascend GPU CPU
<code>mindspore.ops.GLU</code>	计算输入 Tensor 的门线性单元激活函数 (Gated Linear Unit activation function) 值。	Ascend CPU
<code>mindspore.ops.HShrink</code>	更多详情请查看: <code>mindspore.ops.hardshrink()</code> 。	Ascend GPU CPU
<code>mindspore.ops.HSigmoid</code>	更多详情请查看: <code>mindspore.ops.hardsigmoid()</code> 。	Ascend GPU CPU

下页继续

表 4 - 续上页

<code>mindspore.ops.HSwish</code>	更多详情请查看: <code>mindspore.ops.hardswish()</code> 。	Ascend GPU CPU
<code>mindspore.ops.LogSoftmax</code>	更多详情请查看: <code>mindspore.ops.log_softmax()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Mish</code>	逐元素计算输入 Tensor 的 MISH (Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。	Ascend GPU CPU
<code>mindspore.ops.PReLU</code>	更多详情请查看: <code>mindspore.ops.prelu()</code> 。	Ascend GPU CPU
<code>mindspore.ops.ReLU</code>	更多详情请查看: <code>mindspore.ops.relu()</code> 。	Ascend GPU CPU
<code>mindspore.ops.ReLU6</code>	更多详情请查看: <code>mindspore.ops.relu6()</code> 。	Ascend GPU CPU
<code>mindspore.ops.SeLU</code>	激活函数 SeLU (Scaled exponential Linear Unit)。	Ascend GPU CPU
<code>mindspore.ops.Sigmoid</code>	更多详情请查看: <code>mindspore.ops.sigmoid()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Softmax</code>	在指定轴上使用 Softmax 函数做归一化操作。	Ascend GPU CPU
<code>mindspore.ops.Softplus</code>	Softplus 激活函数。	Ascend GPU CPU
<code>mindspore.ops.SoftShrink</code>	更多详情请查看: <code>mindspore.ops.softshrink()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Softsign</code>	Softsign 激活函数。	Ascend GPU CPU
<code>mindspore.ops.Tanh</code>	更多详情请查看: <code>mindspore.ops.tanh()</code> 。	Ascend GPU CPU

mindspore.ops.CeLU

`class mindspore.ops.CeLU(alpha=1.0)`

```
prim = ops.CeLU(alpha)
out = prim(x)
```

等价于

```
ops.celu(x, alpha)
```

更多详情请查看: `mindspore.ops.celu()`。

mindspore.ops.Elu

`class mindspore.ops.Elu(alpha=1.0)`

```
prim = ops.Elu(alpha)
out = prim(input_x)
```

等价于

```
ops.elu(input_x, alpha)
```

更多详情请查看: `mindspore.ops.elu()`。

mindspore.ops.FastGeLU

class mindspore.ops.FastGeLU

```
prim = ops.FastGeLU()
out = prim(x)
```

等价于

```
ops.fast_gelu(x)
```

更多详情请查看: [mindspore.ops.fast_gelu\(\)](#)。

mindspore.ops.GeLU

class mindspore.ops.GeLU

高斯误差线性单元激活函数 (Gaussian Error Linear Units activation function)。

在 [Gaussian Error Linear Units \(GELUs\)](#) 文章中对 GeLU 函数进行了介绍。此外, 也可以参考 [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding](#)。

GeLU 函数定义如下:

$$GELU(x_i) = x_i * P(X < x_i)$$

其中 P 是标准高斯分布的累积分布函数, x_i 是输入的元素。

说明: 在计算 gelu 的输入梯度时, 当输入为 inf, Ascend 与 GPU 在反向传播输出之间存在差异。当输入 x 为 -inf 时, Ascend 的计算结果为 0, GPU 的计算结果为 nan。当输入 x 为 inf 时, Ascend 的计算结果为梯度 dy , GPU 的计算结果为 nan。数学意义上, Ascend 的计算结果精度更高。

输入:

- **x** (Tensor) - 激活函数 GeLU 的输入, 数据类型为 float16、float32 或 float64。

输出:

Tensor, 数据类型和 shape 与 x 的相同。

异常:

- **TypeError** - x 不是 Tensor。
- **TypeError** - x 的数据类型不是 float16、float32 或 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> result = ops.GeLU()(x)
>>> print(result)
[0.841192  1.9545976  2.9963627]
```


mindspore.ops.GLU

class mindspore.ops.GLU (axis=-1)

计算输入 Tensor 的门线性单元激活函数 (Gated Linear Unit activation function) 值。

$$GLU(a, b) = a \otimes \sigma(b)$$

其中, a 表示输入 Tensor x 拆分后的前半元素, b 表示 x 拆分后的另一半元素。这里 σ 为 sigmoid 函数, $\otimes$ 是 Hadamard 乘积。请参考 [Language Modeling with Gated Convluational Networks](#)。

参数:

- **axis** (int, 可选) - 指定分割输入 x 的轴, 取值范围 $[-r, r)$, 其中 r 为 x 的维度数。默认值: -1, 输入 x 的最后一维。

输入:

- **x** (Tensor) - 待计算的 Tensor, 数据类型为浮点型, shape 为 $(*_1, N, *_2)$, 其中 $*$ 为任意额外维度, 且要求 N 为偶数。 N 为 x 在被 $axis$ 选中的维度上的大小。

输出:

Tensor, 数据类型与输入 x 相同, shape 为 $(*_1, M, *_2)$, 其中 $M = N/2$ 。

异常:

- **TypeError** - x 不是 Tensor 或 $axis$ 不是整型。
- **IndexError** - $axis$ 的值超出了 $[-r, r)$ 的范围, 其中 r 为 x 的维度数。
- **RuntimeError** - x 数据类型不被支持。
- **RuntimeError** - x 在被 $axis$ 选中的维度上的长度不为偶数。

支持平台:

Ascend CPU

样例:

```
>>> from mindspore import ops, Tensor
>>> from mindspore import dtype as mstype
>>> import numpy as np
>>> axis = 0
>>> x = Tensor(np.array([0.3220, 0.9545, 0.7879, 0.0975, 0.3698,
...                      0.5135, 0.5740, 0.3435, 0.1895, 0.8764,
...                      0.4980, 0.9673, 0.9879, 0.6988, 0.9022,
...                      0.9304, 0.1558, 0.0153, 0.1559, 0.9852]).reshape([2, 2, 5]),
↳mstype.float32)
>>> glu = ops.GLU(axis=axis)
>>> y = glu(x)
>>> print(y)
[[[0.20028052 0.6916126 0.57412136 0.06512236 0.26307625]
 [0.3682598 0.3093122 0.17306386 0.10212085 0.63814086]]]
```

mindspore.ops.HShrink

class mindspore.ops.HShrink (lambd=0.5)

```
prim = ops.HShrink(lambd)
out = prim(input)
```

等价于

```
ops.hardshrink(input, lambd)
```

更多详情请查看: [*mindspore.ops.hardshrink\(\)*](#) 。

mindspore.ops.HSigmoid

class mindspore.ops.HSigmoid

```
prim = ops.HSigmoid()
out = prim(input)
```

等价于

```
ops.hardsigmoid(input)
```

更多详情请查看: [*mindspore.ops.hardsigmoid\(\)*](#) 。

mindspore.ops.HSwish

class mindspore.ops.HSwish

```
prim = ops.HSwish()
out = prim(input)
```

等价于

```
ops.hardswish(input)
```

更多详情请查看: [*mindspore.ops.hardswish\(\)*](#) 。

mindspore.ops.LogSoftmax

class mindspore.ops.LogSoftmax (axis=- 1)

```
prim = ops.LogSoftmax(axis)
out = prim(logits)
```

等价于

```
ops.log_softmax(logits, axis)
```

更多详情请查看: [*mindspore.ops.log_softmax\(\)*](#) 。

mindspore.ops.Mish

class mindspore.ops.Mish

逐元素计算输入 Tensor 的 MISH (Self Regularized Non-Monotonic Neural Activation Function 自正则化非单调神经激活函数)。

详情请查看[`mindspore.ops.mish\(\)`](#)。

输入：

- **x** (Tensor) - 输入 Tensor。支持数据类型：
 - GPU/CPU: float16、float32、float64。
 - Ascend: float16、float32。

输出：

Tensor，与 *x* 的 shape 和数据类型相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[ -1.0,  4.0, -8.0], [ 2.0, -5.0,  9.0]]), mindspore.float32)
>>> mish = ops.Mish()
>>> output = mish(x)
>>> print(output.shape)
(2, 3)
>>> x = Tensor(2.1, mindspore.float32)
>>> output = mish(x)
>>> print(output)
2.050599
```

mindspore.ops.PReLU

class mindspore.ops.PReLU

```
prim = ops.PReLU()
out = prim(input, weight)
```

等价于

```
ops.prelu(input, weight)
```

更多详情请查看：[`mindspore.ops.prelu\(\)`](#)。

mindspore.ops.ReLU

class mindspore.ops.ReLU

```
prim = ops.ReLU()
out = prim(input)
```

等价于

```
ops.relu(input)
```

更多详情请查看: [mindspore.ops.relu\(\)](#)。

mindspore.ops.ReLU6

class mindspore.ops.ReLU6

```
prim = ops.ReLU6()
out = prim(x)
```

等价于

```
ops.relu6(x)
```

更多详情请查看: [mindspore.ops.relu6\(\)](#)。

mindspore.ops.SELU

class mindspore.ops.SELU

激活函数 SELU (Scaled exponential Linear Unit)。

该激活函数定义为:

$$E_i = scale * \begin{cases} x_i, & \text{if } x_i \geq 0; \\ \alpha * (\exp(x_i) - 1), & \text{otherwise.} \end{cases}$$

其中, α 和 $scale$ 是预定义的常量 ($\alpha = 1.67326324$, $scale = 1.05070098$)。

更多详细信息, 请参见 [Self-Normalizing Neural Networks](#)。

输入:

- **input_x** (Tensor) - 任意维度的 Tensor, 数据类型为 int8、int32、float16、float32、float64 (仅 CPU、GPU)。

输出:

Tensor, 数据类型和 shape 与 *input_x* 的相同。

异常:

- **TypeError** - *input_x* 的数据类型不是 int8、int32、float16、float32、float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[-1.0, 4.0, -8.0], [2.0, -5.0, 9.0]]), mindspore.float32)
>>> selu = ops.SELU()
>>> output = selu(input_x)
>>> print(output)
[[-1.1113307  4.202804 -1.7575096]
 [ 2.101402 -1.7462534  9.456309 ]]
```

mindspore.ops.Sigmoid

class mindspore.ops.Sigmoid

```
prim = ops.Sigmoid()
out = prim(input)
```

等价于

```
ops.sigmoid(input)
```

更多详情请查看: [mindspore.ops.sigmoid\(\)](#)。

mindspore.ops.Softmax

class mindspore.ops.Softmax(*axis=-1*)

在指定轴上使用 Softmax 函数做归一化操作。

更多参考详见[mindspore.ops.softmax\(\)](#)。

参数:

- **axis** (Union[int, tuple], 可选) - 指定 Softmax 操作的轴。默认值: -1。

输入:

- **input** (Tensor) - shape: (N,*), 其中 * 表示任意数量的附加维度。

输出:

Tensor, 数据类型和 shape 与 *input* 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> softmax = ops.Softmax()
>>> output = softmax(input)
>>> print(output)
[0.01165623 0.03168492 0.08612854 0.23412167 0.6364086 ]
```

mindspore.ops.Softplus

class mindspore.ops.Softplus

Softplus 激活函数。

Softplus 为 ReLU 函数的平滑近似。可对一组数值使用来确保转换后输出结果均为正值。函数计算如下：

$$\text{output} = \log(1 + \exp(x))$$

输入：

- **input_x** (Tensor) - 任意维度的输入 Tensor。支持数据类型：
 - GPU/CPU: float16、float32、float64。
 - Ascend: float16、float32。

输出：

Tensor，与 *input_x* 的 shape 和数据类型相同。

异常：

- **TypeError** - *input_x* 不是 Tensor。
- **TypeError** - *input_x* 的数据类型非 float16、float32 或 float64。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([1, 2, 3, 4, 5]), mindspore.float32)
>>> softplus = ops.Softplus()
>>> output = softplus(input_x)
>>> print(output)
[1.3132615 2.126928 3.0485873 4.01815 5.0067153]
```

mindspore.ops.SoftShrink

class mindspore.ops.SoftShrink (*lambd=0.5*)

```
prim = ops.SoftShrink(lambd)
out = prim(input)
```

等价于

```
ops.softshrink(input, lambd)
```

更多详情请查看：[mindspore.ops.softshrink\(\)](#)。

mindspore.ops.Softsign

class mindspore.ops.Softsign

Softsign 激活函数。

更多细节请参考 `mindspore.ops.softsign()` 。

输入：

- **input_x** (Tensor) - shape 为 (N,*) 的 Tensor，其中 * 表示任意个数的维度。它的数据类型必须为 float16 或 float32。

输出：

Tensor，数据类型和 shape 与 *input_x* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([0, -1, 2, 30, -30]), mindspore.float32)
>>> softsign = ops.Softsign()
>>> output = softsign(input_x)
>>> print(output)
[ 0.          -0.5          0.6666667   0.9677419  -0.9677419]
```

mindspore.ops.Tanh

class mindspore.ops.Tanh

```
prim = ops.Tanh()
out = prim(input)
```

等价于

```
ops.tanh(input)
```

更多详情请查看：`mindspore.ops.tanh()` 。

20.2.4 优化器

接口名	概述	支持平台
<code>mindspore.ops.Adam</code>	通过 Adam 算法更新梯度。	Ascend GPU CPU
<code>mindspore.ops.AdamWeightDecay</code>	通过具有权重衰减的自适应矩估计算法 (AdamWeightDecay) 更新梯度。	Ascend GPU CPU
<code>mindspore.ops.AdaptiveAvgPool2D</code>	二维自适应平均池化。	Ascend GPU CPU
<code>mindspore.ops.AdaptiveAvgPool3D</code>	三维自适应平均池化。	Ascend GPU CPU
<code>mindspore.ops.ApplyAdadelta</code>	根据 Adadelta 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops.ApplyAdagrad</code>	根据 Adagrad 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops.ApplyAdagradDA</code>	根据 Adagrad 算法更新 var。	Ascend GPU CPU

下页继续

表 5 – 续上页

<code>mindspore.ops.ApplyAdagradV2</code>	根据 Adagrad 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops.ApplyAdaMax</code>	根据 AdaMax 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops. ApplyAdamWithAmsgradV2</code>	根据 Adam 算法更新变量 var。	Ascend GPU CPU
<code>mindspore.ops.ApplyAddSign</code>	根据 AddSign 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops. ApplyCenteredRMSProp</code>	居中 RMSProp 算法优化器。	Ascend GPU CPU
<code>mindspore.ops.ApplyFtrl</code>	根据 FTRL 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops. ApplyGradientDescent</code>	通过从 var 中减去 $\alpha * \text{delta}$ 来更新 var。	Ascend GPU CPU
<code>mindspore.ops.ApplyMomentum</code>	使用动量算法的优化器。	Ascend GPU CPU
<code>mindspore.ops.ApplyPowerSign</code>	根据 AddSign 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops. ApplyProximalAdagrad</code>	根据 Proximal Adagrad 算法更新网络参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops. ApplyProximalGradientDescent</code>	根据 FOBOS(Forward Backward Splitting) 算法更新网络参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops.ApplyRMSProp</code>	实现均方根传播 Root Mean Square prop(RMSProp) 算法的优化器。	Ascend GPU CPU
<code>mindspore.ops.LARSUpdate</code>	对梯度的平方和应用 LARS(layer-wise adaptive rate scaling) 算法。	Ascend
<code>mindspore.ops. SparseApplyAdagradV2</code>	根据 Adagrad 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops. SparseApplyProximalAdagrad</code>	根据 Proximal Adagrad 算法更新网络参数或者 Tensor。	Ascend GPU
<code>mindspore.ops.SGD</code>	计算随机梯度下降。	Ascend GPU CPU
<code>mindspore.ops.SparseApplyFtrl</code>	根据 FTRL-proximal 算法更新相关参数或者 Tensor。	Ascend GPU CPU
<code>mindspore.ops.SparseApplyFtrlV2</code>	<code>mindspore.ops.SparseApplyFtrlV2</code> 从 2.0 版本开始已被弃用, 并将在未来版本中被移除, 建议使用 <code>mindspore.ops.SparseApplyFtrl</code> 代替。	弃用

mindspore.ops.Adam

class mindspore.ops.Adam(*use_locking=False, use_nesterov=False*)

通过 Adam 算法更新梯度。

Adam 算法详情请参考 Adam: A Method for Stochastic Optimization。

有关更多详细信息, 参见 `mindspore.nn.Adam`。

更新公式如下:

$$\begin{aligned}
 m &= \beta_1 * m + (1 - \beta_1) * g \\
 v &= \beta_2 * v + (1 - \beta_2) * g * g \\
 l &= \alpha * \frac{\sqrt{1 - \beta_2^t}}{1 - \beta_1^t} \\
 w &= w - l * \frac{m}{\sqrt{v} + \epsilon}
 \end{aligned}$$

其中, m 表示第一个动量矩阵, v 表示第二个动量矩阵, g 表示 *gradient*, l 表示缩放因子 *lr*, β_1, β_2 表示 *beta1* 和 *beta2*, t 表示更新步数, $\beta_1^t(\beta_1^t)$ 和 $\beta_2^t(\beta_2^t)$ 表示 *beta1_power* 和 *beta2_power*, α 表示 *learning_rate*, w 表示 *var*, ϵ 表示 *epsilon*。

var、*m*、*v* 和 *gradient* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则低精度数据类型将转换为相对最高精度的数据类型。

参数：

- **use_locking** (bool) - 是否对参数更新加锁保护。如果为 `True`，则 *w*、*m* 和 *v* 的 `Tensor` 更新将受到锁的保护。如果为 `False`，则结果不可预测。默认值：`False`。
- **use_nesterov** (bool) - 是否使用 Nesterov Accelerated Gradient (NAG) 算法更新梯度。如果为 `True`，则使用 NAG 更新梯度。如果为 `False`，则在不使用 NAG 的情况下更新梯度。默认值：`False`。

输入：

- **var** (Union[Parameter, Tensor]) - 需更新的权重。shape: $(N, *)$ ，其中 $*$ 表示任意数量的附加维度，其数据类型可以是 `float16` 或 `float32`。
- **m** (Union[Parameter, Tensor]) - 更新公式中的第一个动量矩阵，shape 应与 *var* 相同。
- **v** (Union[Parameter, Tensor]) - 更新公式中的第二个动量矩阵，shape 应与 *var* 相同。
- **beta1_power** (float) - 在更新公式中的 β_1^t 。
- **beta2_power** (float) - 在更新公式中的 β_2^t 。
- **lr** (float) - 在更新公式中的 l 。其论文建议取值为 10^{-8} 。
- **beta1** (float) - 第一个动量矩阵的指数衰减率。论文建议取值为 0.9。
- **beta2** (float) - 第二个动量矩阵的指数衰减率。论文建议取值为 0.999。
- **epsilon** (float) - 添加到分母中的值，以提高数值稳定性。
- **gradient** (Tensor) - 表示梯度，shape 和数据类型与 *var* 相同。

输出：

3 个 `Tensor` 的 tuple，已更新的参数。

- **var** (Tensor) - shape 和数据类型与输入 *var* 相同。
- **m** (Tensor) - shape 和数据类型与输入 *m* 相同。
- **v** (Tensor) - shape 和数据类型与输入 *v* 相同的。

异常：

- **TypeError** - *use_locking* 和 *use_nesterov* 都不是 `bool`。
- **TypeError** - *var*、*m* 或 *v* 不是 `Parameter`。
- **TypeError** - *beta1_power*、*beta2_power*、*lr*、*beta1*、*beta2*、*epsilon* 或 *gradient* 不是 `Tensor`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops
>>> from mindspore import Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_adam = ops.Adam()
...         self.var = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="var")
```

(下页继续)

(续上页)

```

...     self.m = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="m")
...     self.v = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="v")
...     def construct(self, beta1_power, beta2_power, lr, beta1, beta2, epsilon, grad):
...         out = self.apply_adam(self.var, self.m, self.v, beta1_power, beta2_power, lr,
...         beta1, beta2,
...                                     epsilon, grad)
...         return out
...
>>> net = Net()
>>> gradient = Tensor(np.ones([2, 2]).astype(np.float32))
>>> output = net(0.9, 0.999, 0.001, 0.9, 0.999, 1e-8, gradient)
>>> print(net.var.asnumpy())
[[0.9996838 0.9996838]
 [0.9996838 0.9996838]]

```

mindspore.ops.AdamWeightDecay

class mindspore.ops.AdamWeightDecay (*use_locking=False*)

通过具有权重衰减的自适应矩估计算法 (AdamWeightDecay) 更新梯度。Adam 算法在 [Adam: 随机优化方法](#) 中提出。

AdamWeightDecay 是 Adam 算法的变体, 在 [解耦权重衰减正则化](#) 中提出的。

更新公式如下:

$$\begin{aligned}
 m &= \beta_1 * m + (1 - \beta_1) * g \\
 v &= \beta_2 * v + (1 - \beta_2) * g * g \\
 update &= \frac{m}{\sqrt{v + \epsilon}} \\
 update &= \begin{cases} update + weight\_decay * w & \text{if } weight\_decay > 0 \\ update & \text{otherwise} \end{cases} \\
 w &= w - lr * update
 \end{aligned}$$

m 代表第一个矩向量, v 代表第二个矩向量, g 代表 *gradient*, β_1, β_2 代表 *beta1* 和 *beta2*, lr 代表 *learning_rate*, w 代表 *var*, $decay$ 代表 *weight_decay*, ϵ 代表 *epsilon*。

参数:

- **use_locking** (bool) - 是否对参数更新加锁保护。如果为 `True`, 则 *var*、 m 和 v 张量的更新将受到锁的保护。如果为 `False`, 则结果不可预测。默认值: `False`。

输入:

- **var** (Union[Parameter, Tensor]) - 需要更新的权重。shape 为 $(N, *)$ 其中 $*$ 表示任何数量的附加维度, 数据类型可以是 `float16` 或 `float32`。
- **m** (Union[Parameter, Tensor]) - 更新公式中的第一个动量矩阵, 它的 shape 应该和 *var* 一致, 数据类型可以是 `float16` 或 `float32`。
- **v** (Union[Parameter, Tensor]) - 更新公式中的第二个动量矩阵, shape 与 m 相同。
- **lr** (float) - 更新公式中的 lr 。其论文建议值为 10^{-8} , 数据类型应为 `float32`。
- **beta1** (float) - 第一个动量矩阵的指数衰减率, 数据类型应为 `float32`。论文建议的值是 0.9。
- **beta2** (float) - 第二个动量矩阵的指数衰减率, 数据类型应为 `float32`。论文建议的值是 0.999。
- **epsilon** (float) - 添加到分母中的值, 以提高数值稳定性, 数据类型应为 `float32`。
- **decay** (float) - 权重衰减值, 必须是具有 `float32` 数据类型的标量张量。默认值: 0.0。

- **gradient** (Tensor) - 梯度, shape 与 *var* 相同。

输出:

3 个张量的 Tuple, 为更新后的参数。

- **var** (Tensor) - 具有与 *var* 相同的 shape 和数据类型。
- **m** (Tensor) - 具有与 *m* 相同的 shape 和数据类型。
- **v** (Tensor) - 具有与 *v* 相同的 shape 和数据类型。

异常:

- **TypeError** - 如果 *use_locking* 不是 bool 类型。
- **TypeError** - 如果 *lr*、*beta1*、*beta2*、*epsilon* 或者 *decay* 不是 float32。
- **TypeError** - 如果 *var*、*m* 或者 *v* 数据类型不是 float16 或者 float32。
- **TypeError** - 如果 *gradient* 不是 Tensor。
- **ValueError** - 如果 *epsilon* 小于等于 0。
- **ValueError** - 如果 *beta1*、*beta2* 不在 (0.0,1.0) 范围内。
- **ValueError** - 如果 *decay* 小于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import Tensor, Parameter, ops
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.adam_weight_decay = ops.AdamWeightDecay()
...         self.var = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="var")
...         self.m = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="m")
...         self.v = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="v")
...     def construct(self, lr, beta1, beta2, epsilon, decay, grad):
...         out = self.adam_weight_decay(self.var, self.m, self.v, lr, beta1, beta2,
...                                     epsilon, decay, grad)
...         return out
>>> net = Net()
>>> gradient = Tensor(np.ones([2, 2]).astype(np.float32))
>>> output = net(0.001, 0.9, 0.999, 1e-8, 0.0, gradient)
>>> print(net.var.asnumpy())
[[0.999 0.999]
 [0.999 0.999]]
```

mindspore.ops.AdaptiveAvgPool2D**class** mindspore.ops.AdaptiveAvgPool2D (*output_size*)

二维自适应平均池化。

更多参考详见[`mindspore.ops.adaptive\_avg\_pool2d\(\)`](#)。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **output_size** (Union[int, tuple]) - 输出特征图的 size。*output_size* 可以是二元 tuple，表示 (*H*, *W*)；也可以是单个 int，表示 (*H*, *H*)。*H* 和 *W* 可以是 int 或 None，如果是 None，则意味着输出的 size 与输入相同。

输入：

- **input_x** (Tensor) - AdaptiveAvgPool2D 的输入，为三维或四维的 Tensor，数据类型为 float16、float32 或者 float64。

输出：

Tensor，数据类型与 *input_x* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1: output_size=(None, 2)
>>> input_x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                             [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]],
...                             [[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]])),
mindspore.float32)
>>> adaptive_avg_pool_2d = ops.AdaptiveAvgPool2D((None, 2))
>>> output = adaptive_avg_pool_2d(input_x)
>>> print(output)
[[1.5 2.5]
 [4.5 5.5]
 [7.5 8.5]]
[[1.5 2.5]
 [4.5 5.5]
 [7.5 8.5]]
[[1.5 2.5]
 [4.5 5.5]
 [7.5 8.5]]]
>>> # case 2: output_size=2
>>> adaptive_avg_pool_2d = ops.AdaptiveAvgPool2D(2)
>>> output = adaptive_avg_pool_2d(input_x)
>>> print(output)
[[3. 4.]
 [6. 7.]]
[[3. 4.]
 [6. 7.]]
[[3. 4.]
 [6. 7.]]]
```

(下页继续)

(续上页)

```

[6. 7.]]]
>>> # case 3: output_size=(1, 2)
>>> adaptive_avg_pool_2d = ops.AdaptiveAvgPool2D((1, 2))
>>> output = adaptive_avg_pool_2d(input_x)
>>> print(output)
[[[4.5 5.5]]
 [[4.5 5.5]]
 [[4.5 5.5]]]

```

mindspore.ops.AdaptiveAvgPool3D

class mindspore.ops.**AdaptiveAvgPool3D** (*output_size*)
 三维自适应平均池化。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.adaptive_avg_pool3d()`。

参数：

- **output_size** (Union[int, tuple]) - 指定输出特征图的尺寸，可以为一个整数或者三元 tuple。

输入：

- **x** (Tensor) - AdaptiveAvgPool3D 的输入，为五维或四位的 Tensor。

输出：

Tensor，数据类型与 *x* 相同。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import nn, Tensor
>>> from mindspore.ops import AdaptiveAvgPool3D
>>> class AdaptiveAvgPool3DNet(nn.Cell):
...     def __init__(self, output_size):
...         super(AdaptiveAvgPool3DNet, self).__init__()
...         self.output_size_ = output_size
...         self.adaptive_avg_pool_3d = AdaptiveAvgPool3D(self.output_size_)
...     def construct(self, x_):
...         return self.adaptive_avg_pool_3d(x_)
...
>>> output_size=(1,1,1)
>>> input_x_val = np.zeros((1,1,2,2,2))
>>> input_x_val[:, :, 0, :, :] += 1
>>> input_x = Tensor(input_x_val, mindspore.float32)
>>> adaptive_avg_pool_3d = AdaptiveAvgPool3DNet(output_size)
>>> output = adaptive_avg_pool_3d(input_x)
>>> print(output)
[[[[[0.5]]]]]

```

mindspore.ops.ApplyAdadelta**class mindspore.ops.ApplyAdadelta**

根据 Adadelta 算法更新相关参数或者 Tensor。

Adadelta 算法，具体细节可参考论文 [ADADELTA: AN ADAPTIVE LEARNING RATE METHOD](#)。

$$\begin{aligned} \text{accum} &= \rho * \text{accum} + (1 - \rho) * \text{grad}^2 \\ \text{update} &= \sqrt{\text{accum\_update} + \epsilon} * \frac{\text{grad}}{\sqrt{\text{accum} + \epsilon}} \\ \text{accum\_update} &= \rho * \text{accum\_update} + (1 - \rho) * \text{update}^2 \\ \text{var} &= \text{var} - \text{lr} * \text{update} \end{aligned}$$

其中 ρ 代表 *rho*， ϵ 代表 *epsilon*。

var、*accum*、*accum_update* 和 *grad* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则较低精度的数据类型将转换为相对最高精度的数据类型。

输入：

- **var** (Union[Parameter, Tensor]) - 待更新的公式参数或者 Tensor *var*。数据类型为 float32 或 float16。shape: (N,*)，其中 * 表示任意数量的附加维度。
- **accum** (Union[Parameter, Tensor]) - 待更新的公式参数或者 Tensor *accum*，shape 和数据类型与 *var* 相同。
- **accum_update** (Union[Parameter, Tensor]) - 待更新的公式参数或者 Tensor *accum_update*，shape 和数据类型与 *var* 相同。
- **lr** (Union[Number, Tensor]) - 学习率，必须是 Scalar。数据类型为 float32 或 float16。
- **rho** (Union[Number, Tensor]) - 衰减率，必须是 Scalar。数据类型为 float32 或 float16。
- **epsilon** (Union[Number, Tensor]) - 加在分母上的值，以确保数值稳定，必须是 Scalar。数据类型为 float32 或 float16。
- **grad** (Tensor) - 梯度，shape 和数据类型与 *var* 相同。

输出：

3 个 Tensor 的元组，更新后的数据。

- **var** (Tensor) - shape 和数据类型与 *var* 相同。
- **accum** (Tensor) - shape 和数据类型与 *accum* 相同。
- **accum_update** (Tensor) - shape 和数据类型与 *accum_update* 相同。

异常：

- **TypeError** - 如果 *var*、*accum*、*accum_update*、*lr*、*rho*、*epsilon* 或 *grad* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *accum_update*、*lr*、*rho* 或 *epsilon* 既不是数值型也不是 Tensor。
- **TypeError** - 如果 *var*、*accum*、*accum_update* 和 *grad* 不支持数据类型转换。

支持平台：

Ascend GPU CPU

样例：

```

>>> import numpy as np
>>> import mindspore
>>> from mindspore import nn, Tensor, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_adadelta = ops.ApplyAdadelta()
...         self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                                [0.1, 0.5]]).astype(np.float32)), name="var")
...         self.accum = Parameter(Tensor(np.array([[0.6, 0.5],
...                                                [0.2, 0.6]]).astype(np.float32)), name="accum")
...         self.accum_update = Parameter(Tensor(np.array([[0.9, 0.1],
...                                                [0.7, 0.8]]).astype(np.float32)), name="accum_update")
...     def construct(self, lr, rho, epsilon, grad):
...         out = self.apply_adadelta(self.var, self.accum, self.accum_update, lr, rho, epsilon, grad)
...         return out
>>> net = Net()
>>> lr = Tensor(0.001, mindspore.float32)
>>> rho = Tensor(0.0, mindspore.float32)
>>> epsilon = Tensor(1e-6, mindspore.float32)
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(lr, rho, epsilon, grad)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.99051356e-01,  3.99683774e-01],
 [ 9.91633832e-02,  4.99105573e-01]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 9.00000036e-02,  4.89999980e-01],
 [ 1.00000007e-02,  6.40000045e-01]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 8.99990857e-01,  1.00000791e-01],
 [ 6.99930906e-01,  7.99999774e-01]]))

```

mindspore.ops.ApplyAdagrad

class mindspore.ops.ApplyAdagrad (*update_slots=True*)

根据 Adagrad 算法更新相关参数或者 Tensor。

Adagrad 算法在论文 [Adaptive Subgradient Methods for Online Learning and Stochastic Optimization](#) 中提出。针对不同参数样本数不均匀的问题，自适应的为各个参数分配不同的学习率。

$$\begin{aligned} accum &+= grad * grad \\ var &= lr * grad * \frac{1}{\sqrt{accum}} \end{aligned}$$

var、*accum* 和 *grad* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，较低精度的数据类型将转换为相对最高精度的数据类型。

参数：

- **update_slots** (bool) - 是否更新 *accum* 参数，如果为 True，*accum* 将更新。默认值为：True。

输入：

- **var** (Union[Parameter, Tensor]) - 要更新的权重或者 Tensor。数据类型为 float32 或 float16。shape: $(N, *)$ ，其中 $*$ 表示任意数量的附加维度。
- **accum** (Union[Parameter, Tensor]) - 要更新的累积。shape 必须与 *var* 相同。
- **lr** (Union[Number, Tensor]) - 学习率，必须是 Scalar。数据类型为 float32 或 float16。
- **grad** (Tensor) - 梯度，为一个 Tensor。shape 必须与 *var* 相同。

输出：

2 个 Tensor 组成的 tuple，更新后的数据。

- **var** (Tensor) - shape 和数据类型与 *var* 相同。
- **accum** (Tensor) - shape 和数据类型与 *accum* 相同。

异常：

- **TypeError** - 如果 *var*、*accum*、*lr* 或 *grad* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *lr* 既不是数值型也不是 Tensor。
- **TypeError** - 如果 *var*、*accum* 和 *grad* 不支持数据类型转换。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_adagrad = ops.ApplyAdagrad()
...         self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                                [0.1, 0.5]]).astype(np.float32)), name="var")
...         self.accum = Parameter(Tensor(np.array([[0.6, 0.5],
...                                                [0.2, 0.6]]).astype(np.float32)), name="accum")
...     def construct(self, lr, grad):
...         out = self.apply_adagrad(self.var, self.accum, lr, grad)
...         return out
...
>>> net = Net()
>>> lr = Tensor(0.001, mindspore.float32)
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(lr, grad)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.99638879e-01,  3.99296492e-01],
 [ 9.97817814e-02,  4.99281585e-01]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 6.90000057e-01,  9.90000010e-01],
 [ 2.10000008e-01,  1.24000001e+00]]))
```


`mindspore.ops.ApplyAdagradDA`

class `mindspore.ops.ApplyAdagradDA` (*use_locking=False*)

根据 Adagrad 算法更新 *var*。

Adagrad 算法在论文 [Adaptive Subgradient Methods for Online Learning and Stochastic Optimization](#) 中提出。

$$\begin{aligned}
 &grad\_accum += grad \\
 &grad\_squared\_accum += grad * grad \\
 &tmp\_val = \begin{cases} sign(grad\_accum) * \max\{|grad\_accum| - l1 * global\_step, 0\} & \text{if } l1 > 0 \\ grad\_accum & \text{otherwise} \end{cases} \\
 &x\_value = -1 * lr * tmp\_val \\
 &y\_value = l2 * global\_step * lr + \sqrt{grad\_squared\_accum} \\
 &var = \frac{x\_value}{y\_value}
 \end{aligned}$$

var、*gradient_accumulator*、*gradient_squared_accumulator* 和 *grad* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则较低精度的数据类型将转换为相对最高精度的数据类型。

参数：

- **use_locking** (bool) - 如果为 `True`，*var* 和 *gradient_accumulator* 的更新将受到锁的保护。否则，行为为未定义，很可能出现较少的冲突。默认值为 `False`。

输入：

- **var** (Union[Parameter, Tensor]) - 要更新的变量。数据类型必须为 `float16` 或 `float32`。shape: $(N, *)$ ，其中 $*$ 表示任意数量的附加维度。
- **gradient_accumulator** (Union[Parameter, Tensor]) - 要更新累积的梯度，为公式中的 *grad_accum*。shape 必须与 *var* 相同。
- **gradient_squared_accumulator** (Union[Parameter, Tensor]) - 要更新的平方累积的梯度，为公式中的 *grad_squared_accum*。shape 必须与 *var* 相同。
- **grad** (Tensor) - 梯度，为一个 Tensor。shape 必须与 *var* 相同。
- **lr** ([Number, Tensor]) - 学习率。必须是 `Scalar`。数据类型为 `float32` 或 `float16`。
- **l1** ([Number, Tensor]) - L1 正则化。必须是 `Scalar`。数据类型为 `float32` 或 `float16`。
- **l2** ([Number, Tensor]) - L2 正则化。必须是 `Scalar`。数据类型为 `float32` 或 `float16`。
- **global_step** ([Number, Tensor]) - 训练步骤的编号。必须是 `Scalar`。数据类型为 `int32` 或 `int64`。

输出：

1 个 Tensor 组成的 tuple，更新后的参数或者 Tensor。

- **var** (Tensor) - shape 和数据类型与 *var* 相同。

异常：

- **TypeError** - 如果 *var*、*gradient_accumulator* 或 *gradient_squared_accumulator* 不是 `Parameter` 或者 `Tensor`。
- **TypeError** - 如果 *grad* 不是 `Tensor`。
- **TypeError** - 如果 *lr*、*l1*、*l2* 或者 *global_step* 既不是数值型也不是 `Tensor`。
- **TypeError** - 如果 *use_locking* 不是 `bool`。
- **TypeError** - 如果 *var*、*gradient_accumulator*、*gradient_squared_accumulator*、*grad*、*lr*、*l1* 或 *l2* 的数据类型既不是 `float16` 也不是 `float32`。

- **TypeError** - 如果 `gradient_accumulator`、`gradient_squared_accumulator`、`grad` 与 `var` 的数据类型不相同。
- **TypeError** - 如果 `global_step` 的数据类型不是 `int32` 也不是 `int64`。
- **ValueError** - 如果 `lr`、`l1`、`l2` 和 `global_step` 的 `shape` 大小不为 0。
- **TypeError** - 如果 `var`、`gradient_accumulator`、`gradient_squared_accumulator` 和 `grad` 不支持数据类型转换。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class ApplyAdagradDANet(nn.Cell):
...     def __init__(self, use_locking=False):
...         super(ApplyAdagradDANet, self).__init__()
...         self.apply_adagrad_d_a = ops.ApplyAdagradDA(use_locking)
...         self.var = Parameter(Tensor(np.array([[0.6, 0.4], [0.1, 0.5]]).astype(np.
↪ float32)), name="var")
...         self.gradient_accumulator = Parameter(Tensor(np.array([[0.1, 0.3],
...                               [0.1, 0.5]]).astype(np.
↪ float32)),
...                               name="gradient_accumulator")
...         self.gradient_squared_accumulator = Parameter(Tensor(np.array([[0.2, 0.1],
...                               [0.1, 0.2]]).
↪ astype(np.float32)),
...                               name="gradient_squared_accumulator")
...         self.gradient_accumulator = Parameter(Tensor(np.array([[0.1, 0.3],
...                               [0.1, 0.5]]).astype(np.
↪ float32)),
...                               name="gradient_accumulator")
...     def construct(self, grad, lr, l1, l2, global_step):
...         out = self.apply_adagrad_d_a(self.var, self.gradient_accumulator,
...                                       self.gradient_squared_accumulator, grad, lr, l1, l2,
↪ global_step)
...         return out
>>> net = ApplyAdagradDANet()
>>> grad = Tensor(np.array([[0.3, 0.4], [0.1, 0.2]]).astype(np.float32))
>>> lr = Tensor(0.001, mstype.float32)
>>> l1 = Tensor(0.001, mstype.float32)
>>> l2 = Tensor(0.001, mstype.float32)
>>> global_step = Tensor(2, mstype.int32)
>>> output = net(grad, lr, l1, l2, global_step)
>>> print(output)
[[-0.00073906, -0.00136889],
 [-0.00059699, -0.00142478]]
```

mindspore.ops.ApplyAdagradV2

class mindspore.ops.**ApplyAdagradV2** (*epsilon*, *update_slots=True*)

根据 Adagrad 算法更新相关参数或者 Tensor。

Adagrad 算法在论文 [Adaptive Subgradient Methods for Online Learning and Stochastic Optimization](#) 中提出。

$$\begin{aligned} accum &+ = grad * grad \\ var &- = lr * grad * \frac{1}{\sqrt{accum + \epsilon}} \end{aligned}$$

其中 ϵ 表示 *epsilon*。

var、*accum* 和 *grad* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则低精度数据类型将转换为相对最高精度的数据类型。

说明: *ApplyAdagradV2* 与 *ApplyAdagrad* 不同点在于 *ApplyAdagradV2* 多一个较小的常量值 ϵ 。

参数:

- **epsilon** (float) - 添加到分母上的较小值，以确保数值的稳定性。
- **update_slots** (bool) - 如果为 True，则将更新 *accum*。默认值: True。

输入:

- **var** (Union[Parameter, Tensor]) - 要更新的变量。为任意维度，其数据类型为 float16 或 float32。其 shape 为 $(N,*)$ ，其中 $*$ 为任意数量的额外维度。
- **accum** (Union[Parameter, Tensor]) - 要更新的累积。shape 必须与 *var* 相同。
- **lr** (Union[Number, Tensor]) - 学习率，必须是 float 或具有 float16 或 float32 数据类型的 Scalar 的 Tensor。
- **grad** (Tensor) - 梯度，为一个 Tensor。shape 必须与 *var* 相同。

输出:

2 个 Tensor 组成的 tuple，更新后的参数或者 Tensor。

- **var** (Tensor) - shape 和数据类型与 *var* 相同。
- **accum** (Tensor) - shape 和数据类型与 *accum* 相同。

异常:

- **TypeError** - 如果 *var*、*accum*、*lr* 或 *grad* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *lr* 既不是数值型也不是 Tensor。
- **TypeError** - 如果 *var*、*accum* 和 *grad* 不支持数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
```

(下页继续)

(续上页)

```

...     super(Net, self).__init__()
...     self.apply_adagrad_v2 = ops.ApplyAdagradV2(epsilon=1e-6)
...     self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                         [0.1, 0.5]]).astype(np.float32)), name="var"
... ↪)
...     self.accum = Parameter(Tensor(np.array([[0.6, 0.5],
...                                         [0.2, 0.6]]).astype(np.float32)), name=
... ↪"accum")
...     def construct(self, lr, grad):
...         out = self.apply_adagrad_v2(self.var, self.accum, lr, grad)
...         return out
...
>>> net = Net()
>>> lr = Tensor(0.001, mindspore.float32)
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(lr, grad)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.99638879e-01,  3.99296492e-01],
 [ 9.97817814e-02,  4.99281585e-01]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 6.90000057e-01,  9.90000010e-01],
 [ 2.10000008e-01,  1.24000001e+00]]))

```

mindspore.ops.ApplyAdaMax

class mindspore.ops.ApplyAdaMax

根据 AdaMax 算法更新相关参数或者 Tensor。

AdaMax 优化器是参考 [Adam 论文](#) 中 Adamax 优化相关内容所实现的。

更新公式如下：

$$\begin{aligned}
 m_{t+1} &= \beta_1 * m_t + (1 - \beta_1) * g \\
 v_{t+1} &= \max(\beta_2 * v_t, |g|) \\
 var &= var - \frac{l}{1 - \beta_1^{t+1}} * \frac{m_{t+1}}{v_{t+1} + \epsilon}
 \end{aligned}$$

t 表示更新步数, m 为一阶矩, m_t 是上一步的 m_{t+1} , v 为二阶矩, v_t 是上一步的 v_{t+1} , l 代表学习率 lr , g 代表 $grad$, β_1, β_2 代表 $beta1$ 和 $beta2$, β_1^{t+1} 代表 $beta1\_power$, var 代表要更新的网络参数, ϵ 代表 $epsilon$ 。

var 、 m 、 v 和 $grad$ 的输入符合隐式类型转换规则, 使数据类型一致。如果它们具有不同的数据类型, 则低精度数据类型将转换为相对最高精度的数据类型。

输入：

- **var** (Union[Parameter, Tensor]) - 待更新的网络参数或者 Tensor, 为任意维度。数据类型为 float32 或 float16。其 shape 为 $(N, *)$, 其中 $*$ 为任意数量的额外维度。
- **m** (Union[Parameter, Tensor]) - 一阶矩, shape 与 var 相同。数据类型为 float32 或 float16。
- **v** (Union[Parameter, Tensor]) - 二阶矩。shape 与 var 相同。数据类型为 float32 或 float16。
- **beta1_power** (Union[Number, Tensor]) - $beta_1^t$, 必须是 Scalar。数据类型为 float32 或 float16。
- **lr** (Union[Number, Tensor]) - 学习率, 公式中的 l , 必须是 Scalar。数据类型为 float32 或 float16。
- **beta1** (Union[Number, Tensor]) - 一阶矩的指数衰减率, 必须是 Scalar。数据类型为 float32 或 float16。
- **beta2** (Union[Number, Tensor]) - 二阶矩的指数衰减率, 必须是 Scalar。数据类型为 float32 或 float16。

- **epsilon** (Union[Number, Tensor]) - 加在分母上的值, 以确保数值稳定, 必须是 Scalar。数据类型为 float32 或 float16。
- **grad** (Tensor) - 为梯度, 是一个 Tensor, shape 与 *var* 相同。数据类型为 float32 或 float16。

输出:

3 个 Tensor 组成的 tuple, 更新后的数据。

- **var** (Tensor) - shape 和数据类型与 *var* 相同。
- **m** (Tensor) - shape 和数据类型与 *m* 相同。
- **v** (Tensor) - shape 和数据类型与 *v* 相同。

异常:

- **TypeError** - 如果 *var*、*m*、*v*、*beta_power*、*lr*、*beta1*、*beta2*、*epsilon* 或 *grad* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *beta_power*、*lr*、*beta1*、*beta2* 或 *epsilon* 既不是数值型也不是 Tensor。
- **TypeError** - 如果 *grad* 不是 Tensor。
- **TypeError** - 如果 *var*、*m*、*v* 和 *grad* 不支持数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_ada_max = ops.ApplyAdaMax()
...         self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                                [0.1, 0.5]]).astype(np.float32)), name="var")
...         self.m = Parameter(Tensor(np.array([[0.6, 0.5],
...                                                [0.2, 0.6]]).astype(np.float32)), name="m")
...         self.v = Parameter(Tensor(np.array([[0.9, 0.1],
...                                                [0.7, 0.8]]).astype(np.float32)), name="v")
...     def construct(self, beta1_power, lr, beta1, beta2, epsilon, grad):
...         out = self.apply_ada_max(self.var, self.m, self.v, beta1_power, lr, beta1, beta2,
...                                   epsilon, grad)
...         return out
...
>>> net = Net()
>>> beta1_power = Tensor(0.9, mindspore.float32)
>>> lr = Tensor(0.001, mindspore.float32)
>>> beta1 = Tensor(0.9, mindspore.float32)
>>> beta2 = Tensor(0.99, mindspore.float32)
>>> epsilon = Tensor(1e-10, mindspore.float32)
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(beta1_power, lr, beta1, beta2, epsilon, grad)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.93602717e-01,  3.92571449e-01],
 [ 9.72582996e-02,  4.92249995e-01]]), Tensor(shape=[2, 2], dtype=Float32, value=
```

(下页继续)

(续上页)

```
[ [ 5.69999993e-01, 5.19999981e-01],
  [ 1.89999998e-01, 6.20000005e-01]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 8.90999973e-01, 6.99999988e-01],
 [ 6.93000019e-01, 8.00000012e-01]])
```

mindspore.ops.ApplyAdamWithAmsgradV2

class mindspore.ops.**ApplyAdamWithAmsgradV2** (*use_locking=False*)

根据 Adam 算法更新变量 *var*。

更新公式如下：

$$\begin{aligned} lr_t &:= learning\_rate * \sqrt{1 - \beta_2^t} / (1 - \beta_1^t) \\ m_t &:= \beta_1 * m_{t-1} + (1 - \beta_1) * g \\ v_t &:= \beta_2 * v_{t-1} + (1 - \beta_2) * g * g \\ \hat{v}_t &:= \max(\hat{v}_{t-1}, v_t) \\ var &:= var - lr_t * m_t / (\sqrt{\hat{v}_t} + \epsilon) \end{aligned}$$

t 表示更新步数, m 为一阶矩, v 为二阶矩, $\hat{v}_t$ 表示 *vhat*, lr 代表学习率, g 代表 *grad*, β_1, β_2 代表 *beta1* 和 *beta2*, β_1^t 代表 *beta1_power*, β_2^t 代表 *beta2_power*, var 代表要更新的网络参数, ϵ 代表 *epsilon*。

所有输入符合隐式类型转换规则, 使数据类型一致。如果它们具有不同的数据类型, 则低精度数据类型将转换为相对最高精度的数据类型。

参数：

- **use_locking** (bool) - 如果为 True, *var*, *m* 和 *v* 的更新将受到锁的保护。否则, 可能出现较少的冲突。默认值为 False。

输入：

- **var** (Union[Parameter, Tensor]) - 待更新的网络参数或者 Tensor, 为任意维度。数据类型为 float16、float32 或 float64。
- **m** (Union[Parameter, Tensor]) - 一阶矩, shape 与 *var* 相同。
- **v** (Union[Parameter, Tensor]) - 二阶矩。shape 与 *var* 相同。
- **vhat** (Union[Parameter, Tensor]) - 公式中的 $\hat{v}_t$ 。shape 和类型与 *var* 相同。
- **beta1_power** (Union[float, Tensor]) - 公式中的 β_1^t , 数据类型为 float16、float32 或 float64。
- **beta2_power** (Union[float, Tensor]) - 公式中的 β_2^t , 数据类型为 float16、float32 或 float64。
- **lr** (Union[float, Tensor]) - 学习率。数据类型为 float16、float32 或 float64 的 Tensor。
- **beta1** (Union[float, Tensor]) - 一阶矩的指数衰减率。数据类型为 float16、float32 或 float64。
- **beta2** (Union[float, Tensor]) - 二阶矩的指数衰减率。数据类型为 float16、float32 或 float64。
- **epsilon** (Union[float, Tensor]) - 加在分母上的值, 以确保数值稳定。数据类型为 float16、float32 或 float64。
- **grad** (Tensor) - 为梯度, shape 与 *var* 相同。

输出：

4 个 Tensor 组成的 tuple, 更新后的数据。

- **var** (Tensor) - shape 和数据类型与 *var* 相同。
- **m** (Tensor) - shape 和数据类型与 *m* 相同。

- **v** (Tensor) - shape 和数据类型与 *v* 相同。
- **vhat** (Tensor) - shape 和数据类型与 *vhat* 相同。

异常:

- **TypeError** - 如果 *var*、*m*、*v*、*vhat* 不是 Parameter 或者 Tensor。
- **TypeError** - 如果 *var*、*m*、*v*、*vhat*、*beta1_power*、*beta2_power*、*lr*、*beta1*、*beta2*、*epsilon* 或 *grad* 的数据类型既不是 float16 也不是 float32, 也不是 float64。
- **RuntimeError** - 如果 *var*、*m*、*v*、*vhat* 和 *grad* 不支持数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import ops
>>> import mindspore.nn as nn
>>> from mindspore import Tensor, Parameter
>>> import numpy as np
>>> class ApplyAdamWithAmsgradNet(nn.Cell):
...     def __init__(self, use_locking=False):
...         super(ApplyAdamWithAmsgradNet, self).__init__()
...         self.apply_adam_with_amsgrad = ops.ApplyAdamWithAmsgradV2(use_locking)
...         self.var = Parameter(Tensor(np.array([[0.2, 0.2], [0.2, 0.2]]).astype(np.
↪float32)), name="var")
...         self.m = Parameter(Tensor(np.array([[0.1, 0.2], [0.4, 0.3]]).astype(np.float32)),
↪ name="m")
...         self.v = Parameter(Tensor(np.array([[0.2, 0.1], [0.3, 0.4]]).astype(np.float32)),
↪ name="v")
...         self.vhat = Parameter(Tensor(np.array([[0.1, 0.2], [0.6, 0.2]]).astype(np.
↪float32)), name="vhat")
...         self.beta1 = 0.8
...         self.beta2 = 0.999
...         self.epsilon = 1e-8
...         self.beta1_power = 0.9
...         self.beta2_power = 0.999
...         self.lr = 0.01
...
...     def construct(self, grad):
...         out = self.apply_adam_with_amsgrad(self.var, self.m, self.v, self.vhat,
...                                             self.beta1_power, self.beta2_power, self.lr,
...                                             self.beta1, self.beta2, self.epsilon, grad)
...         return out
>>> net = ApplyAdamWithAmsgradNet()
>>> grad = Tensor(np.array([[0.4, 0.2], [0.2, 0.3]]).astype(np.float32))
>>> output = net(grad)
>>> print(net.var.asnumpy())
[[0.19886853 0.1985858 ]
 [0.19853032 0.19849943]]
```

mindspore.ops.ApplyAddSign

class mindspore.ops.ApplyAddSign

根据 AddSign 算法更新相关参数或者 Tensor。

$$\begin{aligned}
 m_{t+1} &= \beta * m_t + (1 - \beta) * g \\
 \text{update} &= (\alpha + \text{sign\_decay} * \text{sign}(g) * \text{sign}(m)) * g \\
 \text{var} &= \text{var} - lr_{t+1} * \text{update}
 \end{aligned}$$

t 代表更新步长, 而 m 代表第一个动量矩阵, m_t 是 m_{t+1} 的最后时刻, lr 代表学习率 lr , g 代表 $grad$, α 代表 $alpha$, β 代表 $beta$ 。

所有输入数据在 Ascend 上支持的类型为: float16 和 float32, CPU 和 GPU 上支持的类型为: float16、float32 和 float64。

var 、 $accum$ 和 $grad$ 的输入, $sign\_decay$ 和 $beta$ 的输入都遵循隐式类型转换规则, 使数据类型一致。如果它们具有不同的数据类型, 则低精度数据类型将转换为相对最高精度的数据类型。

输入:

- **var** (Union[Parameter, Tensor]) - 要更新的权重或者 Tensor。任意维度, 其 shape 为 $(N, *)$, 其中 $*$ 为任意数量的额外维度。
- **m** (Union[Parameter, Tensor]) - 要更新的权重或者 Tensor, shape 与 var 相同。
- **lr** (Union[Number, Tensor]) - 学习率, 必须是 Scalar。
- **sign_decay** (Union[Number, Tensor]) - 必须是 Scalar。
- **alpha** (Union[Number, Tensor]) - 必须是 Scalar。
- **beta** (Union[Number, Tensor]) - 指数衰减率, 必须是 Scalar。
- **grad** (Tensor) - 梯度, shape 与 var 相同。

输出:

2 个 Tensor 组成的 tuple, 更新后的数据。

- **var** (Tensor) - shape 和数据类型与 var 相同。
- **m** (Tensor) - shape 和数据类型与 m 相同。

异常:

- **TypeError** - 如果 var 、 lr 、 $alpha$ 的数据类型既不是 float16 也不是 float32 或者 float64。
- **TypeError** - 如果 $sign\_decay$ 和 $beta$ 的数据类型都不是 float16、float32 或者 float64。
- **TypeError** - 如果 lr 、 $alpha$ 或 $sign\_decay$ 既不是数值型, 也不是 Tensor。
- **TypeError** - 如果 $grad$ 不是 Tensor。
- **TypeError** - 如果不支持参数的 var 、 $accum$ 和 $grad$ 数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```

>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):

```

(下页继续)

(续上页)

```

...     super(Net, self).__init__()
...     self.apply_add_sign = ops.ApplyAddSign()
...     self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                         [0.1, 0.5]]).astype(np.float32)), name="var"
... ↪ ")
...     self.m = Parameter(Tensor(np.array([[0.6, 0.5],
...                                         [0.2, 0.6]]).astype(np.float32)), name="m")
...     self.lr = 0.001
...     self.alpha = 1.0
...     self.sign_decay = 0.99
...     self.beta = 0.9
...     def construct(self, grad):
...         out = self.apply_add_sign(self.var, self.m, self.lr, self.alpha, self.sign_decay,
... ↪ self.beta, grad)
...         return out
...
>>> net = Net()
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(grad)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.99403024e-01,  3.98607016e-01],
 [ 9.98010039e-02,  4.98407990e-01]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.70000052e-01,  5.19999981e-01],
 [ 1.89999998e-01,  6.20000064e-01]]))

```

mindspore.ops.ApplyCenteredRMSProp

class mindspore.ops.ApplyCenteredRMSProp (*use_locking=False*)

居中 RMSProp 算法优化器。

请参考源代码中的用法: `mindspore.nn.RMSProp`。

更新公式如下:

$$\begin{aligned}
 g_{t+1} &= \rho g_t + (1 - \rho) \nabla Q_i(w) \\
 s_{t+1} &= \rho s_t + (1 - \rho) (\nabla Q_i(w))^2 \\
 m_{t+1} &= \beta m_t + \frac{\eta}{\sqrt{s_{t+1} - g_{t+1}^2 + \epsilon}} \nabla Q_i(w) \\
 w &= w - m_{t+1}
 \end{aligned}$$

其中 w 代表 `var`, 将会更新。 g_{t+1} 代表 `mean_gradient`, g_t 是上一步的 g_{t+1} 。 s_{t+1} 代表 `mean_square`, s_t 是上一步的 s_{t+1} , m_{t+1} 代表 `moment`, m_t 是上一步的 m_{t+1} 。 ρ 代表 `decay`。 β 是动量, 代表 `momentum`。 ϵ 是一个添加在分母上的较小值, 以避免被零除, 表示 `epsilon`。 η 代表 `learning_rate`。 $\nabla Q_i(w)$ 代表 `grad`。

说明: `ApplyCenteredRMSProp` 和 `ApplyRMSProp` 的区别在于前者使用居中 RMSProp 算法, 而居中 RMSProp 算法使用居中第二矩阵的估计 (即, 归一化的方差), 而不是使用 (不确定的) 第二矩阵的正则 RMSProp。这通常有助于训练, 但在计算和内存方面消耗更大。

警告: 在此算法的密集实现中, `mean_gradient`、`mean_square` 和 `moment` 在 `grad` 为零时仍将被更新。但在稀疏实现中, `mean_gradient`、`mean_square` 和 `moment` 不会在 `grad` 为零的迭代中被更新。

参数:

- **use_locking** (bool) - 是否对参数更新增加锁保护。默认值: `False`。

输入:

- **var** (Parameter) - 要更新的权重。
- **mean_gradient** (Tensor) - 均值梯度, 数据类型必须与 *var* 相同。
- **mean_square** (Tensor) - 均方梯度, 数据类型必须与 *var* 相同。
- **moment** (Tensor) - *var* 的增量, 数据类型必须与 *var* 相同。
- **grad** (Tensor) - 梯度, 数据类型必须与 *var* 相同。
- **learning_rate** (Union[Number, Tensor]) - 学习率。必须是 float 或 Scalar 的 Tensor, 数据类型为 float16 或 float32。
- **decay** (float) - 衰减率。
- **momentum** (float) - 动量。
- **epsilon** (float) - 添加在分母上的较小值, 以避免被零除。

输出:

Tensor, 更新后的数据。

异常:

- **TypeError** - 如果 *use_locking* 不是 bool。
- **TypeError** - 如果 *var*、*mean_gradient*、*mean_square*、*moment* 或 *grad* 不是 Tensor。
- **TypeError** - 如果 *learning_rate* 既不是数值型也不是 Tensor。
- **TypeError** - 如果 *learning_rate* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *decay*、*momentum* 或 *epsilon* 不是 float。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_centerd_rms_prop = ops.ApplyCenteredRMSProp()
...         self.var = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="var")
...
...     def construct(self, mean_grad, mean_square, moment, grad, decay, momentum, epsilon,
...lr):
...         out = self.apply_centerd_rms_prop(self.var, mean_grad, mean_square, moment, grad,
...lr, decay, momentum, epsilon)
...         return out
...
>>> net = Net()
>>> mean_grad = Tensor(np.ones([2, 2]).astype(np.float32))
>>> mean_square = Tensor(np.ones([2, 2]).astype(np.float32))
>>> moment = Tensor(np.ones([2, 2]).astype(np.float32))
>>> grad = Tensor(np.ones([2, 2]).astype(np.float32))
```

(下页继续)

(续上页)

```
>>> output = net(mean_grad, mean_square, moment, grad, 0.0, 1e-10, 0.001, 0.01)
>>> print(net.var.asnumpy())
[[0.68377227 0.68377227]
 [0.68377227 0.68377227]]
```

mindspore.ops.ApplyFtrl

class mindspore.ops.**ApplyFtrl** (use_locking=False)

根据 FTRL 算法更新相关参数或者 Tensor。

更多详细信息请参见 [mindspore.nn.FTRL](#)。

说明:

- 当前在 Ascend 平台上仅支持正数，其余场景计算结果未定义。
- *var*、*accum*、*linear* 和 *grad* 的输入符合隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则低精度数据类型将转换为相对最高精度的数据类型。

参数:

- **use_locking** (bool) - 是否对参数更新加锁保护。默认值: False。

输入:

- **var** (Union[Parameter, Tensor]) - 要更新的权重或者 Tensor。数据类型必须为 float16 或 float32。shape: (N,*), 其中 * 表示任意数量的附加维度。
- **accum** (Union[Parameter, Tensor]) - 要更新的累积, shape 必须与 *var* 相同。
- **linear** (Union[Parameter, Tensor]) - 要更新的线性系数, shape 必须与 *var* 相同。
- **grad** (Tensor) - 梯度。数据类型必须为 float16 或 float32。
- **lr** (Union[Number, Tensor]) - 学习率, 必须为正值。它必须是 float 或数据类型为 float16 或 float32 的 Scalar 的 Tensor。默认值: 0.001。
- **l1** (Union[Number, Tensor]) - l1 正则化, 必须大于或等于零。它必须是 float 类型或数据类型为 float16 或 float32 的 Scalar 的 Tensor。默认值: 0.0。
- **l2** (Union[Number, Tensor]) - l2 正则化, 必须大于或等于零。它必须是 float 类型或数据类型为 float16 或 float32 的 Scalar 的 Tensor。默认值: 0.0。
- **lr_power** (Union[Number, Tensor]) - 在训练期间控制降低学习率, 必须小于或等于零。如果 *lr_power* 为零, 则使用固定学习率。它必须是 float 类型或数据类型为 float16 或 float32 的 Scalar 的 Tensor。默认值: -0.5。

输出:

- **var** (Tensor) - 表示更新后的 *var*。由于输入已更新, 因此当平台为 GPU 时, 此值始终为零。

异常:

- **TypeError** - 如果 *use_locking* 不是 bool。
- **TypeError** - 如果 *var*、*grad*、*lr*、*l1*、*l2* 或 *lr_power* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *lr*、*l1*、*l2* 或 *lr_power* 既不是数值型也不是 Tensor。
- **TypeError** - 如果 *grad* 不是 Tensor。
- **TypeError** - 如果 *var*、*accum*、*linear* 参数类型不一致。

- **TypeError** - 如果 *grad*、*lr*、*l1*、*l2*、*lr_power* 的参数类型与 *var* 不一致且精度大于 *var*。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class ApplyFtrlNet(nn.Cell):
...     def __init__(self):
...         super(ApplyFtrlNet, self).__init__()
...         self.apply_ftrl = ops.ApplyFtrl()
...         self.lr = 0.001
...         self.l1 = 0.0
...         self.l2 = 0.0
...         self.lr_power = -0.5
...         self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                                [0.1, 0.5]]).astype(np.float32)), name="var")
...         self.accum = Parameter(Tensor(np.array([[0.6, 0.5],
...                                                [0.2, 0.6]]).astype(np.float32)), name="accum")
...         self.linear = Parameter(Tensor(np.array([[0.9, 0.1],
...                                                [0.7, 0.8]]).astype(np.float32)), name="linear")
...     def construct(self, grad):
...         out = self.apply_ftrl(self.var, self.accum, self.linear, grad, self.lr, self.l1, self.l2, self.lr_power)
...         return out
>>> net = ApplyFtrlNet()
>>> input_x = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(input_x)
>>> print(net.var.asnumpy())
[[ 0.0390525  0.11492836]
 [ 0.00066425  0.15075898]]
```

mindspore.ops.ApplyGradientDescent

class mindspore.ops.ApplyGradientDescent

通过从 *var* 中减去 $\alpha * \delta$ 来更新 *var*。

$$var = var - \alpha * \delta$$

其中 α 代表 *alpha*， δ 代表 *delta*。

var 和 *delta* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则低精度数据类型将转换为相对最高精度数据类型。

输入：

- **var** (Union[Parameter, Tensor]) - 要更新的变量，为任意维度，其数据类型为 float32 或 float16。其 shape 为 (N,*), 其中 * 为任意数量的额外维度。
- **alpha** (Union[Number, Tensor]) - 调节系数，必须是 Scalar。数据类型为 float32 或 float16。

- **delta** (Tensor) - 变化的 Tensor, shape 与 *var* 相同。

输出:

Tensor, 更新后的 *var*。

异常:

- **TypeError** - 如果 *var* 或 *alpha* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *delta* 不是 Tensor。
- **TypeError** - 如果 *alpha* 既不是数值型也不是 Tensor。
- **TypeError** - 如果不支持 *var* 和 *delta* 数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_gradient_descent = ops.ApplyGradientDescent()
...         self.var = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="var")
...         self.alpha = 0.001
...     def construct(self, delta):
...         out = self.apply_gradient_descent(self.var, self.alpha, delta)
...         return out
...
>>> net = Net()
>>> delta = Tensor(np.array([[0.1, 0.1], [0.1, 0.1]]).astype(np.float32))
>>> output = net(delta)
>>> print(output)
[[0.9999 0.9999]
 [0.9999 0.9999]]
```

mindspore.ops.ApplyMomentum

class mindspore.ops.ApplyMomentum (*use_nesterov=False, use_locking=False, gradient_scale=1.0*)

使用动量算法的优化器。

更多详细信息, 请参阅论文 [On the importance of initialization and momentum in deep learning](#)。

输入的 *variable*、*accumulation* 和 *gradient* 的输入遵循隐式类型转换规则, 使数据类型一致。如果它们具有不同的数据类型, 则低精度数据类型将转换为相对最高精度的数据类型。

有关公式和用法的更多详细信息, 请参阅 [mindspore.nn.Momentum](#)。

参数:

- **use_locking** (bool, 可选) - 是否对参数更新加锁保护。默认值: False。
- **use_nesterov** (bool, 可选) - 是否使用 nesterov 动量。默认值: False。
- **gradient_scale** (float, 可选) - 梯度的缩放比例。默认值: 1.0。

输入:

- **variable** (Union[Parameter, Tensor]) - 要更新的权重, 数据类型必须为 float64、int64、float、float16、int16、int32、int8、uint16、uint32、uint64、uint8、complex64、complex128。
- **accumulation** (Union[Parameter, Tensor]) - 按动量权重计算的累加梯度值, 数据类型与 *variable* 相同。
- **learning_rate** (Union[Number, Tensor]) - 学习率, 必须是 float64、int64、float、float16、int16、int32、int8、uint16、uint32、uint64、uint8、complex64、complex128 或为 float64、int64、float、float16、int16、int32、int8、uint16、uint32、uint64、uint8、complex64、complex128 数据类型的 Scalar 的 Tensor。
- **gradient** (Tensor) - 梯度, 数据类型与 *variable* 相同。
- **momentum** (Union[Number, Tensor]) - 动量, 必须是 float64、int64、float、float16、int16、int32、int8、uint16、uint32、uint64、uint8、complex64、complex128 类型的数值, 或是具有 float64、int64、float、float16、int16、int32、int8、uint16、uint32、uint64、uint8、complex64、complex128 数据类型的 Scalar 的 Tensor。

输出:

Tensor, 更新后的参数。

异常:

- **TypeError** - 如果 *use_locking* 或 *use_nesterov* 不是 bool, 或 *gradient_scale* 不是 float。
- **TypeError** - 如果 *var*、*accum* 和 *grad* 不支持数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_momentum = ops.ApplyMomentum()
...         self.variable = Parameter(Tensor(np.array([[0.6, 0.4],
...                                                    [0.1, 0.5]]).astype(np.float32)), name=
↪ "variable")
...         self.accumulate = Parameter(Tensor(np.array([[0.6, 0.5],
...                                                    [0.2, 0.6]]).astype(np.float32)), name=
↪ "accumulate")
...     def construct(self, lr, grad, moment):
...         out = self.apply_momentum(self.variable, self.accumulate, lr, grad, moment)
...         return out
>>> net = Net()
>>> lr = Tensor(0.1, mindspore.float32)
>>> moment = Tensor(0.9, mindspore.float32)
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(lr, grad, moment)
>>> print(output)
[[0.51600003 0.285      ]
 [0.072      0.366      ]]
```

mindspore.ops.ApplyPowerSign

class mindspore.ops.ApplyPowerSign

根据 AddSign 算法更新相关参数或者 Tensor。

AddSign 算法可参阅论文 [Neural Optimizer Search with Reinforcement Learning](#)。

$$\begin{aligned}
 m_{t+1} &= \beta * m_t + (1 - \beta) * g \\
 \text{update} &= \exp(\logbase * \text{sign\_decay} * \text{sign}(g) * \text{sign}(m)) * g \\
 \text{var} &= \text{var} - lr_{t+1} * \text{update}
 \end{aligned}$$

t 表示更新步数, 而 m 为一阶矩, m_t 是上一步的 m_{t+1} , lr 表示 lr , g 表示 $grad$, β 表示 $beta$ 。

所有输入都遵循隐式类型转换规则, 以使数据类型一致。如果 lr 、 $\logbase$ 、 sign_decay 或 $beta$ 是数值型, 则会自动转换为 Tensor, 数据类型与操作中涉及的 Tensor 的数据类型一致。如果输入是 Tensor, 并且具有不同的数据类型, 则低精度数据类型将转换为最高精度的数据类型。

说明: 目前 Ascend 平台上暂未开放对 float64 数据类型的支持。

输入:

- **var** (Union[Parameter, Tensor]) - 要更新的变量。数据类型为 float64、float32 或 float16。如果 var 的数据类型为 float16, 则所有输入的数据类型必须与 var 相同。shape: $(N, *)$, 其中 $*$ 表示任意数量的附加维度。
- **m** (Union[Parameter, Tensor]) - 要更新的变量, shape 与 var 相同。
- **lr** (Union[Number, Tensor]) - 学习率, 应该是 Scalar 或 Tensor, 数据类型为 float64、float32 或 float16。
- **logbase** (Union[Number, Tensor]) - 应该是 Scalar 或 Tensor, 数据类型为 float64、float32 或 float16。
- **sign_decay** (Union[Number, Tensor]) - 应该是 Scalar 或 Tensor, 数据类型为 float64、float32 或 float16。
- **beta** (Union[Number, Tensor]) - 指数衰减率, 应该是 Scalar 或 Tensor, 数据类型为 float64、float32 或 float16。
- **grad** (Tensor) - 梯度, shape 与 var 相同。

输出:

2 个 Tensor 组成的 tuple, 更新后的参数或者 Tensor。

- **var** (Tensor) - shape 和数据类型与 var 相同。
- **m** (Tensor) - shape 和数据类型与 m 相同。

异常:

- **TypeError** - 如果 var 、 lr 、 $\logbase$ 、 sign_decay 、 $beta$ 或 $grad$ 的数据类型不是 float16、float32 或者 float64。
- **TypeError** - 如果 lr 、 $\logbase$ 、 sign_decay 或 $beta$ 既不是数值型也不是 Tensor。
- **TypeError** - 如果 $grad$ 不是 Tensor。
- **TypeError** - 如果 lr 、 $\logbase$ 、 sign_decay 和 $grad$ 不支持数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_power_sign = ops.ApplyPowerSign()
...         self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                                [0.1, 0.5]]).astype(np.float32)), name="var")
...
...         self.m = Parameter(Tensor(np.array([[0.6, 0.5],
...                                                [0.2, 0.6]]).astype(np.float32)), name="m")
...
...         self.lr = 0.001
...         self.logbase = np.e
...         self.sign_decay = 0.99
...         self.beta = 0.9
...     def construct(self, grad):
...         out = self.apply_power_sign(self.var, self.m, self.lr, self.logbase,
...                                     self.sign_decay, self.beta, grad)
...         return out
...
>>> net = Net()
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(grad)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.95575690e-01,  3.89676481e-01],
 [ 9.85252112e-02,  4.88201708e-01]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.70000052e-01,  5.19999981e-01],
 [ 1.89999998e-01,  6.20000064e-01]]))
```

mindspore.ops.ApplyProximalAdagrad

class mindspore.ops.ApplyProximalAdagrad (*use_locking=False*)

根据 Proximal Adagrad 算法更新网络参数或者 Tensor。请参阅论文 [Efficient Learning using Forward-Backward Splitting](#)。

$$\begin{aligned} accum &+ grad * grad \\ prox\_v &= var - lr * grad * \frac{1}{\sqrt{accum}} \\ var &= \frac{sign(prox\_v)}{1+lr*l2} * \max(|prox\_v| - lr * l1, 0) \end{aligned}$$

输入 *var*、*accum* 和 *grad* 之间必须遵守隐式类型转换规则以保证数据类型的统一。如果数据类型不同，低精度的数据类型将被自动转换到高精度的数据类型。

参数：

- **use_locking** (bool, 可选) - 是否对参数更新加锁保护。默认值：False。

输入：

- **var** (Union[Parameter, Tensor]) - 公式中的“var”。数据类型需为 float16 或 float32。shape 为 (N,*), 其中 * 表示任何数量的附加维度。
- **accum** (Union[Parameter, Tensor]) - 公式中的“accum”。与 *var* 的 shape 和数据类型相同。
- **lr** (Union[Number, Tensor]) - 学习率，必须为标量，数据类型为 float16 或 float32。
- **l1** (Union[Number, Tensor]) - l1 正则化强度，必须为标量，数据类型为 float16 或 float32。
- **l2** (Union[Number, Tensor]) - l2 正则化强度，必须为标量，数据类型为 float16 或 float32。

- **grad** (Tensor) - 梯度, 与 *var* 的 *shape* 和数据类型相同。

输出:

包含两个 Tensor 的 Tuple, 已被更新的参数或者 Tensor。

- **var** (Tensor) - 与输入 *var* 的 *shape* 和数据类型相同。
- **accum** (Tensor) - 与输入 *accum* 的 *shape* 和数据类型相同。

异常:

- **TypeError** - *use_blocking* 不是 bool 类型。
- **TypeError** - *var*、*lr*、*l1* 或 *l2* 的数据类型不是 float16 或 float32。
- **TypeError** - *lr*、*l1* 或 *l2* 的数据类型不是 Number 或 Tensor。
- **TypeError** - *grad* 不是 Tensor。
- **TypeError** - *var*、*accum* 和 *grad* 网络参数的数据类型转换错误。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_proximal_adagrad = ops.ApplyProximalAdagrad()
...         self.var = Parameter(Tensor(np.array([[0.6, 0.4],
...                                               [0.1, 0.5]]).astype(np.float32)), name="var")
...         self.accum = Parameter(Tensor(np.array([[0.6, 0.5],
...                                               [0.2, 0.6]]).astype(np.float32)), name="accum")
...         self.lr = 0.01
...         self.l1 = 0.0
...         self.l2 = 0.0
...     def construct(self, grad):
...         out = self.apply_proximal_adagrad(self.var, self.accum, self.lr, self.l1, self.l2, grad)
...         return out
>>> net = Net()
>>> grad = Tensor(np.array([[0.3, 0.7], [0.1, 0.8]]).astype(np.float32))
>>> output = net(grad)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 5.96388459e-01,  3.92964751e-01],
 [ 9.78178233e-02,  4.92815793e-01]]), Tensor(shape=[2, 2], dtype=Float32, value=
[[ 6.90000057e-01,  9.90000010e-01],
 [ 2.10000008e-01,  1.24000001e+00]]))
```

mindspore.ops.ApplyProximalGradientDescent

class mindspore.ops.ApplyProximalGradientDescent

根据 FOBOS(Forward Backward Splitting) 算法更新网络参数或者 Tensor。请参阅论文 [Efficient Learning using Forward-Backward Splitting](#)。

$$\text{prox\_v} = \text{var} - \alpha * \delta$$

$$\text{var} = \frac{\text{sign}(\text{prox\_v})}{1 + \alpha * l2} * \max(|\text{prox\_v}| - \alpha * l1, 0)$$

其中 α 为 *alpha* , δ 为 *delta* 。

输入 *var* 和 *delta* 之间必须遵守隐式类型转换规则以保证数据类型的统一。如果数据类型不同, 低精度的数据类型将被自动转换到高精度的数据类型。

输入:

- **var** (Union[Parameter, Tensor]) - Tensor, 公式中的“var”。数据类型为 float16 或 float32。shape 为 (N,*), 其中 * 表示任何数量的附加维度。
- **alpha** (Union[Number, Tensor]) - 比例系数, 必须为标量。数据类型为 float16 或 float32。
- **l1** (Union[Number, Tensor]) - l1 正则化强度, 必须为标量。数据类型为 float16 或 float32。
- **l2** (Union[Number, Tensor]) - l2 正则化强度, 必须为标量。数据类型为 float16 或 float32。
- **delta** (Tensor) - 梯度 Tensor。

输出:

Tensor, 更新后的 *var* 。

异常:

- **TypeError** - *var* 、 *alpha* 、 *l1* 或 *l2* 的数据类型非 float16 或 float32。
- **TypeError** - *alpha* 、 *l1* 或 *l2* 不是 Number 或 Tensor。
- **TypeError** - *delta* 不是 Tensor。
- **TypeError** - *var* 和 *delta* 之间的数值转换不被支持。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_proximal_gradient_descent = ops.ApplyProximalGradientDescent()
...         self.var = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="var")
...         self.alpha = 0.001
...         self.l1 = 0.1
...         self.l2 = 0.1
...     def construct(self, delta):
...         out = self.apply_proximal_gradient_descent(self.var, self.alpha, self.l1, self.
↪l2, delta)
...         return out
... 
```

(下页继续)

(续上页)

```
>>> net = Net()
>>> delta = Tensor(np.array([[0.1, 0.1], [0.1, 0.1]]).astype(np.float32))
>>> output = net(delta)
>>> print(output)
[[0.99969995 0.99969995]
 [0.99969995 0.99969995]]
```

mindspore.ops.ApplyRMSProp

class mindspore.ops.**ApplyRMSProp** (*use_locking=False*)

实现均方根传播 Root Mean Square prop(RMSProp) 算法的优化器。请参考源码中的用法 `mindspore.nn.RMSProp`。

ApplyRMSProp 算法的更新公式如下所示：

$$\begin{aligned} s_{t+1} &= \rho s_t + (1 - \rho)(\nabla Q_i(w))^2 \\ m_{t+1} &= \beta m_t + \frac{\eta}{\sqrt{s_{t+1} + \epsilon}} \nabla Q_i(w) \\ w &= w - m_{t+1} \end{aligned}$$

其中 w 代表待更新的网络参数 *var*。 s_{t+1} 为均方梯度 *mean_square*， s_t 为上一步的 s_{t+1} ， m_{t+1} 为 *moment*， m_t 为上一步的 m_{t+1} 。 ρ 为 *decay*。 β 为动量项 *momentum*。 ϵ 是避免除数为零的平滑项 *epsilon*。 η 为 *learning_rate*。 $\nabla Q_i(w)$ 代表 *grad*。

警告： 在该算法的稠密实现版本中，即使 *grad* 为零，*mean_square* 和 *moment* 仍会被更新。但在该稀疏实现版本中，当 *grad* 为零时，*mean_square* 和 *moment* 将不会被更新。

参数：

- **use_locking** (bool, 可选) - 是否对参数更新加锁保护。默认值: `False`。

输入：

- **var** (Parameter) - 待更新的网络参数。
- **mean_square** (Tensor) - 均方梯度，数据类型需与 *var* 相同。
- **moment** (Tensor) - 一阶矩，数据类型需与 *var* 相同。
- **learning_rate** (Union[Number, Tensor]) - 学习率，需为浮点数或者数据类型为 `float16` 或 `float32` 的标量矩阵。
- **grad** (Tensor) - 梯度，数据类型需与 *var* 相同。
- **decay** (float) - 衰减率，需为常量。
- **momentum** (float) - 移动平均的动量项 *momentum*，需为常量。
- **epsilon** (float) - 避免除数为零的平滑项，需为常量。

输出：

Tensor，待更新的网络参数。

异常：

- **TypeError** - *use_locking* 不是 bool 类型。
- **TypeError** - *var*、*mean_square*、*moment* 或 *decay* 不是 Tensor。
- **TypeError** - *learning_rate* 不是数值也不是 Tensor。

- **TypeError** - *decay*、*momentum* 或 *epsilon* 的数据类型非 float。
- **TypeError** - *learning_rate* 的数据类型不是 float16 或 float32。
- **ValueError** - *decay*、*momentum* 或 *epsilon* 不是常量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.apply_rms_prop = ops.ApplyRMSProp()
...         self.var = Parameter(Tensor(np.ones([2, 2]).astype(np.float32)), name="var")
...
...     def construct(self, mean_square, moment, grad, decay, momentum, epsilon, lr):
...         out = self.apply_rms_prop(self.var, mean_square, moment, lr, grad, decay,
    ↪momentum, epsilon)
...         return out
...
>>> net = Net()
>>> mean_square = Tensor(np.ones([2, 2]).astype(np.float32))
>>> moment = Tensor(np.ones([2, 2]).astype(np.float32))
>>> grad = Tensor(np.ones([2, 2]).astype(np.float32))
>>> output = net(mean_square, moment, grad, 0.0, 1e-10, 0.001, 0.01)
>>> print(net.var.asnumpy())
[[0.990005  0.990005]
 [0.990005  0.990005]]
```

mindspore.ops.LARSUpdate

class mindspore.ops.**LARSUpdate** (*epsilon=1e-05, hyperpara=0.001, use_clip=False*)
对梯度的平方和应用 LARS(layer-wise adaptive rate scaling) 算法。

更多细节请参考[mindspore.nn.LARS](#)。

参数：

- **epsilon** (float, 可选) - 添加在分母中，提高数值稳定性。默认值：1e-05。
- **hyperpara** (float, 可选) - 计算局部学习率的信任系数。默认值：0.001。
- **use_clip** (bool, 可选) - 计算局部学习速率时是否裁剪。默认值：False。

输入：

- **weight** (Tensor) - 权重 Tensor，shape: (N,*), 其中 * 表示任意数量的附加维度。
- **gradient** (Tensor) - *weight* 的梯度，与 *weight* 的 shape 和数据类型相同。
- **norm_weight** (Tensor) - 标量 Tensor，权重的平方和。
- **norm_gradient** (Tensor) - 标量 Tensor，梯度的平方和。
- **weight_decay** (Union[Number, Tensor]) - 衰减率。必须为标量 Tensor 或 Number。
- **learning_rate** (Union[Number, Tensor]) - 学习率。必须为标量 Tensor 或 Number。

输出：

Tensor，计算后的梯度。

异常：

- **TypeError** - *epsilon* 或 *hyperpara* 不是 float 类型。
- **TypeError** - *use_clip* 不是 bool 类型。
- **TypeError** - *weight*、*gradient*、*norm_weight* 或 *norm_gradient* 不是 Tensor。
- **TypeError** - *weight_decay* 或 *learning_rate* 非 Number 或 Tensor。
- **TypeError** - *gradient* 与 *weight* 的 shape 不同。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.lars = ops.LARSUpdate()
...         self.reduce = ops.ReduceSum()
...         self.square = ops.Square()
...     def construct(self, weight, gradient):
...         w_square_sum = self.reduce(self.square(weight))
...         grad_square_sum = self.reduce(self.square(gradient))
...         grad_t = self.lars(weight, gradient, w_square_sum, grad_square_sum, 0.0, 1.0)
...         return grad_t
...
>>> weight = Tensor(np.array([[0.5, 0.8, 0.2], [0.6, 0.4, 0.2]]).astype(np.float32))
>>> gradient = Tensor(np.array([[0.4, 0.4, 0.5], [0.2, 0.4, 0.3]]).astype(np.float32))
>>> net = Net()
>>> output = net(Tensor(weight), Tensor(gradient))
>>> print(output)
[[0.0005265  0.0005265 0.00065813]
 [0.00026325 0.0005265 0.00039488]]
```

mindspore.ops.SparseApplyAdagradV2

class mindspore.ops.SparseApplyAdagradV2 (*lr*, *epsilon*, *update_slots=True*, *use_locking=False*)

根据 Adagrad 算法更新相关参数或者 Tensor。

$$\begin{aligned} accum+ &= grad * grad \\ var- &= lr * grad * \frac{1}{\sqrt{accum+\epsilon}} \end{aligned}$$

ϵ 代表 *epsilon*。

var、*accum* 和 *grad* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则低精度数据类型将转换为相对最高精度的数据类型。

参数：

- **lr** (float) - 学习率。

- **epsilon** (float) - 添加到分母上的较小值, 以确保数值的稳定性。
- **update_slots** (bool, 可选) - 如果为 True, 则将更新 *accum*。默认值: True。
- **use_locking** (bool, 可选) - 是否对参数更新加锁保护。默认值: False。

输入:

- **var** (Union[Parameter, Tensor]) - 要更新的变量。为任意维度, 其数据类型为 float16 或 float32。其 shape 为 $(N, *)$, 其中 $*$ 为任意数量的额外维度。
- **accum** (Union[Parameter, Tensor]) - 要更新的累积。shape 必须与 *var* 相同。
- **grad** (Tensor) - 梯度, 为一个 Tensor。shape 必须与 *var* 相同, 且需要满足当 $var.shape > 1$ 时 $grad.shape[1:] = var.shape[1:]$ 。
- **indices** (Tensor) - *var* 和 *accum* 第一维度的索引向量, 数据类型为 int32, 且需要保证 $indices.shape[0] = grad.shape[0]$ 。indices 的值需要唯一, 否则结果将不可预测。

输出:

2 个 Tensor 组成的 tuple, 更新后的参数或者 Tensor。

- **var** (Tensor) - shape 和数据类型与 *var* 相同。
- **accum** (Tensor) - shape 和数据类型与 *accum* 相同。

异常:

- **TypeError** - 如果 *lr* 或者 *epsilon* 不是 float 类型。
- **TypeError** - 如果 *update_slots* 或者 *use_locking* 不是布尔值。
- **TypeError** - 如果 *var*、*accum*、*lr* 或 *grad* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *indices* 的数据类型不是 int32。
- **RuntimeError** - 如果 *var*、*accum* 和 *grad* 不支持数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.sparse_apply_adagrad_v2 = ops.SparseApplyAdagradV2(lr=1e-8, epsilon=1e-6)
...         self.var = Parameter(Tensor(np.array([[0.2]]).astype(np.float32)), name="var")
...         self.accum = Parameter(Tensor(np.array([[0.1]]).astype(np.float32)), name="accum")
...     def construct(self, grad, indices):
...         out = self.sparse_apply_adagrad_v2(self.var, self.accum, grad, indices)
...         return out
...
>>> net = Net()
>>> grad = Tensor(np.array([[0.7]]).astype(np.float32))
>>> indices = Tensor(np.array([0]), mindspore.int32)
>>> output = net(grad, indices)
>>> print(output)
```

(下页继续)

(续上页)

```
(Tensor(shape=[1, 1], dtype=Float32, value=
[[ 1.999999988e-01]]), Tensor(shape=[1, 1], dtype=Float32, value=
[[ 5.899999974e-01]]))
```

mindspore.ops.SparseApplyProximalAdagrad

class mindspore.ops.SparseApplyProximalAdagrad (*use_locking=False*)

根据 Proximal Adagrad 算法更新网络参数或者 Tensor。与 `mindspore.ops.ApplyProximalAdagrad` 相比，增加了一个索引 Tensor。

$$\begin{aligned} accum &+ grad * grad \\ prox\_v &= var - lr * grad * \frac{1}{\sqrt{accum}} \\ var &= \frac{sign(prox\_v)}{1+lr*l2} * \max(|prox\_v| - lr * l1, 0) \end{aligned}$$

var、*accum* 和 *grad* 的输入遵循隐式类型转换规则，使数据类型一致。如果它们具有不同的数据类型，则较低精度的数据类型将转换为相对最高精度的数据类型。

参数：

- **use_locking** (bool) - 如果为 True，则将保护 *var* 和 *accum* 参数不被更新。默认值：False。

输入：

- **var** (Union[Parameter, Tensor]) - 公式中的“var”。数据类型必须为 float16 或 float32。shape 为 (N,*), 其中 * 表示任何附加维度。
- **accum** (Union[Parameter, Tensor]) - 公式中的“accum”，与 *var* 的 shape 相同。
- **lr** (Union[Number, Tensor]) - 学习率，必须为 float 或为 Tensor，其数据类型为 float16 或 float32。必须大于零。
- **l1** (Union[Number, Tensor]) - l1 正则化，必须为 float 或为 Tensor，其数据类型为 float16 或 float32。必须大于等于零。
- **l2** (Union[Number, Tensor]) - l2 正则化，必须为 float 或为 Tensor，其数据类型为 float16 或 float32。必须大于等于零。
- **grad** (Tensor) - 梯度。如果 *var* 的 shape 大于 1，那么 `grad.shape[1:] = var.shape[1:]`。
- **indices** (Tensor) - *var* 和 *accum* 第一维度中的索引。如果 *indices* 中存在重复项，则无意义。数据类型必须是 int32、int64 和 `indices.shape[0] = grad.shape[0]`。

输出：

两个 Tensor 组成的 tuple，更新后的参数或者 Tensor。

- **var** (Tensor) - shape 和数据类型与输入 *var* 相同。
- **accum** (Tensor) - shape 和数据类型与输入 *accum* 相同。

异常：

- **TypeError** - 如果 *use_locking* 不是 bool。
- **TypeError** - 如果 *var*、*accum*、*lr*、*l1*、*l2* 或 *grad* 的数据类型既不是 float16 也不是 float32。
- **TypeError** - 如果 *indices* 的数据类型既不是 int32 也不是 int64。
- **ValueError** - 如果 *lr* ≤ 0 或者 *l1* < 0 或者 *l2* < 0。
- **RuntimeError** - 如果不支持参数的 *var*、*accum* 和 *grad* 数据类型转换。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.sparse_apply_proximal_adagrad = ops.SparseApplyProximalAdagrad()
...         self.var = Parameter(Tensor(np.array([[4.1, 7.2], [1.1, 3.0]], np.float32)),
... name="var")
...         self.accum = Parameter(Tensor(np.array([[0, 0], [0, 0]], np.float32)), name=
... name="accum")
...         self.lr = 1.0
...         self.l1 = 1.0
...         self.l2 = 0.0
...     def construct(self, grad, indices):
...         out = self.sparse_apply_proximal_adagrad(self.var, self.accum, self.lr, self.l1,
... self.l2, grad, indices)
...         return out
...
>>> net = Net()
>>> grad = Tensor(np.array([[1, 1], [1, 1]], np.float32))
>>> indices = Tensor(np.array([0, 1], np.int32))
>>> output = net(grad, indices)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Float32, value=
[[ 2.09999990e+00,  5.19999981e+00],
[ 0.00000000e+00,  1.00000000e+00]], Tensor(shape=[2, 2], dtype=Float32, value=
[[ 1.00000000e+00,  1.00000000e+00],
[ 1.00000000e+00,  1.00000000e+00]]))
```

mindspore.ops.SGD

class mindspore.ops.SGD (dampening=0.0, weight_decay=0.0, nesterov=False)

计算随机梯度下降。动量为可选。

Nesterov 动量基于论文 [On the importance of initialization and momentum in deep learning](#) 里的公式。

说明: 如果参数没有分组, 优化器中的 *weight_decay* 将应用于名称中没有 'beta' 或 'gamma' 的网络参数。用户可以将参数分组来改变权重衰减的策略。当参数被分组时, 每个组可以设置 *weight_decay*。如果没有, 优化器中的 *weight_decay* 将被应用。有关更多详细信息, 请参阅: [mindspore.nn.SGD](#)。

参数:

- **dampening** (float) - 动量的抑制因子。默认值: 0.0。
- **weight_decay** (float) - 权重衰减系数 (L2 惩罚)。默认值: 0.0。
- **nesterov** (bool) - 是否启用 Nesterov 动量。默认值: False。

输入:

- **parameters** (Tensor) - 待更新的参数。数据类型为 float16 或 float32。

- **gradient** (Tensor) - 梯度，数据类型为 float16 或 float32。
- **learning_rate** (Tensor) - 学习率，是一个 Scalar 的 Tensor，数据类型为 float16 或 float32。例如 Tensor(0.1, mindspore.float32)。
- **accum** (Tensor) - 待更新的累加器（速度）。数据类型为 float16 或 float32。
- **momentum** (Tensor) - 动量，是一个 Scalar 的 Tensor，数据类型为 float16 或 float32。例如 Tensor(0.1, mindspore.float32)。
- **stat** (Tensor) - 待更新的状态，其 shape 与梯度相同，数据类型为 float16 或 float32。

输出：

Tensor，更新后的参数。

异常：

- **TypeError** - *dampening* 或 *weight_decay* 不是 float。
- **TypeError** - *nesterov* 不是 bool。
- **TypeError** - *parameters*、*gradient*、*learning_rate*、*accum*、*momentum* 或 *stat* 不是 Tensor。
- **TypeError** - *parameters*、*gradient*、*learning_rate*、*accum*、*momentum* 或 *stat* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> sgd = ops.SGD()
>>> parameters = Tensor(np.array([2, -0.5, 1.7, 4]), mindspore.float32)
>>> gradient = Tensor(np.array([1, -1, 0.5, 2]), mindspore.float32)
>>> learning_rate = Tensor(0.01, mindspore.float32)
>>> accum = Tensor(np.array([0.1, 0.3, -0.2, -0.1]), mindspore.float32)
>>> momentum = Tensor(0.1, mindspore.float32)
>>> stat = Tensor(np.array([1.5, -0.3, 0.2, -0.7]), mindspore.float32)
>>> output = sgd(parameters, gradient, learning_rate, accum, momentum, stat)
>>> print(output.asnumpy())
[1.99 -0.4903 1.695 3.9801]
```

mindspore.ops.SparseApplyFtrl

class mindspore.ops.SparseApplyFtrl(lr, l1, l2, lr_power, use_locking=False)

根据 FTRL-proximal 算法更新相关参数或者 Tensor。

更多详细信息请参见 `mindspore.nn.FTRL`。

参数：

- **lr** (float) - 学习率，必须为正值。
- **l1** (float) - l1 正则化，必须大于或等于零。
- **l2** (float) - l2 正则化，必须大于或等于零。
- **lr_power** (float) - 在训练期间控制降低学习率，必须小于或等于零。如果 **lr_power** 为零，则使用固定学习率。

- **use_locking** (bool, 可选) - 是否对参数更新加锁保护。默认值: `False`。

输入:

- **var** (Union[Parameter, Tensor]) - 要更新的权重。数据类型必须为 `float16` 或 `float32`。shape 为 $(N,*)$ ，其中 $*$ 表示任意数量的附加维度。
- **accum** (Union[Parameter, Tensor]) - 要更新的累数值，shape 必须与 `var` 相同。
- **linear** (Union[Parameter, Tensor]) - 要更新的线性系数，shape 必须与 `var` 相同。
- **grad** (Tensor) - 梯度，为一个 Tensor。需要满足：如果 `var.shape > 1`，则 `grad.shape[1:] = var.shape[1:]`。
- **indices** (Tensor) - `var` 和 `accum` 第一维度的索引向量，数据类型为 `int32` 或 `int64`，且需要保证 `indices.shape[0] = grad.shape[0]`。

输出:

- **var** (Tensor) - shape 和数据类型与输入 `var` 相同。
- **accum** (Tensor) - shape 和数据类型与输入 `accum` 相同。
- **linear** (Tensor) - shape 和数据类型与输入 `linear` 相同。

异常:

- **TypeError** - 如果 `lr`、`l1`、`l2` 或 `lr_power` 不是 `float` 类型。
- **TypeError** - 如果 `use_locking` 不是 `bool`。
- **TypeError** - 如果 `var`、`grad`、`linear` 或者 `grad` 的数据类型既不是 `float16` 也不是 `float32`。
- **TypeError** - 如果 `indices` 不是 `int32` 也不是 `int64` 类型。
- **RuntimeError** - 如果 `var`、`grad`、`linear` 或者 `grad` 不支持数据类型转换。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, Parameter, ops
>>> class SparseApplyFtrlNet(nn.Cell):
...     def __init__(self):
...         super(SparseApplyFtrlNet, self).__init__()
...         self.sparse_apply_ftrl = ops.SparseApplyFtrl(lr=0.01, l1=0.0, l2=0.0, lr_power=-
    ↪0.5)
...         self.var = Parameter(Tensor(np.array([[0.2]]).astype(np.float32)), name="var")
...         self.accum = Parameter(Tensor(np.array([[0.1]]).astype(np.float32)), name="accum
    ↪")
...         self.linear = Parameter(Tensor(np.array([[0.6]]).astype(np.float32)), name=
    ↪"linear")
...     def construct(self, grad, indices):
...         out = self.sparse_apply_ftrl(self.var, self.accum, self.linear, grad, indices)
...         return out
...
>>> net = SparseApplyFtrlNet()
>>> grad = Tensor(np.array([[0.7]]).astype(np.float32))
>>> indices = Tensor(np.ones([1]), mindspore.int32)
>>> output = net(grad, indices)
>>> print(output)
```

(下页继续)

(续上页)

```
(Tensor(shape=[1, 1], dtype=Float32, value=
[[2.00000003e-01]]), Tensor(shape=[1, 1], dtype=Float32, value=
[[1.00000001e-01]]), Tensor(shape=[1, 1], dtype=Float32, value=
[[6.00000024e-01]]))
```

mindspore.ops.SparseApplyFtrlV2

class mindspore.ops.SparseApplyFtrlV2 (lr, l1, l2, l2_shrinkage, lr_power, use_locking=False)
mindspore.ops.SparseApplyFtrlV2 从 2.0 版本开始已被弃用，并将在未来版本中被移除，建议使用 mindspore.ops.SparseApplyFtrl 代替。
支持平台：
Deprecated

20.2.5 距离函数

接口名	概述	支持平台
<code>mindspore.ops.Cdist</code>	更多详情请查看: <code>mindspore.ops.cdist()</code> 。	Ascend GPU CPU
<code>mindspore.ops.EditDistance</code>	计算 Levenshtein 编辑距离。	Ascend CPU
<code>mindspore.ops.LpNorm</code>	返回输入矩阵或向量的 p-范数。	Ascend GPU CPU
<code>mindspore.ops.Pdist</code>	计算输入中每对行向量之间的 p-范数距离。	GPU CPU

mindspore.ops.Cdist

class mindspore.ops.Cdist (p=2.0)

```
prim = ops.Cdist(p)
out = prim(x1, x2)
```

等价于

```
ops.cdist(x1, x2, p)
```

更多详情请查看: `mindspore.ops.cdist()`。

mindspore.ops.EditDistance

class mindspore.ops.EditDistance (normalize=True)
计算 Levenshtein 编辑距离。它用于测量两个序列的相似性。输入是可变长度的序列，由 SpaseTensors (hypothesis_indices, hypothesis_values, hypothesis_shape) 和 (truth_indices, truth_values, truth_shape) 提供。

$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} \text{lev}_{a,b}(i-1,j) + 1 \\ \text{lev}_{a,b}(i,j-1) + 1 \\ \text{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

其中 *a* 表示预测值，*b* 表示真实值。为了便于理解，这里的 *i* 和 *j* 可以被视为 *a* 和 *b* 的长度。

警告:

- 如果输入 *truth_indices* 或者 *hypothesis_indices* 不是有序的, 可能会导致计算结果不符合预期, 建议调用该接口之前确保输入的稀疏张量 *truth_indices* 和 *hypothesis_indices* 都是升序排列的。

参数:

- **normalize** (bool) - 如果为 True, 则编辑距离将按真实值长度标准化。默认值: True。

输入:

- **hypothesis_indices** (Tensor) - 预测列表的索引。类型为 Tensor, 数据类型为 int64, 其 shape 为 (N, R) 。
- **hypothesis_values** (Tensor) - 预测列表的值。类型为 Tensor, 必须是长度为 N 的一维向量。
- **hypothesis_shape** (Tensor) - 预测列表的 shape。类型为 Tensor, 必须是长度为 R 的向量, 数据类型为 int64。只能是常量。
- **truth_indices** (Tensor) - 真实列表的索引。类型为 Tensor, 数据类型为 int64, 其 shape 为 (M, R) 。
- **truth_values** (Tensor) - 真实列表的值。类型为 Tensor, 必须是长度为 M 的一维向量。
- **truth_shape** (Tensor) - 真实列表的 shape。类型为 Tensor, 必须是长度为 R 的向量, 数据类型为 int64。只能是常量。

输出:

Tensor, 其秩为 $R-1$, 数据类型为 float32。

异常:

- **TypeError** - 如果 *normalize* 不是 bool。

支持平台:

Ascend CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> class EditDistance(nn.Cell):
...     def __init__(self, hypothesis_shape, truth_shape, normalize=True):
...         super(EditDistance, self).__init__()
...         self.edit_distance = ops.EditDistance(normalize)
...         self.hypothesis_shape = hypothesis_shape
...         self.truth_shape = truth_shape
...
...     def construct(self, hypothesis_indices, hypothesis_values, truth_indices, truth_
...         values):
...         return self.edit_distance(hypothesis_indices, hypothesis_values, self.hypothesis_
...         shape,
...                                     truth_indices, truth_values, self.truth_shape)
...
>>> hypothesis_indices = Tensor(np.array([[0, 0, 0], [1, 0, 1], [1, 1, 1]]).astype(np.int64))
>>> hypothesis_values = Tensor(np.array([1, 2, 3]).astype(np.float32))
>>> hypothesis_shape = Tensor(np.array([1, 1, 2]).astype(np.int64))
>>> truth_indices = Tensor(np.array([[0, 1, 0], [0, 0, 1], [1, 1, 0], [1, 0, 1]]).astype(np.
...         int64))
```

(下页继续)

(续上页)

```
>>> truth_values = Tensor(np.array([1, 3, 2, 1]).astype(np.float32))
>>> truth_shape = Tensor(np.array([2, 2, 2]).astype(np.int64))
>>> edit_distance = EditDistance(hypothesis_shape, truth_shape)
>>> output = edit_distance(hypothesis_indices, hypothesis_values, truth_indices, truth_
↪values)
>>> print(output)
[[1. 1.]
 [1. 1.]]
```

mindspore.ops.LpNorm

class mindspore.ops.LpNorm (axis, p=2, keep_dims=False, epsilon=1e-12)

返回输入矩阵或向量的 p-范数。

$$output = \|input\|_p = \left(\sum_{i=1}^n |input|^p \right)^{1/p}$$

参数：

- **axis** (int,list,tuple) - 指定计算范数的维度。
- **p** (int, 可选) - 范数的阶。默认值：2。
- **keep_dims** (bool, 可选) - 输出 Tensor 是否保留原有的维度。默认值：False。
- **epsilon** (float, 可选) - 范数下界，当计算的范数小于此值时，用 *epsilon* 替换该结果。默认值：1e-12。

输入：

- **input** (Tensor) - 输入 Tensor，数据类型为 float16 或 float32。

输出：

Tensor，数据类型与 *input* 一致，其 shape 由 *axis* 决定。如果输入 shape 为 (2,3,4)，*axis* 为 [0,1]，则输出 shape 为 (4,)。

异常：

- **TypeError** - 若 *input* 不是 Tensor。
- **TypeError** - 若 *input* 的数据类型不是 float16 或 float32。
- **TypeError** - 若 *p* 不是 int。
- **TypeError** - 若 *axis* 不是 int、list 或 tuple。
- **TypeError** - 若 *axis* 是 list 或 tuple，但含有非 int 元素。
- **TypeError** - 若 *keep_dims* 不是 bool。
- **ValueError** - 若 *axis* 的元素不在 $[-r, r)$ 范围内，其中 *r* 为 *input* 的秩。
- **ValueError** - 若 *axis* shape 长度大于 *input* shape 长度。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[1.0, 2.0], [3.0, 4.0]], [[5.0, 6.0], [7.0, 8.0]])).
    ↳ astype(np.float32))
>>> op = ops.LpNorm(axis=[0, 1], p=2, keep_dims=False)
>>> output = op(input_x)
>>> print(output)
[ 9.165152 10.954452]
```

mindspore.ops.Pdist

class mindspore.ops.Pdist (p=2.0)
计算输入中每对行向量之间的 p-范数距离。
更多细节请参考 `mindspore.ops.pdist()`。

说明：Pdist 算子由于涉及求幂次方运算，在 float16 类型输入场景下，很容易会因为数值溢出而产生 inf/nan 的计算结果，建议采用 float32 类型输入。

- 参数：
- **p** (float, 可选) - 范数距离的阶， $p \in [0, \infty)$ 。默认值：2.0。
- 输入：
- **x** (Tensor) - 输入 Tensor。类型：float16、float32 或 float64。
- 输出：
- Tensor，类型与 *x* 一致。
- 支持平台：
- GPU CPU
- 样例：

```
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> x = Tensor(np.array([[1.0, 1.0], [2.0, 2.0], [3.0, 3.0]]).astype(np.float32))
>>> op = ops.Pdist(p=2.0)
>>> y = op(x)
>>> print(y)
[1.4142135 2.828427 1.4142135]
```

20.2.6 采样算子

接口名	概述	支持平台
<code>mindspore.ops.ComputeAccidentalHits</code>	计算与目标类完全匹配的抽样样本的位置 id。	Ascend
<code>mindspore.ops.LogUniformCandidateSampler</code>	使用 log-uniform(Zipfian) 分布对一组类别进行采样。	Ascend CPU

下页继续

表 7 - 续上页

<code>mindspore.ops. UniformCandidateSampler</code>	使用均匀分布对一组类别进行采样。	Ascend GPU CPU

mindspore.ops.ComputeAccidentalHits

class mindspore.ops.ComputeAccidentalHits (num_true=1)

计算与目标类完全匹配的抽样样本的位置 id。

当目标类与抽样类匹配时，我们称之为“accidental hit”。accidental hit 的计算结果包含 indices、ids、weights 三部分，其中 index 代表目标类中的行号，id 代表候选抽样中的位置，weight 为 float 类型中的最大值。

参数：

- **num_true** (int, 可选) - 每个训练样本的目标类数。默认值：1。

输入：

- **true_classes** (Tensor) - 目标类。数据类型为 int64，shape 为 (batch_size, num_true)。
- **sampled_candidates** (Tensor) - 算子的候选采样结果，代表训练样本的类别。其数据类型为 int64，shape 为 (num_sampled,)。

输出：

3 个 Tensor 组成的元组。

- **indices** (Tensor) - shape 为 (num_accidental_hits,) 的 Tensor，数据类型为 int32。
- **ids** (Tensor) - shape 为 (num_accidental_hits,) 的 Tensor，数据类型为 int64。
- **weights** (Tensor) - shape 为 (num_accidental_hits,) 的 Tensor，类型为 float32。

异常：

- **TypeError** - num_true 的数据类型不为 int。
- **TypeError** - true_classes 或 sampled_candidates 不是 Tensor。
- **TypeError** - true_classes 或 sampled_candidates 的数据类型既不是 int32 也不是 int64。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> true_classes = np.array([[1, 2], [0, 4], [3, 3]])
>>> sampled_candidates = np.array([0, 1, 2, 3, 4])
>>> sampler = ops.ComputeAccidentalHits(2)
>>> indices, ids, weights = sampler(Tensor(true_classes), Tensor(sampled_candidates))
>>> print(indices, ids, weights)
[0 0 1 1 2 2]
[1 2 0 4 3 3]
[-3.4028235e+38 -3.4028235e+38 -3.4028235e+38 -3.4028235e+38 -3.4028235e+38 -3.4028235e+38]
```

mindspore.ops.LogUniformCandidateSampler

class mindspore.ops.LogUniformCandidateSampler (num_true=1, num_sampled=5, unique=True, range_max=5, seed=0)

使用 log-uniform(Zipfian) 分布对一组类别进行采样。

该操作从整数范围 $[0, range\_max)$ 中随机采样一个采样类 (*sampled_candidates*) 的 Tensor。

更多参考详见 `mindspore.ops.log_uniform_candidate_sampler()`。

警告：Ascend 后端不支持随机数重现功能，*seed* 参数不起作用。

参数：

- **num_true** (int, 可选) - 每个训练样本的目标类数。默认值：1。
- **num_sampled** (int, 可选) - 随机采样的类数。默认值：5。
- **unique** (bool, 可选) - 确认批处理中的所有采样类是否都是唯一的。如果 *unique* 为 True，则批处理中的所有采样类都唯一。默认值：True。
- **range_max** (int, 可选) - 可能的类数。当 *unique* 为 True 时，*range_max* 必须大于或等于 *num_sampled*。默认值：5。
- **seed** (int, 可选) - 随机种子，必须是非负。默认值：0。

输入：

- **true_classes** (Tensor) - 目标类，其数据类型为 int64，shape 为 $(batch\_size, num\_true)$ 。

输出：

3 个 Tensor 组成的元组。

- **sampled_candidates** (Tensor) - shape 为 $(num\_sampled,)$ 且数据类型与 *true_classes* 相同的 Tensor。
- **true_expected_count** (Tensor) - shape 与 *true_classes* 相同且数据类型为 float32 的 Tensor。
- **sampled_expected_count** (Tensor) - shape 与 *sampled_candidates* 相同且数据类型为 float32 的 Tensor。

支持平台：

Ascend CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> sampler = ops.LogUniformCandidateSampler(2, 5, True, 5)
>>> output1, output2, output3 = sampler(Tensor(np.array([[1, 7], [0, 4], [3, 3]])))
>>> print(output1, output2, output3)
[3 2 0 4 1]
[[0.92312991 0.49336370]
 [0.99248987 0.65806371]
 [0.73553443 0.73553443]]
[0.73553443 0.82625800 0.99248987 0.65806371 0.92312991]
```


`mindspore.ops.UniformCandidateSampler`

class `mindspore.ops.UniformCandidateSampler` (*num_true, num_sampled, unique, range_max, seed=0, remove_accidental_hits=False*)

使用均匀分布对一组类别进行采样。

此函数使用均匀分布，从 $[0, \text{range\_max}-1]$ 中采样一组类 (`sampled_candidates`)。如果 `unique` 为 `True`，则候选采样没有重复；如果 `unique` 为 `False`，则有重复。

更多参考详见 `mindspore.ops.uniform_candidate_sampler()`。

警告：

- Ascend 后端不支持随机数重现功能，`seed` 参数不起作用。
- Ascend 后端暂不支持动态 `shape` 场景。

参数：

- **num_true** (int) - 每个训练样本的目标类数。
- **num_sampled** (int) - 随机采样的类数。`sampled_candidates` 的 `shape` 将为 `num_sampled`。如果 `unique` 为 `True`，则 `num_sampled` 必须小于或等于 `range_max`。
- **unique** (bool) - 表示一个 `batch` 中的所有采样类是否唯一。
- **range_max** (int) - 可能的类数，该值必须是非负的。
- **seed** (int, 可选) - 随机种子，该值必须是非负的。如果 `seed` 的值为 0，则 `seed` 的值将被随机生成的值替换。默认值：0。
- **remove_accidental_hits** (bool, 可选) - 表示是否移除 accidental hit。accidental hit 表示其中一个 `true_classes` 目标类匹配 `sampled_candidates` 采样类之一，设置为 `True` 表示移除等于目标类的采样类。默认值：`False`。

输入：

- **true_classes** (Tensor) - 输入 Tensor，目标类，其 `shape` 为 $(\text{batch\_size}, \text{num\_true})$ 。其元素值范围需要在 $[0, \text{range\_max})$ 。

输出：

- **sampled_candidates** (Tensor) - 候选采样与目标类之间不存在联系，其 `shape` 为 $(\text{num\_sampled},)$ 。
- **true_expected_count** (Tensor) - 在每组目标类的采样分布下的预期计数。Shape 为 $(\text{batch\_size}, \text{num\_true})$ 。
- **sampled_expected_count** (Tensor) - 每个候选采样分布下的预期计数。Shape 为 $(\text{num\_sampled},)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> sampler = ops.UniformCandidateSampler(1, 3, False, 4, 1)
>>> output1, output2, output3 = sampler(Tensor(np.array([[1], [3], [4], [6], [3]], dtype=np.
    ↳int64)))
>>> print(output1.shape)
(3,)
>>> print(output2.shape)
```

(下页继续)

(续上页)

```
(5, 1)
>>> print (output3.shape)
(3,)
```

20.2.7 图像处理

接口名	概述	支持平台
<code>mindspore.ops.BoundingBoxDecode</code>	解码边界框位置信息。	Ascend GPU CPU
<code>mindspore.ops.BoundingBoxEncode</code>	编码边界框位置信息。	Ascend GPU CPU
<code>mindspore.ops.CheckValid</code>	检查边界框。	Ascend GPU CPU
<code>mindspore.ops.CropAndResize</code>	从输入图像 Tensor 中提取切片并调整其大小。	Ascend GPU CPU
<code>mindspore.ops.ExtractVolumePatches</code>	<code>ops.ExtractVolumePatches</code> 从 2.3 版本开始已被弃用，并将在未来版本中被移除。	弃用
<code>mindspore.ops.IOU</code>	计算矩形的 IOU，即真实区域和预测区域的交并比。	Ascend GPU CPU
<code>mindspore.ops.L2Normalize</code>	L2 范数归一化算子。	Ascend GPU CPU
<code>mindspore.ops.NMSWithMask</code>	非极大值抑制算法（NMS, Non-maximum Suppression）。	Ascend GPU CPU
<code>mindspore.ops.ResizeBilinearV2</code>	使用双线性插值调整图像尺寸到指定的大小。	Ascend GPU CPU
<code>mindspore.ops.ROIALign</code>	感兴趣区域对齐 (RoI Align) 运算。	Ascend GPU CPU

mindspore.ops.BoundingBoxDecode

```
class mindspore.ops.BoundingBoxDecode (max_shape, means=(0.0, 0.0, 0.0, 0.0), stds=(1.0, 1.0, 1.0, 1.0),
                                         wh_ratio_clip=0.016)
```

解码边界框位置信息。

算子的功能是计算偏移量，此算子将偏移量转换为 Bbox，用于在后续图像中标记目标等信息。

参数：

- **max_shape** (tuple) - 解码框计算的上限值。
- **means** (tuple, 可选) - 计算 *deltas* 的均值。默认值：(0.0, 0.0, 0.0, 0.0)。
- **stds** (tuple, 可选) - 计算 *deltas* 的标准差。默认值：(1.0, 1.0, 1.0, 1.0)。
- **wh_ratio_clip** (float, 可选) - 解码框计算的宽高比限制。默认值：0.016。

输入：

- **anchor_box** (Tensor) - 锚框。锚框的 shape 必须为 (n, 4)。
- **deltas** (Tensor) - 框的增量。它的 shape 与 *anchor_box* 相同。

输出：

Tensor，解码框。它的数据类型和 shape 与 *anchor_box* 相同。

- 异常：
- **TypeError** - 如果 *means*、*stds* 或 *max_shape* 不是 tuple。
 - **TypeError** - 如果 *wh_ratio_clip* 不是 float。
 - **TypeError** - 如果 *anchor_box* 或 *deltas* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> anchor_box = Tensor([[4, 1, 2, 1], [2, 2, 2, 3]], mindspore.float32)
>>> deltas = Tensor([[3, 1, 2, 2], [1, 2, 1, 4]], mindspore.float32)
>>> boundingbox_decode = ops.BoundingBoxDecode(means=(0.0, 0.0, 0.0, 0.0), stds=(1.0, 1.0, 1.0, 1.0),
...                                             max_shape=(768, 1280), wh_ratio_clip=0.016)
>>> output = boundingbox_decode(anchor_box, deltas)
>>> print(output)
[[ 4.194528  0.          0.          5.194528]
 [ 2.1408591  0.          3.8591409  60.598152  ]]
```

mindspore.ops.BoundingBoxEncode

class mindspore.ops.BoundingBoxEncode (*means=(0.0, 0.0, 0.0, 0.0), stds=(1.0, 1.0, 1.0, 1.0)*)

编码边界框位置信息。

算子的功能是计算预测边界框和真实边界框之间的偏移，并将此偏移作为损失变量。

参数:

- **means** (tuple) - 计算编码边界框的均值。默认值: (0.0, 0.0, 0.0, 0.0)。
- **stds** (tuple) - 计算增量的标准偏差。默认值: (1.0, 1.0, 1.0, 1.0)。

输入:

- **anchor_box** (Tensor) - 锚框。锚框的 **shape** 必须为 $(n, 4)$ 。
- **groundtruth_box** (Tensor) - 真实边界框。它的 **shape** 与锚框相同。

输出:

Tensor, 编码边界框。数据类型和 **shape** 与输入 *anchor_box* 相同。

异常:

- **TypeError** - 如果 *means* 或 *stds* 不是 tuple。
- **TypeError** - 如果 *anchor_box* 或 *groundtruth_box* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> anchor_box = Tensor([[2, 2, 2, 3], [2, 2, 2, 3]], mindspore.float32)
>>> groundtruth_box = Tensor([[1, 2, 1, 4], [1, 2, 1, 4]], mindspore.float32)
>>> boundingbox_encode = ops.BoundingBoxEncode(means=(0.0, 0.0, 0.0, 0.0), stds=(1.0, 1.0, 1.0, 1.0))
>>> output = boundingbox_encode(anchor_box, groundtruth_box)
>>> print(output)
[[ -1.   0.25  0.   0.40551758]
 [ -1.   0.25  0.   0.40551758]]
```

mindspore.ops.CheckValid

class mindspore.ops.CheckValid

检查边界框。

检查由 *bboxes* 指定的一些边界框是否是有效的。如果边界框在由 *img metas* 确定的边界内部则返回 True，否则返回 False。

输入：

- **bboxes** (Tensor) - shape 大小为 $(N, 4)$ 。 N 表示边界框的数量，4 表示 $x0$ 、 $y0$ 、 $x1$ 、 $y1$ 。数据类型必须是 float16 或 float32。
- **img metas** (Tensor) - 原始图片的信息 (,)，指定有效边界为 (, ,)。数据类型必须是 float16 或 float32。

输出：

Tensor, shape 为 $(N,)$ ，类型为 bool，指出边界框是否在图片内。True 表示在，False 表示不在。

异常：

- **TypeError** - 如果 *bboxes* 或者 *img metas* 不是 Tensor。
- **TypeError** - 如果 *bboxes* 或者 *img metas* 的数据类型既不是 float16，也不是 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.check_valid = ops.CheckValid()
...     def construct(self, x, y):
...         valid_result = self.check_valid(x, y)
...         return valid_result
...
>>> bboxes = Tensor(np.linspace(0, 6, 12).reshape(3, 4), mindspore.float32)
>>> img_metas = Tensor(np.array([2, 1, 3]), mindspore.float32)
>>> net = Net()
>>> output = net(bboxes, img_metas)
>>> print(output)
[ True False False]
```

mindspore.ops.CropAndResize

class mindspore.ops.CropAndResize (method='bilinear', extrapolation_value=0.0)

从输入图像 Tensor 中提取切片并调整其大小。

说明：如果输出 shape 依赖于 *crop_size* 的值，则 *crop_size* 必须为常量。目前该算子仅在 *method* 为“bilinear”支持反向，其他方法将直接返回 0。

参数：

- **method** (str, 可选) - 指定调整大小的采样方法, 为可选字符串。提供的方法有: "bilinear"、"nearest" 或 "bilinear_v2"。默认值: "bilinear"。
 - "nearest": 最近邻插值。每个输出像素的值为最近的输入像素的值。这种方法简单快速, 但可能导致块状或像素化的输出。
 - "bilinear": 双线性插值。每个输出像素是最接近的四个输入像素的加权平均值, 使用双线性插值计算。与最近邻插值相比, 此方法产生更平滑的结果。
 - "bilinear_v2": 优化后的双线性插值算法, 在某些情况下可能会产生更好的结果 (更高的精度和速度)。
- **extrapolation_value** (float, 可选) - 外插值, 数据类型为 float。默认值: 0.0。

输入:

- **x** (Tensor) - 输入为四维的 Tensor, 其 shape 必须是 $(batch, image\_height, image\_width, depth)$ 。支持的数据类型: int8、int16、int32、int64、float16、float32、float64、uint8、uint16。
- **boxes** (Tensor) - 二维 Tensor, 其 shape 为 $(num\_boxes, 4)$ 。第 i 行表示 $box\_index[i]$ 图像区域的坐标, 并且坐标 $[y1, x1, y2, x2]$ 是归一化后的值。归一化后的坐标值 y , 映射到图像 $y * (image\_height - 1)$ 处, 因此归一化后的图像高度范围为 $[0, 1]$, 映射到实际图像高度范围为 $[0, image\_height - 1]$ 。我们允许 $y1 > y2$, 在这种情况下, 视为原始图像的上下翻转变换。宽度尺寸的处理类似。坐标取值允许在 $[0, 1]$ 范围之外, 在这种情况下, 我们使用 *extrapolation_value* 外插值进行补齐。支持的数据类型: float32。
- **box_index** (Tensor) - *boxes* 的索引, 其 shape 为 (num_boxes) 的一维 Tensor, 数据类型为 int32, 取值范围为 $[0, batch)$ 。box_index[i] 的值表示第 i 个框的图像的值。
- **crop_size** (Tuple[int]) - 两个 int32 元素组成的 tuple: $(crop\_height, crop\_width)$ 。只能是常量。所有裁剪后的图像大小都将调整为此大小, 且不保留图像内容的宽高比, 裁剪高度和裁剪宽度都需要为正。

输出:

四维 Tensor, 其 shape 为 $(num\_boxes, crop\_height, crop\_width, depth)$, 数据类型类型为 float32。

异常:

- **TypeError** - *x*、*boxes* 或 *box_index* 不是 Tensor。
- **TypeError** - *crop_size* 不是 int32 类型的 tuple, 或 *crop_size* 的长度不是 2。
- **TypeError** - *boxes* 的数据类型不是 float, 或者, *box_index* 的数据类型不是 int32。
- **TypeError** - *method* 不是字符串。
- **TypeError** - *extrapolation_value* 不是浮点值。
- **ValueError** - *x* 的维度不是四维。
- **ValueError** - *boxes* 的纬度不是二维。
- **ValueError** - *boxes* 的第二维不是 4。
- **ValueError** - *box_index* 的维度不是一维。
- **ValueError** - *box_index* 的第一维与 *boxes* 的第一维不相等。
- **ValueError** - *box_index* 存在元素不在 $[0, batch)$ 的范围内。
- **ValueError** - *crop_size* 的数据不是正整数。
- **ValueError** - *method* 不是 "bilinear"、"nearest"、"bilinear_v2" 之一。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import nn, ops, Tensor
>>> class CropAndResizeNet(nn.Cell):
...     def __init__(self, crop_size):
...         super(CropAndResizeNet, self).__init__()
...         self.crop_and_resize = ops.CropAndResize()
...         self.crop_size = crop_size
...
...     def construct(self, x, boxes, box_index):
...         return self.crop_and_resize(x, boxes, box_index, self.crop_size)
...
>>> BATCH_SIZE = 1
>>> NUM_BOXES = 5
>>> IMAGE_HEIGHT = 256
>>> IMAGE_WIDTH = 256
>>> CHANNELS = 3
>>> image = np.random.normal(size=[BATCH_SIZE, IMAGE_HEIGHT, IMAGE_WIDTH, CHANNELS]).
...         astype(np.float32)
>>> boxes = np.random.uniform(size=[NUM_BOXES, 4]).astype(np.float32)
>>> box_index = np.random.uniform(size=[NUM_BOXES], low=0, high=BATCH_SIZE).astype(np.int32)
>>> crop_size = (24, 24)
>>> crop_and_resize = CropAndResizeNet(crop_size=crop_size)
>>> output = crop_and_resize(Tensor(image), Tensor(boxes), Tensor(box_index))
>>> print(output.shape)
(5, 24, 24, 3)
```

mindspore.ops.ExtractVolumePatches

class mindspore.ops.**ExtractVolumePatches** (*kernel_size, strides, padding*)

ops.ExtractVolumePatches 从 2.3 版本开始已被弃用，并将在未来版本中被移除。

支持平台:

Deprecated

mindspore.ops.IOU

class mindspore.ops.**IOU** (*mode='iou'*)

计算矩形的 IOU，即真实区域和预测区域的交并比。

根据真实区域和预测区域计算 IOU(intersection over union) 或 IOF(intersection over foreground)。

更多参考详见 *mindspore.ops.iou()*。

参数:

- **mode** (str) - 指定计算方法，现支持 'iou' (intersection over union) 或 'iof' (intersection over foreground) 模式。默认值: 'iou'。

输入:

- **anchor_boxes** (Tensor) - 预测区域，shape 为 $(N, 4)$ 的 Tensor。“N”表示预测区域的数量，“4”表示“x0”、“y0”、“x1”和“y1”。数据类型为 float16 或 float32。
- **gt_boxes** (Tensor) - 真实区域，shape 为 $(M, 4)$ 的 Tensor。“M”表示真实区域的数量，“4”表示“x0”、“y0”、“x1”和“y1”。数据类型为 float16 或 float32。

输出:

IOU 值的 Tensor, shape 为 (M, N) 的 Tensor, 数据类型与 *anchor_boxes* 的相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> iou = ops.IOU(mode='iou')
>>> anchor_boxes = Tensor(np.random.randint(1.0, 5.0, [3, 4]), mindspore.float16)
>>> gt_boxes = Tensor(np.random.randint(1.0, 5.0, [3, 4]), mindspore.float16)
>>> output = iou(anchor_boxes, gt_boxes)
>>> print(output.shape)
(3, 3)
```

mindspore.ops.L2Normalize

class mindspore.ops.L2Normalize (*axis=0, epsilon=1e-4*)

L2 范数归一化算子。

该算子将对输入 x 在给定 *axis* 上的元素进行归一化。函数定义如下:

$$\text{output} = \frac{x}{\sqrt{\max(\sum_i |x_i|^2, \epsilon)}}$$

其中 ϵ 表示 *epsilon*, $\sum_i |x_i|^2$ 表示计算输入 x 在给定 *axis* 上元素的平方和。

说明: 在 Ascend 上, 暂时不支持 float64 数据类型。

参数:

- **axis** (Union[list(int), tuple(int), int], 可选) - 指定计算 L2 范数的轴。默认值: 0。
- **epsilon** (float, 可选) - 为了数值稳定性而引入的很小的浮点数。默认值: $1e-4$ 。

输入:

- **x** (Tensor) - 计算归一化的输入。shape 为 $(N, *)$, 其中 $*$ 表示任意的附加维度数。数据类型必须为 float16、float32 和 float64。

输出:

Tensor, shape 和数据类型与 x 的相同。

异常:

- **TypeError** - *axis* 不是 list、tuple 或 int。
- **TypeError** - *epsilon* 不是 float。

- **TypeError** - x 不是 Tensor。
- **TypeError** - x 的数据类型不是 float16、float32 或 float64。
- **ValueError** - x 的维度不大于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> l2_normalize = ops.L2Normalize()
>>> x = Tensor(np.random.randint(-256, 256, (2, 3, 4)), mindspore.float32)
>>> output = l2_normalize(x)
>>> print(output.shape)
(2, 3, 4)
```

mindspore.ops.NMSWithMask

class mindspore.ops.NMSWithMask (*iou_threshold=0.5*)

非极大值抑制算法（NMS, Non-maximum Suppression）。当在计算机视觉领域中进行目标检测时，目标检测算法将生成多个边界框，并计算分数最高的边界框与其他边界框的交并比（IOU），然后根据设定的阈值删除框。在 Ascend 平台上，边界框的分数将被忽略，仅根据框之间的 IOU 来选择框。这意味着如果要删除分数较低的框，则需要提前按分数对输入框进行降序排序。IOU 的计算如下：

$$\text{IOU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

警告： 一次最多支持 2864 个输入框。

参数：

- **iou_threshold** (float) - 指定删除框的 IOU 的阈值。默认值：0.5。

输入：

- **bboxes** (Tensor) - 边界框，shape: ($N, 5$)， N 为边界框的数量。每个边界框包含 5 个值，前 4 个值为边界框的坐标 (x_0 、 y_0 、 x_1 、 y_1)，代表左上角和右下角的点。最后一个值为边界框的分数。数据类型支持 float16 或 float32。

输出：

tuple[Tensor]，包含三个 Tensor：output_boxes、output_idx 和 selected_mask。

- **output_boxes** (Tensor) - shape: ($N, 5$)。在 GPU 和 CPU 平台上，它是一个边界框的排序列表，按分数对输入 *bboxes* 进行降序排序。在 Ascend 平台上，它与输入 *bboxes* 相同。
- **output_idx** (Tensor) - shape: ($N, \text{}$)。output_boxes 的索引列表。
- **selected_mask** (Tensor) - shape: ($N, \text{}$)。输出边界框的掩码列表。在 *output_boxes* 上应用此掩码以获取非极大值抑制算法（NMS）计算后的边界框，或在 *output_idx* 上应用此掩码以获取边界框索引。

异常：

- **ValueError** - *iou_threshold* 不是 float。

- **ValueError** - 输入 Tensor 的第一个维度小于或等于 0。
- **TypeError** - *bboxes* 的数据类型非 float16 或 float32。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bbox = np.array([[100.0, 100.0, 50.0, 68.0, 0.63], [150.0, 75.0, 165.0, 115.0, 0.55],
...                  [12.0, 190.0, 288.0, 200.0, 0.9], [28.0, 130.0, 106.0, 172.0, 0.3]])
>>> bbox[:, 2] += bbox[:, 0]
>>> bbox[:, 3] += bbox[:, 1]
>>> inputs = Tensor(bbox, mindspore.float32)
>>> nms = ops.NMSWithMask(0.1)
>>> output_boxes, indices, mask = nms(inputs)
>>> indices_np = indices.asnumpy()
>>> print(indices_np[mask.asnumpy()])
[0 1 2]
```

mindspore.ops.ResizeBilinearV2

class mindspore.ops.ResizeBilinearV2 (*align_corners=False, half_pixel_centers=False*)

使用双线性插值调整图像尺寸到指定的大小。

调整过程只改变输入图像最低维度的数据，即图像的高和宽。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **align_corners** (bool, 可选) - 如果为 True，则使用比例 $(new\_height - 1)/(height - 1)$ 对输入进行缩放，此时输入图像和输出图像的四个角严格对齐。如果为 False，使用比例 $new\_height/height$ 对输入进行缩放。默认值：False。
- **half_pixel_centers** (bool, 可选) - 是否使用半像素中心对齐。如果设置为 True，那么 *align_corners* 应该设置为 False。默认值：False。

输入：

- **x** (Tensor) - 输入图像为四维的 Tensor，其 shape 为 $(batch, channels, height, width)$ ，支持的数据类型有：float16、float32。
- **size** (Union[tuple[int], list[int], Tensor]) - 调整后图像的尺寸，为含有两个元素的一维的 Tensor、list 或 tuple，两个元素分别为 (new_height, new_width) 。

输出：

Tensor，调整大小后的图像，是一个四维 Tensor，其 shape 为 $(batch, channels, new\_height, new\_width)$ ，数据类型与 *x* 一致。

异常：

- **TypeError** - *align_corners* 不是 bool。

- **TypeError** - `half_pixel_centers` 不是 bool。
- **TypeError** - `align_corners` 和 `half_pixel_centers` 同时为 True。
- **ValueError** - `half_pixel_centers` 为 True，同时运行平台为 CPU。
- **ValueError** - `x` 维度不是 4。
- **ValueError** - `size` 为 Tensor 且维度不是 1。
- **ValueError** - `size` 含有元素个数不是 2。

支持平台:

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> x = Tensor([[[[1, 2, 3, 4, 5], [1, 2, 3, 4, 5]]]], mindspore.float32)
>>> output = ops.ResizeBilinearV2()(x, (5, 5))
>>> print(output)
[[[1. 2. 3. 4. 5.]
  [1. 2. 3. 4. 5.]
  [1. 2. 3. 4. 5.]
  [1. 2. 3. 4. 5.]
  [1. 2. 3. 4. 5.]]]]
```

mindspore.ops.RoIAlign

```
class mindspore.ops.ROIAlign (pooled_height, pooled_width, spatial_scale, sample_num=2, roi_end_mode=1)
```

感兴趣区域对齐 (RoI Align) 运算。

RoI Align 通过在特征图上对附近网格点进行双线性插值计算每个采样点。**RoI Align** 不对 **RoI**、其子区域或采样点的中任何坐标执行量化。参阅论文 [Mask R-CNN](#)。

参数:

- **pooled_height** (int) - 输出特征高度。
- **pooled_width** (int) - 输出特征宽度。
- **spatial_scale** (float) - 缩放系数，将原始图像坐标映射到输入特征图坐标。设 RoI 的高度在原始图像中为 *ori_h*，在输入特征图中为 *fea_h*，则 *spatial_scale* 应为 *fea_h / ori_h*。
- **sample_num** (int) - 采样数。默认值：2。
- **roi_end_mode** (int) - 值必须为 0 或 1。如果值为 0，则使用该算子的历史实现。如果值为 1，则对 RoI 末尾的像素进行偏移，偏移量为 $+1 * spatial\_scale$ 。默认值：1。

输入:

- **features** (Tensor) - 输入特征, shape: (N, C, H, W) 。数据类型支持 float16 和 float32。
- **rois** (Tensor) - shape: $(rois\_n, 5)$ 。数据类型支持 float16 和 float32。*rois_n* 为 RoI 的数量。第二个维度的大小必须为 5, 分别代表 $(image\_index, top\_left\_x, top\_left\_y, bottom\_right\_x, bottom\_right\_y)$ 。*image_index* 表示图像的索引; *top_left_x* 和 *top_left_y* 分别对应 RoI 左上角坐标的 *x* 和 *y* 值; *bottom_right_x* 和 *bottom_right_y* 分别对应 RoI 右下角坐标的 *x* 和 *y* 值。

输出：

Tensor, shape: (*rois_n*, *C*, *pooled_height*, *pooled_width*) .

异常:

- **TypeError** - *pooled_height*、*pooled_width*、*sample_num* 或 *roi_end_mode* 不是 int 类型。
- **TypeError** - *spatial_scale* 不是 float 类型。
- **TypeError** - *features* 或 *rois* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> features = Tensor(np.array([[[[1., 2.], [3., 4.]]]]), mindspore.float32)
>>> rois = Tensor(np.array([[0, 0.2, 0.3, 0.2, 0.3]]), mindspore.float32)
>>> roi_align = ops.ROIAlign(2, 2, 0.5, 2)
>>> output = roi_align(features, rois)
>>> print(output)
[[[1.775 2.025]
  [2.275 2.525]]]]
```

20.2.8 文本处理

接口名	概述	支持平台
<code>mindspore.ops.NoRepeatNGram</code>	如果 n-grams 出现重复, 则更新对应 n-gram 词序列出现的概率。	Ascend GPU CPU

mindspore.ops.NoRepeatNGram

class mindspore.ops.NoRepeatNGram (*ngram_size=1*)
如果 **n-grams** 出现重复, 则更新对应 **n-gram** 词序列出现的概率。

在 *beam search* 过程中, 如果已生成的词序列中已经存在连续的 *ngram_size* 个词, 那么在后续预测时, 将避免再次出现该连续的 *ngram_size* 个词。例如: 当 *ngram_size* 为 3 时, 已生成的词序列为 [1,2,3,2,3], 则下一个预测的词不会为 2, 并且 *log_probs* 的值将被替换成 `-FLOAT_MAX`。因为连续的 3 个词 2、3、2 不会在词序列中出现两次。

参数:

- **ngram_size** (int, 可选) - 指定 **n-gram** 的长度, 必须大于 0。默认值: 1。

输入:

- **state_seq** (Tensor) - **n-gram** 词序列, 是一个三维 Tensor, 其 shape 为: (*batch_size, beam_width, m*)。
- **log_probs** (Tensor) - **n-gram** 词序列对应出现的概率, 是一个三维 Tensor, 其 shape 为: (*batch_size, beam_width, vocab_size*)。当 **n-gram** 重复时, *log_probs* 的值将被负 `FLOAT_MAX` 替换。

输出:

- **log_probs** (Tensor) - 数据类型和 shape 与输入 *log_probs* 相同。

异常:

- **TypeError** - 如果 *ngram_size* 不是 int。
- **TypeError** - 如果 *state_seq* 或 *log_probs* 不是 Tensor。

- **TypeError** - 如果 *state_seq* 的数据类型不是 `int`。
- **TypeError** - 如果 *log_probs* 的数据类型不是 `float`。
- **ValueError** - 如果 *ngram_size* 小于 0。
- **ValueError** - 如果 *ngram_size* 大于 *m*。
- **ValueError** - 如果 *state_seq* 或 *log_probs* 不是三维的 `Tensor`。
- **ValueError** - 如果 *state_seq* 和 *log_probs* 的 *batch_size* 不相等。
- **ValueError** - 如果 *state_seq* 和 *log_probs* 的 *beam_width* 不相等。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> no_repeat_ngram = ops.NoRepeatNGram(ngram_size=3)
>>> state_seq = Tensor([[[1, 2, 1, 2, 5, 1, 2],
...                      [9, 3, 9, 5, 4, 1, 5]],
...                    [[4, 8, 6, 4, 5, 6, 4],
...                      [4, 8, 8, 4, 3, 4, 8]]], dtype=mindspore.int32)
>>> log_probs = Tensor([[[0.7, 0.8, 0.6, 0.9, 0.2, 0.8, 0.4, 0.6, 0.2, 0.7],
...                       [0.4, 0.5, 0.6, 0.7, 0.8, 0.1, 0.9, 0.8, 0.7, 0.1]],
...                     [[0.9, 0.7, 0.6, 0.3, 0.5, 0.3, 0.5, 0.4, 0.8, 0.6],
...                       [0.5, 0.8, 0.8, 0.7, 0.7, 0.8, 0.2, 0.7, 0.9, 0.7]]],
dtype=mindspore.float32)
>>> output = no_repeat_ngram(state_seq, log_probs)
>>> print(output)
[[[ 6.9999999e-01 -3.4028235e+38  6.0000002e-01  8.9999998e-01
   2.0000000e-01 -3.4028235e+38  4.0000001e-01  6.0000002e-01
   2.0000000e-01  6.9999999e-01]
 [ 4.0000001e-01  5.0000000e-01  6.0000002e-01  6.9999999e-01
   8.0000001e-01  1.0000000e-01  8.9999998e-01  8.0000001e-01
   6.9999999e-01  1.0000000e-01]]
 [[ 8.9999998e-01  6.9999999e-01  6.0000002e-01  3.0000001e-01
   5.0000000e-01 -3.4028235e+38  5.0000000e-01  4.0000001e-01
   8.0000001e-01  6.0000002e-01]
 [ 5.0000000e-01  8.0000001e-01  8.0000001e-01  6.9999999e-01
   6.9999999e-01  8.0000001e-01  2.0000000e-01  6.9999999e-01
  -3.4028235e+38  6.9999999e-01]]]
```

20.3 数学运算算子

接口名	概述	支持平台
<code>mindspore.ops.Bincount</code>	计算整数数组中每个值的出现次数。	Ascend GPU CPU
<code>mindspore.ops.Cholesky</code>	计算单个或成批对称正定矩阵的 Cholesky 分解。	GPU CPU
<code>mindspore.ops.Complex</code>	给定复数的实部与虚部，返回一个复数的 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.ComplexAbs</code>	返回输入复数的模。	Ascend GPU CPU

下页继续

表 10 – 续上页

<code>mindspore.ops.Cross</code>	返回 <i>input</i> 和 <i>other</i> 沿着维度 <i>dim</i> 上的向量积 (叉积)。	Ascend CPU
<code>mindspore.ops.FFTWithSize</code>	傅里叶变换, 可以对参数进行调整, 以实现 FFT/IFFT/RFFT/IRFFT。	Ascend GPU CPU
<code>mindspore.ops.Gcd</code>	更多详情请查看: <code>mindspore.ops.gcd()</code> 。	Ascend GPU CPU

20.3.1 mindspore.ops.Bincount

class mindspore.ops.Bincount
计算整数数组中每个值的出现次数。

警告: 这是一个实验性 API, 后续可能修改或删除。

输入:

- **array** (Tensor) - int32 数据类型的 Tensor。
- **size** (Tensor) - int32 数据类型的非负 Tensor。
- **weights** (Tensor) - 与 *array* shape 相同或维度为 0 的 Tensor。当维度为 0 时, 所有权重均等于 1。数据类型必须是以下类型之一: int32、int64、float32、float64。

输出:

Tensor, 数据类型与 *weights* 相同。

异常:

- **TypeError** - *array* 的数据类型不是 int32。
- **TypeError** - *size* 的数据类型不是 int32。
- **ValueError** - *size* 中的数据存在负值。
- **ValueError** - *weights* 是空 Tensor。
- **ValueError** - *weights* 的 size 不为 0 且和 *array* 的 shape 不同。
- **TypeError** - *weights* 的数据类型不是 int32、int64、float32 或 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> array = Tensor(np.array([1, 2, 2, 3, 3, 3, 4, 4, 4, 4]), mindspore.int32)
>>> size = Tensor(5, mindspore.int32)
>>> weights = Tensor(np.array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1]), mindspore.float32)
>>> bincount = ops.Bincount()
>>> bins = bincount(array, size, weights)
>>> print(bins)
[0. 1. 2. 3. 4.]
```

20.3.2 mindspore.ops.Cholesky

class mindspore.ops.Cholesky (upper=False)
计算单个或成批对称正定矩阵的 Cholesky 分解。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.cholesky()`。

参数：

- **upper** (bool, 可选) - 为 True 返回上三角矩阵，否则返回下三角矩阵。默认值：False。

输入：

- **input_x** (Tensor) - shape 大小为 $(*, N, N)$ ，其中 * 是零个或多个由对称正定矩阵组成的批处理维，数据类型为 float32 或 float64。

输出：

Tensor，shape 和数据类型与 `input_x` 相同。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[1.0, 1.0], [1.0, 2.0]]), mindspore.float32)
>>> output = ops.Cholesky()(input_x)
>>> print(output)
[[1. 0.]
 [1. 1.]]
```

20.3.3 mindspore.ops.Complex

class mindspore.ops.Complex
给定复数的实部与虚部，返回一个复数的 Tensor。

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **real** (Tensor) - 实部的取值。数据类型：float32，float64。
- **imag** (Tensor) - 虚部的取值。数据类型：float32，float64。

输出：

Complex 类型的 Tensor。

异常：

- **TypeError** - 输入的数据类型不是 float32 或 float64 之一。

- **TypeError** - 输入的实部与虚部数据类型不一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> real = Tensor(np.array([1]), mindspore.float32)
>>> imag = Tensor(np.array([2]), mindspore.float32)
>>> complex = ops.Complex()
>>> output = complex(real, imag)
>>> print(output)
[1.+2.j]
```

20.3.4 mindspore.ops.ComplexAbs

class mindspore.ops.ComplexAbs

返回输入复数的模。

输入复数的形式为 $a + bj$ ，其中 a 为实部， b 为虚部。

$$y = \sqrt{a^2 + b^2}$$

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 复数 Tensor，格式须为 complex64 或 complex128。

输出：

Tensor。如果 x 的类型是 complex64，则输出的类型是 float32；如果 x 的类型是 complex128，则输出的类型是 float64。

异常：

- **TypeError** - 输入 x 不是 Tensor。
- **TypeError** - 输入 x 不是 complex64 或 complex128 格式。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.asarray(np.complex(3+4j)), mindspore.complex64)
>>> complex_abs = ops.ComplexAbs()
>>> output = complex_abs(x)
>>> print(output)
5.0
```

20.3.5 mindspore.ops.Cross

class mindspore.ops.**Cross** (*dim=-65530*)
 返回 *input* 和 *other* 沿着维度 *dim* 上的向量积（叉积）。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.cross()`。

参数：

- **dim** (int, 可选) - 沿着此维进行叉积操作。默认值：-65530。

输入：

- **input** (Tensor) - 输入 Tensor。
- **other** (Tensor) - 另一个输入 Tensor，数据类型和 *shape* 必须和 *input* 一致，并且他们的 *dim* 维度的长度应该为 3。

输出：

Tensor，数据类型与输入相同。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import dtype as mstype
>>> from mindspore import ops
>>> cross = ops.Cross(dim = 0)
>>> x1 = Tensor([1, 2, 3], mstype.int8)
>>> x2 = Tensor([1, 2, 3], mstype.int8)
>>> output = cross(x1, x2)
>>> print(output)
[0 0 0]
```

20.3.6 mindspore.ops.FFTWithSize

class mindspore.ops.**FFTWithSize** (*signal_ndim, inverse, real, norm='backward', onesided=True, signal_sizes=()*)
 傅里叶变换，可以对参数进行调整，以实现 FFT/IFFT/RFFT/IRFFT。

对于 FFT，它计算以下表达式：

$$X[\omega_1, \dots, \omega_d] = \sum_{n_1=0}^{N_1-1} \cdots \sum_{n_d=0}^{N_d-1} x[n_1, \dots, n_d] e^{-j 2\pi \sum_{i=0}^d \frac{\omega_i n_i}{N_i}},$$

其中， $d = \text{signal\_ndim}$ 是信号的维度， N_i 则是信号第 i 个维度的大小。

对于 IFFT，它计算以下表达式：

$$X[\omega_1, \dots, \omega_d] = \frac{1}{\prod_{i=1}^d N_i} \sum_{n_1=0}^{N_1-1} \cdots \sum_{n_d=0}^{N_d-1} x[n_1, \dots, n_d] e^{j 2\pi \sum_{i=0}^d \frac{\omega_i n_i}{N_i}},$$

其中, $d = \text{signal\_ndim}$ 是信号的维度, N_i 则是信号第 i 维的大小。

说明:

- FFT/IFFT 要求 complex64 或 complex128 类型的输入, 返回 complex64 或 complex128 类型的输出。
- RFFT 要求 bool, uint8, int8, int16, int32, int64, float32 或 float64 类型的输入, 返回 complex64 或 complex128 类型的输出。
- IRFFT 要求 complex64 或 complex128 类型的输入, 返回 float32 或 float64 类型的输出。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **signal_ndim** (int) - 表示每个信号中的维数, 控制着傅里叶变换的维数, 其值只能为 1、2 或 3。
- **inverse** (bool) - 表示该操作是否为逆变换, 用以选择 FFT 和 RFFT 或 IFFT 和 IRFFT。
 - 如果为 True, 则为 IFFT 和 IRFFT。
 - 如果为 False, FFT 和 RFFT。
- **real** (bool) - 表示该操作是否为实变换, 与 *inverse* 共同决定具体的变换模式:
 - *inverse* 为 False, *real* 为 False: 对应 FFT 模式。
 - *inverse* 为 True, *real* 为 False: 对应 IFFT 模式。
 - *inverse* 为 False, *real* 为 True: 对应 RFFT 模式。
 - *inverse* 为 True, *real* 为 True: 对应 IRFFT 模式。
- **norm** (str, 可选) - 表示该操作的规范化方式, 可选值: ["backward", "forward", "ortho"]。默认值: "backward"。
 - "backward", 正向变换不缩放, 逆变换按 $1/n$ 缩放, 其中 n 表示输入 x 的元素数量。
 - "ortho", 正向变换与逆变换均按 $1/\sqrt{n}$ 缩放。
 - "forward", 正向变换按 $1/n$ 缩放, 逆变换不缩放。
- **onesided** (bool, 可选) - 控制输入是否减半以避免冗余。默认值: True。
- **signal_sizes** (tuple, 可选) - 原始信号的大小 (RFFT 变换之前的信号, 不包含 batch 这一维), 只有在 IRFFT 模式下和设置 *onesided* 为 True 时需要该参数, 需要满足以下条件。默认值: ()。
 - *signal_sizes* 的长度等于 IRFFT 的 *signal_ndim*: $\text{len}(\text{signal\_sizes}) = \text{signal\_ndim}$ 。
 - *signal_sizes* 的最后一个维度除以 2 等于 IRFFT 输入的最后一个维度: $\text{signal\_size}[-1]/2+1 = x.\text{shape}[-1]$ 。
 - 除了最后一个维度外, *signal_sizes* 的维度与输入 *shape* 完全相同: $\text{signal\_sizes}[:-1] = x.\text{shape}[:-1]$ 。

输入:

- **x** (Tensor) - 输入 Tensor 的维数必须大于或等于 *signal_ndim*。

输出:

Tensor, 表示复数到复数、实数到复数或复数到实数傅里叶变换的结果。

异常:

- **TypeError** - 如果 FFT/IFFT/IRFF 的输入类型不是以下类型之一：complex64、complex128。
- **TypeError** - 如果输入的类型不是 Tensor。
- **ValueError** - 如果输入 x 的维度小于 `signal_ndim`。
- **ValueError** - 如果 `signal_ndim` 大于 3 或小于 1。
- **ValueError** - 如果 `norm` 取值不是“backward”、“forward”或“ortho”。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case FFT: signal_ndim: 1, inverse: False, real: False.
>>> fft_in = Tensor(np.array([2, 1, 2]), mindspore.complex64)
>>> fft_net = ops.FFTWithSize(signal_ndim=1, inverse=False, real=False)
>>> fft_output = fft_net(fft_in)
>>> print(fft_output)
[5.          +0.j          0.5          +0.86602545j 0.50000006-0.8660255j ]
>>> # case IFFT: signal_ndim: 1, inverse: True, real: False.
>>> ifft_in = fft_output
>>> ifft_net = ops.FFTWithSize(signal_ndim=1, inverse=True, real=False)
>>> ifft_output = ifft_net(ifft_in)
>>> print(ifft_output)
[2.          -1.9868216e-08j 0.99999994+0.0000000e+00j
 1.9999999 +7.9472862e-08j]
>>> # case RFFT2D: signal_ndim: 2, inverse: False, real: True.
>>> rfft_in = Tensor(np.array([2, 1, 2], [3, 1, 6])), mindspore.float32)
>>> rfft_net = ops.FFTWithSize(signal_ndim=2, inverse=False, real=True)
>>> rfft_output = rfft_net(rfft_in)
>>> print(rfft_output)
[[ 1.5000000e+01+1.1920929e-07j -2.3841858e-07+5.1961522e+00j]
 [-5.0000000e+00-2.9802322e-08j  9.9999988e-01-3.4641016e+00j]]
>>> # case IRFFT2D: signal_ndim: 2, inverse: True, real: True.
>>> irfft_in = rfft_output
>>> irfft_net = ops.FFTWithSize(signal_ndim=2, inverse=True, real=True, signal_sizes=rfft_in.
↳shape)
>>> irfft_output = irfft_net(irfft_in)
>>> print(irfft_output)
[[2.          1.          2.          ]
 [3.          0.99999994 5.9999995 ]]
```

20.3.7 mindspore.ops.Gcd

class mindspore.ops.Gcd

```
prim = ops.Gcd()
out = prim(input, other)
```

等价于

```
ops.gcd(input, other)
```

更多详情请查看: `mindspore.ops.gcd()`。

20.3.8 逐元素运算

接口名	概述	支持平台
<code>mindspore.ops.Abs</code>	更多详情请查看: <code>mindspore.ops.abs()</code> 。	Ascend GPU CPU
<code>mindspore.ops.AccumulateNV2</code>	逐元素将所有输入的 Tensor 相加。	Ascend GPU
<code>mindspore.ops.ACos</code>	更多详情请查看: <code>mindspore.ops.acos()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Acosh</code>	更多详情请查看: <code>mindspore.ops.acosh()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Add</code>	更多详情请查看: <code>mindspore.ops.add()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Addcdiv</code>	将 <code>x1</code> 和 <code>x2</code> 的逐元素相除的结果乘以 <code>value</code> , 并将其添加到 <code>input_data</code> 中。	Ascend GPU CPU
<code>mindspore.ops.Addcmul</code>	将 <code>x1</code> 和 <code>x2</code> 的逐元素相乘的结果乘以 <code>value</code> , 并将其添加到 <code>input_data</code> 中。	Ascend GPU CPU
<code>mindspore.ops.AddN</code>	逐元素将所有输入的 Tensor 相加。	Ascend GPU CPU
<code>mindspore.ops.Angle</code>	更多详情请查看: <code>mindspore.ops.angle()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Asin</code>	更多详情请查看: <code>mindspore.ops.asin()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Asinh</code>	更多详情请查看: <code>mindspore.ops.asinh()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Atan</code>	更多详情请查看: <code>mindspore.ops.atan()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Atan2</code>	更多详情请查看: <code>mindspore.ops.atan2()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Atanh</code>	更多详情请查看: <code>mindspore.ops.atanh()</code> 。	Ascend GPU CPU
<code>mindspore.ops.BesselI0</code>	逐元素计算第一类零阶修正 Bessel 函数值。	GPU CPU
<code>mindspore.ops.BesselI0e</code>	逐元素计算指数缩放第一类零阶修正贝塞尔函数。	Ascend GPU CPU
<code>mindspore.ops.BesselI1</code>	逐元素计算第一类一阶修正 Bessel 函数值。	GPU CPU
<code>mindspore.ops.BesselI1e</code>	逐元素计算指数缩放第一类一阶修正贝塞尔函数。	Ascend GPU CPU
<code>mindspore.ops.BesselJ0</code>	逐元素计算输入数据的第一类零阶的 Bessel 函数。	GPU CPU
<code>mindspore.ops.BesselJ1</code>	逐元素计算输入数据的第一类一阶的 Bessel 函数。	GPU CPU
<code>mindspore.ops.BesselK0</code>	逐元素计算第二类零阶修正 Bessel 函数值。	GPU CPU
<code>mindspore.ops.BesselK0e</code>	逐元素计算指数缩放第二类零阶修正 Bessel 函数值。	GPU CPU
<code>mindspore.ops.BesselK1</code>	逐元素计算第二类一阶修正 Bessel 函数值。	GPU CPU
<code>mindspore.ops.BesselK1e</code>	逐元素计算指数缩放第二类一阶修正 Bessel 函数值。	GPU CPU
<code>mindspore.ops.BesselY0</code>	逐元素计算输入数据的第二类零阶 Bessel 函数。	GPU CPU
<code>mindspore.ops.BesselY1</code>	逐元素计算输入数据的第二类一阶 Bessel 函数。	GPU CPU
<code>mindspore.ops.BitwiseAnd</code>	逐元素执行两个 Tensor 的与运算。	Ascend GPU CPU

下页继续

表 11 – 续上页

<code>mindspore.ops.BitwiseOr</code>	逐元素执行两个 Tensor 的或运算。	Ascend GPU CPU
<code>mindspore.ops.BitwiseXor</code>	逐元素执行两个 Tensor 的异或运算。	Ascend GPU CPU
<code>mindspore.ops.Ceil</code>	更多详情请查看: <code>mindspore.ops.ceil()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Conj</code>	更多详情请查看: <code>mindspore.ops.conj()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Cos</code>	更多详情请查看: <code>mindspore.ops.cos()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Cosh</code>	更多详情请查看: <code>mindspore.ops.cosh()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Digamma</code>	计算对数 gamma 函数在输入上的导数。	GPU CPU
<code>mindspore.ops.Div</code>	逐元素计算第一输入 Tensor 除以第二输入 Tensor 的商。	Ascend GPU CPU
<code>mindspore.ops.DivNoNaN</code>	对 $x1$ 和 $x2$ 逐元素执行安全除法, 如果 $x2$ 的元素为 0, 则返回 0。	Ascend GPU CPU
<code>mindspore.ops.Einsum</code>	此算子使用爱因斯坦求和约定 (Einsum) 进行 Tensor 计算。	GPU
<code>mindspore.ops.Erf</code>	更多详情请查看: <code>mindspore.ops.erf()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Erfc</code>	更多详情请查看: <code>mindspore.ops.erfc()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Erfinv</code>	更多详情请查看: <code>mindspore.ops.erfinv()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Exp</code>	更多详情请查看: <code>mindspore.ops.exp()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Expm1</code>	更多详情请查看: <code>mindspore.ops.expm1()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Floor</code>	更多详情请查看: <code>mindspore.ops.floor()</code> 。	Ascend GPU CPU
<code>mindspore.ops.FloorDiv</code>	更多详情请查看: <code>mindspore.ops.floor_divide()</code> 。	Ascend GPU CPU
<code>mindspore.ops.FloorMod</code>	更多详情请查看: <code>mindspore.ops.floor_mod()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Geqrf</code>	更多详情请查看: <code>mindspore.ops.geqrf()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Imag</code>	返回包含输入 Tensor 的虚部。	Ascend GPU CPU
<code>mindspore.ops.Inv</code>	按元素计算输入 Tensor 的倒数。	Ascend GPU CPU
<code>mindspore.ops.Invert</code>	翻转输入 Tensor 的所有元素。	Ascend GPU CPU
<code>mindspore.ops.Lerp</code>	更多详情请查看: <code>mindspore.ops.lerp()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Log</code>	更多详情请查看: <code>mindspore.ops.log()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Log1p</code>	更多详情请查看: <code>mindspore.ops.log1p()</code> 。	Ascend GPU CPU
<code>mindspore.ops.LogicalAnd</code>	逐元素进行两个 Tensor 的逻辑与运算。	Ascend GPU CPU
<code>mindspore.ops.LogicalNot</code>	逐元素计算一个 Tensor 的逻辑非运算。	Ascend GPU CPU
<code>mindspore.ops.LogicalOr</code>	逐元素计算两个 Tensor 的逻辑或运算。	Ascend GPU CPU
<code>mindspore.ops.LogicalXor</code>	逐元素计算两个 Tensor 的逻辑异或运算。	Ascend CPU
<code>mindspore.ops.Logit</code>	逐元素计算 Tensor 的 logit 值。	Ascend GPU CPU
<code>mindspore.ops.Mod</code>	将第一个输入 Tensor 逐元素除以第二个输入 Tensor , 并取余。	Ascend GPU CPU
<code>mindspore.ops.Mul</code>	更多详情请查看: <code>mindspore.ops.mul()</code> 。	Ascend GPU CPU
<code>mindspore.ops.MulNoNaN</code>	逐元素计算输入乘积。	Ascend GPU CPU
<code>mindspore.ops.Neg</code>	更多详情请查看: <code>mindspore.ops.neg()</code> 。	Ascend GPU CPU

下页继续

表 11 – 续上页

<code>mindspore.ops.NextAfter</code>	更多详情请查看: <code>mindspore.ops.nextafter()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Polar</code>	将极坐标转化为笛卡尔坐标。	Ascend GPU CPU
<code>mindspore.ops.Polygamma</code>	计算关于 x 的多伽马函数的 a 阶导数。	GPU CPU
<code>mindspore.ops.Pow</code>	更多详情请查看: <code>mindspore.ops.pow()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Real</code>	更多详情请查看: <code>mindspore.ops.real()</code> 。	Ascend GPU CPU
<code>mindspore.ops.RealDiv</code>	第一个输入 Tensor 元素为分子, 第二个输入 Tensor 元素为分母, 逐元素进行浮点型除法运算。	Ascend GPU CPU
<code>mindspore.ops.Reciprocal</code>	返回输入 Tensor 的倒数。	Ascend GPU CPU
<code>mindspore.ops.Rint</code>	逐元素计算最接近输入数据的整数。	Ascend GPU CPU
<code>mindspore.ops.Round</code>	对输入数据进行四舍五入, 得到最接近的整数数值。	Ascend GPU CPU
<code>mindspore.ops.Rsqrt</code>	更多详情请查看: <code>mindspore.ops.rsqrt()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Sign</code>	更多详情请查看: <code>mindspore.ops.sign()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Sin</code>	更多详情请查看: <code>mindspore.ops.sin()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Sinc</code>	更多详情请查看: <code>mindspore.ops.sinc()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Sinh</code>	更多详情请查看: <code>mindspore.ops.sinh()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Sqrt</code>	更多详情请查看: <code>mindspore.ops.sqrt()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Square</code>	更多详情请查看: <code>mindspore.ops.square()</code> 。	Ascend GPU CPU
<code>mindspore.ops.SquaredDifference</code>	第一个输入 Tensor 逐元素减去第二个输入 Tensor , 并返回其平方。	Ascend GPU CPU
<code>mindspore.ops.SquareSumAll</code>	计算输入 Tensor 的平方和。	Ascend GPU CPU
<code>mindspore.ops.Sub</code>	逐元素用第一个输入 Tensor 减去第二个输入 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.Tan</code>	逐元素计算输入元素的正切值。	Ascend GPU CPU
<code>mindspore.ops.Trunc</code>	更多详情请查看: <code>mindspore.ops.trunc()</code> 。	Ascend GPU CPU
<code>mindspore.ops.TruncateDiv</code>	将第一个输入 Tensor 与第二个输入 Tensor 逐元素相除, 结果将向 0 取整。	Ascend GPU CPU
<code>mindspore.ops.TruncateMod</code>	逐元素取模。	Ascend GPU CPU
<code>mindspore.ops.Xdivy</code>	将第一个输入 Tensor 除以第二个输入 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.Xlogy</code>	计算第一个输入 Tensor 乘以第二个输入 Tensor 的对数。	Ascend GPU CPU
<code>mindspore.ops.Zeta</code>	计算输入 Tensor 的 Hurwitz zeta 函数 $\zeta(x,q)$ 值。	Ascend GPU CPU

mindspore.ops.Abs

class mindspore.ops.Abs

```
prim = ops.Abs()
out = prim(input)
```

等价于

```
ops.abs(input)
```

更多详情请查看: [mindspore.ops.abs\(\)](#)。

mindspore.ops.AccumulateNV2

class mindspore.ops.AccumulateNV2

逐元素将所有输入的 Tensor 相加。

有关更多详细信息, 请参阅: [mindspore.ops.accumulate_n\(\)](#)。

输入:

- **x** (Union(tuple[Tensor], list[Tensor])) - AccumulateNV2 的输入, 由多个 Tensor 组成的 tuple 或 list, 其数据类型为数值型, 且每个元素的 shape 必须相等。

输出:

Tensor, 数据类型和 shape 与输入 *x* 的每个条目相同。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, nn
>>> class NetAccumulateNV2(nn.Cell):
...     def __init__(self):
...         super(NetAccumulateNV2, self).__init__()
...         self.accumulateNV2 = ops.AccumulateNV2()
...
...     def construct(self, *z):
...         return self.accumulateNV2(z)
...
>>> net = NetAccumulateNV2()
>>> x = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> y = Tensor(np.array([4, 5, 6]), mindspore.float32)
>>> output = net(x, y, x, y)
>>> print(output)
[10. 14. 18.]
```

mindspore.ops.ACos

class mindspore.ops.ACos

```
prim = ops.ACos()
out = prim(input)
```

等价于

```
ops.acos(input)
```

更多详情请查看: [mindspore.ops.acos\(\)](#)。

mindspore.ops.Acosh

class mindspore.ops.Acosh

```
prim = ops.Acosh()
out = prim(input)
```

等价于

```
ops.acosh(input)
```

更多详情请查看: [mindspore.ops.acosh\(\)](#)。

mindspore.ops.Add

class mindspore.ops.Add

```
prim = ops.Add()
out = prim(input, other)
```

等价于

```
ops.add(input, other)
```

更多详情请查看: [mindspore.ops.add\(\)](#)。

mindspore.ops.Addcddiv

class mindspore.ops.Addcddiv

将 $x1$ 和 $x2$ 的逐元素相除的结果乘以 $value$ ，并将其添加到 $input\_data$ 中。计算操作如下：

$$y[i] = input\_data[i] + value[i] * (x1[i]/x2[i])$$

输入：

- **input_data** (Tensor) - 要添加的 Tensor。
- **x1** (Tensor) - 分子 Tensor。
- **x2** (Tensor) - 分母 Tensor。
- **value** (Tensor) - Tensor $x1/x2$ 的乘数。

输出：

Tensor, 具有与 $x1/x2$ 相同的 shape 和 dtype。

异常：

- **TypeError** - 如果 $x1$ 、 $x2$ 、 $value$ 、 $input\_data$ 不是 Tensor。
- **TypeError** - 如果 $x1$ 、 $x2$ 、 $value$ 、 $input\_data$ 的 dtype 不一致。
- **ValueError** - 如果无法将 $x1$ 广播到 $x2$ 。
- **ValueError** - 如果无法将 $value$ 广播到 $x1/x2$ 。
- **ValueError** - 如果无法将 $input\_data$ 广播到 $value*(x1/x2)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_data = Tensor(np.array([1, 1, 1, 1]), mindspore.float32)
>>> x1 = Tensor(np.array([1, 2, 3, 4]), mindspore.float32)
>>> x2 = Tensor(np.array([4, 3, 2, 1]), mindspore.float32)
>>> value = Tensor([1], mindspore.float32)
>>> addcddiv = ops.Addcddiv()
>>> y = addcddiv(input_data, x1, x2, value)
>>> print(y)
[1.25      1.6666667 2.5      5.      ]
```

mindspore.ops.Addcmul

class mindspore.ops.Addcmul

将 $x1$ 和 $x2$ 的逐元素相乘的结果乘以 $value$ ，并将其添加到 $input\_data$ 中。计算操作如下：

$$output[i] = input\_data[i] + value[i] * (x1[i] * x2[i])$$

输入：

- **input_data** (Tensor) - 要添加的 Tensor。
- **x1** (Tensor) - 要乘以的 Tensor。
- **x2** (Tensor) - 要乘以的 Tensor。
- **value** (Tensor) - Tensor $x1*x2$ 的乘数。

输出：

Tensor, 具有与 $x1*x2$ 相同的 shape 和 dtype。

异常：

- **TypeError** - 如果 $x1$ 、 $x2$ 、 $value$ 、 $input\_data$ 不是 Tensor。
- **TypeError** - 如果 $x1$ 、 $x2$ 、 $value$ 、 $input\_data$ 的 dtype 不一致。
- **ValueError** - 如果无法将 $x1$ 广播到 $x2$ 。
- **ValueError** - 如果无法将 $value$ 广播到 $x1 * x2$ 。

- **ValueError** - 如果无法将 *input_data* 广播到 $value*(x1*x2)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_data = Tensor(np.array([1, 1, 1]), mindspore.float32)
>>> x1 = Tensor(np.array([[1], [2], [3]]), mindspore.float32)
>>> x2 = Tensor(np.array([[1, 2, 3]]), mindspore.float32)
>>> value = Tensor([1], mindspore.float32)
>>> addcmul = ops.Addcmul()
>>> y = addcmul(input_data, x1, x2, value)
>>> print(y)
[[ 2.  3.  4.]
 [ 3.  5.  7.]
 [ 4.  7. 10.]]
```

mindspore.ops.AddN

class mindspore.ops.AddN

逐元素将所有输入的 Tensor 相加。

更多参考详见 [*mindspore.ops.addn\(\)*](#)。

输入：

- **x** (Union(tuple[Tensor], list[Tensor])) - Tensor 组成的 tuple 或 list，类型为 bool 型或数值型。

输出：

Tensor，与 *x* 的每个 Tensor 具有相同的 shape 和数据类型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops
>>> class NetAddN(nn.Cell):
...     def __init__(self):
...         super(NetAddN, self).__init__()
...         self.addN = ops.AddN()
...
...     def construct(self, *z):
...         return self.addN(z)
...
>>> net = NetAddN()
>>> x = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> y = Tensor(np.array([4, 5, 6]), mindspore.float32)
>>> output = net(x, y, x, y)
>>> print(output)
[10. 14. 18.]
```

mindspore.ops.Angle

class mindspore.ops.Angle

```
prim = ops.Angle()  
out = prim(input)
```

等价于

```
ops.angle(input)
```

更多详情请查看: [*mindspore.ops.angle\(\)*](#) 。

mindspore.ops.Asin

class mindspore.ops.Asin

```
prim = ops.Asin()  
out = prim(input)
```

等价于

```
ops.asin(input)
```

更多详情请查看: [*mindspore.ops.asin\(\)*](#) 。

mindspore.ops.Asinh

class mindspore.ops.Asinh

```
prim = ops.Asinh()  
out = prim(input)
```

等价于

```
ops.asinh(input)
```

更多详情请查看: [*mindspore.ops.asinh\(\)*](#) 。

mindspore.ops.Atan

class mindspore.ops.Atan

```
prim = ops.Atan()  
out = prim(input)
```

等价于

```
ops.atan(input)
```

更多详情请查看: [*mindspore.ops.atan\(\)*](#) 。

mindspore.ops.Atan2

class mindspore.ops.Atan2

```
prim = ops.Atan2()
out = prim(input, other)
```

等价于

```
ops.atan2(input, other)
```

更多详情请查看: [mindspore.ops.atan2\(\)](#)。

mindspore.ops.Atanh

class mindspore.ops.Atanh

```
prim = ops.Atanh()
out = prim(input)
```

等价于

```
ops.atanh(input)
```

更多详情请查看: [mindspore.ops.atanh\(\)](#)。

mindspore.ops.BesselI0

class mindspore.ops.BesselI0

逐元素计算第一类零阶修正 Bessel 函数值。

计算公式定义如下:

$$I_0(x) = J_0(ix) = \sum_{m=0}^{\infty} \frac{x^{2m}}{2^{2m}(m!)^2}$$

其中 J_0 是第一类零阶 Bessel 函数。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多细节请参考[mindspore.ops.bessel_i0\(\)](#)。

输入:

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出:

Tensor, shape 和数据类型与 x 相同。

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessell_i0 = ops.BesselI0()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessell_i0(x)
>>> print(output)
[1.0144521 1.1797839 1.0241698 1.0020262]
```

mindspore.ops.BesselI0e

class mindspore.ops.BesselI0e

逐元素计算指数缩放第一类零阶修正贝塞尔函数。

计算公式定义如下：

$$I_0e(x) = e^{(-|x|)} * I_0(x) = e^{(-|x|)} * \sum_{m=0}^{\infty} \frac{x^{2m}}{2^{2m}(m!)^2}$$

其中 I_0 是第一类零阶修正 Bessel 函数。

输入：

- **x (Tensor)** - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 x 相同。

异常：

- **TypeError** - x 不是 Tensor。
- **TypeError** - x 的数据类型不是 float16、float32 或 float64。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessell_i0e = ops.BesselI0e()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessell_i0e(x)
>>> print(output)
[0.7979961 0.5144438 0.75117415 0.9157829 ]
```

mindspore.ops.BesselI1

class mindspore.ops.BesselI1

逐元素计算第一类一阶修正 Bessel 函数值。

计算公式定义如下：

$$I_1(x) = i^{-1} J_1(ix) = \sum_{m=0}^{\infty} \frac{x^{2m+1}}{2^{2m+1} m!(m+1)!}$$

其中 J_1 是第一类一阶的 Bessel 函数。

警告：这是一个实验性 API，后续可能修改或删除。

更多细节请参考 `mindspore.ops.bessel_i1()`。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 x 相同。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessel_i1 = ops.BesselI1()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessel_i1(x)
>>> print(output)
[0.1208661  0.45177728 0.1568694  0.04504559]
```

mindspore.ops.BesselI1e

class mindspore.ops.BesselI1e

逐元素计算指数缩放第一类一阶修正贝塞尔函数。

计算公式定义如下：

$$I_1 e(x) = e^{(-|x|)} * I_1(x) = e^{(-|x|)} * \sum_{m=0}^{\infty} \frac{x^{2m+1}}{2^{2m+1} m!(m+1)!}$$

其中 I_1 是第一类一阶修正 Bessel 函数。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 x 相同。

异常:

- **TypeError** - x 不是 Tensor。
- **TypeError** - x 的数据类型不是 float16, float32 或 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessell1e = ops.BesselI1e()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessell1e(x)
>>> print(output)
[0.09507662 0.19699717 0.11505538 0.04116856]
```

mindspore.ops.BesselJ0

class mindspore.ops.BesselJ0

逐元素计算输入数据的第一类零阶的 Bessel 函数。

计算公式定义如下:

$$J_0(x) = \frac{1}{\pi} \int_0^{\pi} \cos(x \sin \theta) d\theta = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m}}{2^{2m} (m!)^2}$$

警告: 这是一个实验性 API, 后续可能修改或删除。

输入:

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出:

Tensor, shape 和数据类型与 x 相同。

异常:

- **TypeError** - x 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> besselj0 = ops.BesselJ0()
>>> x = Tensor(np.array([0.5, 1., 2., 4.]), mindspore.float32)
>>> output = besselj0(x)
>>> print(output)
[0.93846981 0.76519769 0.22389078 -0.39714981]
```

mindspore.ops.BesselJ1

class mindspore.ops.BesselJ1

逐元素计算输入数据的第一类一阶的 Bessel 函数。

计算公式定义如下：

$$J_1(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \theta - \theta) d\theta = \sum_{m=0}^{\infty} \frac{(-1)^m x^{2m+1}}{2^{2m+1} m! (m+1)!}$$

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 *x* 相同。

异常：

- **TypeError** - *x* 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessel_j1 = ops.BesselJ1()
>>> x = Tensor(np.array([0.5, 1., 2., 4.]), mindspore.float32)
>>> output = bessel_j1(x)
>>> print(output)
[0.24226846  0.44005059  0.57672481 -0.06604333]
```

mindspore.ops.BesselK0

class mindspore.ops.BesselK0

逐元素计算第二类零阶修正 Bessel 函数值。

计算公式定义如下：

$$K_0(x) = \lim_{\nu \rightarrow 0} \left(\frac{\pi}{2} \right) \frac{I_{-\nu}(x) - I_{\nu}(x)}{\sin(\nu\pi)} = \int_0^\infty e^{-x \cosh t} dt$$

其中 I_0 是第一类零阶修正 Bessel 函数。

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 x 相同。

异常：

- **TypeError** - x 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessell_k0 = ops.BesselK0()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessell_k0(x)
>>> print(output)
[1.579826  0.5402144 1.3424659 2.5310173]
```

mindspore.ops.BesselK0e

class mindspore.ops.BesselK0e

逐元素计算指数缩放第二类零阶修正 Bessel 函数值。

计算公式定义如下：

$$K_0 e(x) = e^{(-|x|)} * K_0(x) = e^{(-|x|)} * \int_0^\infty e^{-x \cosh t} dt$$

其中 K_0 是第二类零阶修正 Bessel 函数。

警告： 这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 x 相同。

异常：

- **TypeError** - x 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台：

GPU CPU

样例：


```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessell_k0e = ops.BesselK0e()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessell_k0e(x)
>>> print(output)
[2.0083523 1.2388839 1.8303517 2.769374 ]
```

mindspore.ops.BesselK1

class mindspore.ops.BesselK1

逐元素计算第二类一阶修正 Bessel 函数值。

计算公式定义如下：

$$K_1(x) = \lim_{\nu \rightarrow 1} \left(\frac{\pi}{2} \right) \frac{I_{-\nu}(x) - I_{\nu}(x)}{\sin(\nu\pi)} = \int_0^{\infty} e^{-x \cosh t} \cosh(t) dt$$

其中 I_1 是第一类一阶修正 Bessel 函数。

警告： 这是一个实验性 API，后续可能修改或删除。

输入：

- **x (Tensor)** - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 x 相同。

异常：

- **TypeError** - x 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessell_k1 = ops.BesselK1()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessell_k1(x)
>>> print(output)
[3.9190812 0.8143549 2.9440577 10.974864]
```

mindspore.ops.BesselK1e

class mindspore.ops.BesselK1e

逐元素计算指数缩放第二类一阶修正 Bessel 函数值。

计算公式定义如下：

$$K_1 e(x) = e^{(-|x|)} * K_1(x) = e^{(-|x|)} * \int_0^\infty e^{-x \cosh t} \cosh(t) dt$$

其中 K_1 是第二类一阶修正 Bessel 函数。

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor，shape 和数据类型与 x 相同。

异常：

- **TypeError** - x 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessell_k1e = ops.BesselK1e()
>>> x = Tensor(np.array([0.24, 0.83, 0.31, 0.09]), mindspore.float32)
>>> output = bessell_k1e(x)
>>> print(output)
[ 4.9821286  1.8675754  4.0140023 12.008413 ]
```

mindspore.ops.BesselY0

class mindspore.ops.BesselY0

逐元素计算输入数据的第二类零阶 Bessel 函数。

计算公式定义如下：

$$Y_0(x) = \lim_{n \rightarrow 0} \frac{J_n(x) \cos n\pi - J_{-n}(x)}{\sin n\pi}$$

其中 J_0 为第一类零阶 Bessel 函数。

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 *x* 相同。

异常：

- **TypeError** - *x* 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessely0 = ops.Bessely0()
>>> x = Tensor(np.array([0.5, 1., 2., 4.]), mindspore.float32)
>>> output = bessely0(x)
>>> print(output)
[-0.44451873  0.08825696  0.51037567 -0.01694074]
```

mindspore.ops.Bessely1

class mindspore.ops.Bessely1

逐元素计算输入数据的第二类一阶 Bessel 函数。

计算公式定义如下：

$$Y_1(x) = \lim_{n \rightarrow 1} \frac{J_n(x) \cos n\pi - J_{-n}(x)}{\sin n\pi}$$

其中 J_1 为第一类一阶 Bessel 函数。

警告： 这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型应为 float16、float32 或 float64。

输出：

Tensor, shape 和数据类型与 *x* 相同。

异常：

- **TypeError** - *x* 不是 float16、float32 或 float64 数据类型的 Tensor。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> bessel_y1 = ops.BesselY1()
>>> x = Tensor(np.array([0.5, 1., 2., 4.]), mindspore.float32)
>>> output = bessel_y1(x)
>>> print(output)
[-1.47147239 -0.78121282 -0.10703243  0.39792571]
```

mindspore.ops.BitwiseAnd

class mindspore.ops.BitwiseAnd

逐元素执行两个 Tensor 的与运算。

更多细节详见 `mindspore.ops.bitwise_and()`。

输入：

- **x** (Tensor) - 第一个输入 Tensor，其 shape 为 $(N,*)$ ，其中 $*$ 为任意数量的额外维度。
- **y** (Tensor) - 第二个输入 Tensor，数据类型与 x 一致。

输出：

Tensor，是一个与 x 相同类型的 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> y = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> bitwise_and = ops.BitwiseAnd()
>>> output = bitwise_and(x, y)
>>> print(output)
[ 0  0  1 -1  1  0  1]
```

mindspore.ops.BitwiseOr

class mindspore.ops.BitwiseOr

逐元素执行两个 Tensor 的或运算。

更多细节详见 `mindspore.ops.bitwise_or()`。

输入：

- **x** (Tensor) - 第一个输入 Tensor，其 shape 为 $(N,*)$ ，其中 $*$ 为任意数量的额外维度。
- **y** (Tensor) - 第二个输入 Tensor，数据类型与 x 一致。

输出：

Tensor，是一个与 x 相同类型的 Tensor。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> y = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> bitwise_or = ops.BitwiseOr()
>>> output = bitwise_or(x, y)
>>> print(output)
[ 0  1  1 -1 -1  3  3]
```

mindspore.ops.BitwiseXor

class mindspore.ops.BitwiseXor

逐元素执行两个 Tensor 的异或运算。

警告: 该 API 在 CPU 上性能较差, 推荐在 Ascend/GPU 上运行。

更多细节详见 `mindspore.ops.bitwise_xor()`。

输入:

- **x** (Tensor) - 第一个输入 Tensor, 其 shape 为 $(N, *)$, 其中 $*$ 为任意数量的额外维度。
- **y** (Tensor) - 第二个输入 Tensor, 数据类型与 x 一致。

输出:

Tensor, 是一个与 x 相同类型的 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([0, 0, 1, -1, 1, 1, 1]), mindspore.int16)
>>> y = Tensor(np.array([0, 1, 1, -1, -1, 2, 3]), mindspore.int16)
>>> bitwise_xor = ops.BitwiseXor()
>>> output = bitwise_xor(x, y)
>>> print(output)
[ 0  1  0  0 -2  3  2]
```

mindspore.ops.Ceil

class mindspore.ops.Ceil

```
prim = ops.Ceil()  
out = prim(input)
```

等价于

```
ops.ceil(input)
```

更多详情请查看: [*mindspore.ops.ceil\(\)*](#) 。

mindspore.ops.Conj

class mindspore.ops.Conj

```
prim = ops.Conj()  
out = prim(input)
```

等价于

```
ops.conj(input)
```

更多详情请查看: [*mindspore.ops.conj\(\)*](#) 。

mindspore.ops.Cos

class mindspore.ops.Cos

```
prim = ops.Cos()  
out = prim(input)
```

等价于

```
ops.cos(input)
```

更多详情请查看: [*mindspore.ops.cos\(\)*](#) 。

mindspore.ops.Cosh

class mindspore.ops.Cosh

```
prim = ops.Cosh()  
out = prim(input)
```

等价于

```
ops.cosh(input)
```

更多详情请查看: [*mindspore.ops.cosh\(\)*](#) 。

mindspore.ops.Digamma

class mindspore.ops.Digamma

计算对数 gamma 函数在输入上的导数。

$$P(x) = grad(\ln(\gamma(x)))$$

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型为 *float16*、*float32* 或者 *float64*。

输出：

Tensor，和输入 *x* 具有相同的数据类型。

异常：

- **TypeError** - 如果输入 *x* 不是 Tensor。
- **TypeError** - 输入输入 *x* 的数据类型不是 *float16*、*float32* 或者 *float64*。

支持平台：

GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.5, 0.5, 9]).astype(np.float16))
>>> digamma = ops.Digamma()
>>> output = digamma(x)
>>> print(output)
[ 0.0365 -1.964  2.14  ]
```

mindspore.ops.Div

class mindspore.ops.Div

逐元素计算第一输入 Tensor 除以第二输入 Tensor 的商。

更多参考详见 `mindspore.ops.div()`。

说明：

- 当两个输入具有不同的 shape 时，它们的 shape 必须要能广播为一个共同的 shape。
- 两个输入不能同时为 bool 类型。[True, Tensor(True, bool_), Tensor(np.array([True]), bool_)] 等都为 bool 类型。
- 两个输入遵循隐式类型转换规则，使数据类型保持一致。

输入：

- **x** (Union[Tensor, number.Number, bool]) - 第一个输入，是一个 number.Number、bool 值或数据类型为 *number* 或 *bool_* 的 Tensor。

- **y** (Union[Tensor, number.Number, bool]) - 第二个输入, 是一个 number.Number、bool 值或数据类型为 number 或 bool_ 的 Tensor。

输出:

Tensor, shape 与输入 x, y 广播后的 shape 相同, 数据类型为两个输入中精度较高的类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1 :has same data type and shape of the two inputs
>>> x = Tensor(np.array([-4.0, 5.0, 6.0]), mindspore.float32)
>>> y = Tensor(np.array([3.0, 2.0, 3.0]), mindspore.float32)
>>> div = ops.Div()
>>> output = div(x, y)
>>> print(output)
[-1.3333334  2.5          2.          ]
>>> # case 2 : different data type and shape of the two inputs
>>> x = Tensor(np.array([-4.0, 5.0, 6.0]), mindspore.float32)
>>> y = Tensor(2, mindspore.int32)
>>> output = div(x, y)
>>> print(output)
[-2.  2.5  3.]
>>> print(output.dtype)
Float32
```

mindspore.ops.DivNoNan

class mindspore.ops.DivNoNan

对 $x1$ 和 $x2$ 逐元素执行安全除法, 如果 $x2$ 的元素为 0, 则返回 0。

$x1$ 和 $x2$ 的输入遵循隐式类型转换规则, 使数据类型一致。输入必须是两个 Tensor 或一个 Tensor 和一个 Scalar。当输入是两个 Tensor 时, 它们的 dtype 不能同时是 bool 类型的, 它们的 shape 可以广播。当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常数。

$$output_i = \begin{cases} 0, & \text{if } x2_i = 0 \\ x1_i/x2_i, & \text{if } x2_i \neq 0 \end{cases}$$

输入:

- **x1** (Union[Tensor, number.Number, bool]) - 第一个输入是 number.Number、bool 或者 Tensor, 数据类型为 number。
- **x2** (Union[Tensor, number.Number, bool]) - 当第一个输入是 bool 或数据类型为 number 或 bool_ 的 Tensor 时, 第二个输入是 number.Number 或 bool。当第一个输入是 Scalar 时, 第二个输入必须是数据类型为 number 或 bool_ 的 Tensor。

输出:

Tensor, shape 与广播后的 shape 相同, 数据类型为两个输入中精度较高或数位较高的类型。

异常:

- **TypeError** - 如果 $x1$ 和 $x2$ 不是 number.Number、bool 或 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x1 = Tensor(np.array([-1.0, 0., 1.0, 5.0, 6.0]), mindspore.float32)
>>> x2 = Tensor(np.array([0., 0., 0., 2.0, 3.0]), mindspore.float32)
>>> div_no_nan = ops.DivNoNaN()
>>> output = div_no_nan(x1, x2)
>>> print(output)
[0.  0.  0.  2.5 2. ]
```

mindspore.ops.Einsum

class mindspore.ops.**Einsum**(*equation*)

此算子使用爱因斯坦求和约定 (Einsum) 进行 Tensor 计算。支持对角线、约和、转置、矩阵乘、乘积、内积运算等。

参数:

- **equation** (str) - 属性, 表示要执行的计算。该值只能使用 letter([a-z][A-Z])、commas(,)、ellipsis(…) 和 arrow(->)。letter([a-z][A-Z]) 表示输入的 Tensor 的维度, commas(,) 表示 Tensor 维度之间的分隔符, ellipsis(…) 表示不关心的 Tensor 维度, arrow(->) 的左侧表示输入 Tensor, 右侧表示所需的输出维度。

输入:

- **x** (Union[tuple(Tensor), list(Tensor)]) - 用于计算的输入 Tensor, 输入必须是 Tensor 的 tuple/list。当输入只有一个 Tensor 时, 可以输入 (Tensor,)。支持数据类型 float16、float32、float64, 输入各 Tensor 的数据类型必须相同。

输出:

Tensor, shape 可以从方程中获得, 数据类型与输入 Tensor 相同。

异常:

- **TypeError** - 如果 *equation* 本身无效, 或者 *equation* 与输入 Tensor 不匹配。
- **TypeError** - 如果输入 Tensor 的数据类型不一致或者数据类型不是 float16、float32、float64。

支持平台:

GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.0, 2.0, 4.0]), mindspore.float32)
>>> equation = "i->"
>>> einsum = ops.Einsum(equation)
>>> output = einsum([x])
>>> print(output)
[7.]
>>>
>>> x = Tensor(np.array([1.0, 2.0, 4.0]), mindspore.float32)
>>> y = Tensor(np.array([2.0, 4.0, 3.0]), mindspore.float32)
```

(下页继续)

(续上页)

```

>>> equation = "i,i->i"
>>> einsum = ops.Einsum(equation)
>>> output = einsum((x, y))
>>> print(output)
[ 2.  8. 12.]
>>>
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> y = Tensor(np.array([[2.0, 3.0], [1.0, 2.0], [4.0, 5.0]]), mindspore.float32)
>>> equation = "ij,jk->ik"
>>> einsum = ops.Einsum(equation)
>>> output = einsum((x, y))
>>> print(output)
[[16. 22.]
 [37. 52.]]
>>>
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> equation = "ij->ji"
>>> einsum = ops.Einsum(equation)
>>> output = einsum((x,))
>>> print(output)
[[1. 4.]
 [2. 5.]
 [3. 6.]]
>>>
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> equation = "ij->j"
>>> einsum = ops.Einsum(equation)
>>> output = einsum((x,))
>>> print(output)
[5. 7. 9.]
>>>
>>> x = Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.float32)
>>> equation = "...->"
>>> einsum = ops.Einsum(equation)
>>> output = einsum((x,))
>>> print(output)
[21.]
>>>
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> y = Tensor(np.array([2.0, 4.0, 1.0]), mindspore.float32)
>>> equation = "j,i->ji"
>>> einsum = ops.Einsum(equation)
>>> output = einsum((x, y))
>>> print(output)
[[ 2.  4.  1.]
 [ 4.  8.  2.]
 [ 6. 12.  3.]]

```

mindspore.ops.Erf

class mindspore.ops.Erf

```
prim = ops.Erf()  
out = prim(input)
```

等价于

```
ops.erf(input)
```

更多详情请查看: [mindspore.ops.erf\(\)](#) 。

mindspore.ops.Erfc

class mindspore.ops.Erfc

```
prim = ops.Erfc()  
out = prim(input)
```

等价于

```
ops.erfc(input)
```

更多详情请查看: [mindspore.ops.erfc\(\)](#) 。

mindspore.ops.Erfinv

class mindspore.ops.Erfinv

```
prim = ops.Erfinv()  
out = prim(input)
```

等价于

```
ops.erfinv(input)
```

更多详情请查看: [mindspore.ops.erfinv\(\)](#) 。

mindspore.ops.Exp

class mindspore.ops.Exp

```
prim = ops.Exp()  
out = prim(input)
```

等价于

```
ops.exp(input)
```

更多详情请查看: [mindspore.ops.exp\(\)](#) 。

mindspore.ops.Expm1

class mindspore.ops.**Expm1**

```
prim = ops.Expm1()  
out = prim(input)
```

等价于

```
ops.expm1(input)
```

更多详情请查看: [*mindspore.ops.expm1\(\)*](#) 。

mindspore.ops.Floor

class mindspore.ops.**Floor**

```
prim = ops.Floor()  
out = prim(input)
```

等价于

```
ops.floor(input)
```

更多详情请查看: [*mindspore.ops.floor\(\)*](#) 。

mindspore.ops.FloorDiv

class mindspore.ops.**FloorDiv**

```
prim = ops.FloorDiv()  
out = prim(input, other)
```

等价于

```
ops.floor_divide(input, other)
```

更多详情请查看: [*mindspore.ops.floor_divide\(\)*](#) 。

mindspore.ops.FloorMod

class mindspore.ops.**FloorMod**

```
prim = ops.FloorMod()  
out = prim(x, y)
```

等价于

```
ops.floor_mod(x, y)
```

更多详情请查看: [*mindspore.ops.floor_mod\(\)*](#) 。

mindspore.ops.Geqr

class mindspore.ops.Geqr

```
prim = ops.Geqr()
out = prim(input)
```

等价于

```
ops.geqr(input)
```

更多详情请查看: [mindspore.ops.geqr\(\)](#)。

mindspore.ops.Imag

class mindspore.ops.Imag

返回包含输入 Tensor 的虚部。如果输入为实数，则返回零。

输入:

- **input** (Tensor) - 输入 Tensor。

输出:

Tensor, shape 与 *input* 相同。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import ops
>>> x = mindspore.tensor(1.3+0.4j, mindspore.complex64)
>>> imag = ops.Imag()
>>> output = imag(x)
>>> print(output)
0.4
```

mindspore.ops.Inv

class mindspore.ops.Inv

按元素计算输入 Tensor 的倒数。

更多参考详见[mindspore.ops.inv\(\)](#)。

输入:

- **x** (Tensor) - 任意维度的 Tensor。数据类型必须是 float16、float32 或 int32。

输出:

Tensor, shape 和数据类型与 *x* 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> inv = ops.Inv()
>>> x = Tensor(np.array([0.25, 0.4, 0.31, 0.52]), mindspore.float32)
>>> output = inv(x)
>>> print(output)
[4.          2.5          3.2258065 1.923077 ]
```

mindspore.ops.Invert

class mindspore.ops.Invert

翻转输入 Tensor 的所有元素。

详情请查看[*mindspore.ops.invert\(\)*](#)。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> invert = ops.Invert()
>>> x = Tensor(np.array([25, 4, 13, 9]), mindspore.int16)
>>> output = invert(x)
>>> print(output)
[-26 -5 -14 -10]
```

mindspore.ops.Lerp

class mindspore.ops.Lerp

```
prim = ops.Lerp()
out = prim(input, end, weight)
```

等价于

```
ops.lerp(input, end, weight)
```

更多详情请查看: [*mindspore.ops.lerp\(\)*](#)。

mindspore.ops.Log

class mindspore.ops.Log

```
prim = ops.Log()
out = prim(input)
```

等价于

```
ops.log(input)
```

更多详情请查看: [mindspore.ops.log\(\)](#)。

mindspore.ops.Log1p

class mindspore.ops.Log1p

```
prim = ops.Log1p()
out = prim(input)
```

等价于

```
ops.log1p(input)
```

更多详情请查看: [mindspore.ops.log1p\(\)](#)。

mindspore.ops.LogicalAnd

class mindspore.ops.LogicalAnd

逐元素进行两个 Tensor 的逻辑与运算。

详情请查看[mindspore.ops.logical_and\(\)](#)。

输入:

- **x** (Union[Tensor, bool]) - 第一个输入是 bool 或数据类型可被隐式转换为 bool 的 Tensor。
- **y** (Union[Tensor, bool]) - 当第一个输入是 Tensor 的时候, 第二个输入是 bool 或者数据类型可被隐式转换为 bool 的 Tensor。

输出:

Tensor, 其 shape 与 *x* 和 *y* 广播后的 shape 相同, 数据类型为 bool。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([True, False, True]), mindspore.bool_)
>>> y = Tensor(np.array([True, True, False]), mindspore.bool_)
>>> logical_and = ops.LogicalAnd()
>>> output = logical_and(x, y)
>>> print(output)
```

(下页继续)

(续上页)

```
[ True False False]
>>> x = Tensor(1, mindspore.bool_)
>>> y = Tensor(0, mindspore.bool_)
>>> output = ops.LogicalAnd()(x, y)
>>> print(output)
False
>>> x = True
>>> y = Tensor(0, mindspore.bool_)
>>> output = ops.LogicalAnd()(x, y)
>>> print(output)
False
>>> x = True
>>> y = Tensor(np.array([True, False]), mindspore.bool_)
>>> output = ops.LogicalAnd()(x, y)
>>> print(output)
[True False]
```

mindspore.ops.LogicalNot

class mindspore.ops.LogicalNot

逐元素计算一个 Tensor 的逻辑非运算。

详情请查看[*mindspore.ops.logical_not\(\)*](#)。

输入：

- **x** (Tensor) - 输入 Tensor。

输出：

Tensor, shape 与 *x* 相同, 数据类型为 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([True, False, True]), mindspore.bool_)
>>> logical_not = ops.LogicalNot()
>>> output = logical_not(x)
>>> print(output)
[False True False]
```


mindspore.ops.LogicalOr

class mindspore.ops.LogicalOr

逐元素计算两个 Tensor 的逻辑或运算。

详情请查看[mindspore.ops.logical_or\(\)](#)。

输入：

- **x** (Union[Tensor, bool]) - 第一个输入是 bool 或数据类型可被隐式转换为 bool 的 Tensor。
- **y** (Union[Tensor, bool]) - 当第一个输入是 Tensor 的时候，第二个输入是 bool 或者数据类型可被隐式转换为 bool 的 Tensor。

输出：

Tensor，其 shape 与 *x* 和 *y* 广播后的 shape 相同，数据类型为 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([True, False, True]), mindspore.bool_)
>>> y = Tensor(np.array([True, True, False]), mindspore.bool_)
>>> logical_or = ops.LogicalOr()
>>> output = logical_or(x, y)
>>> print(output)
[ True  True  True]
>>> x = Tensor(1, mindspore.bool_)
>>> y = Tensor(0, mindspore.bool_)
>>> output = ops.LogicalOr()(x, y)
>>> print(output)
True
>>> x = True
>>> y = Tensor(0, mindspore.bool_)
>>> output = ops.LogicalOr()(x, y)
>>> print(output)
True
>>> x = True
>>> y = Tensor(np.array([True, False]), mindspore.bool_)
>>> output = ops.LogicalOr()(x, y)
>>> print(output)
[True True]
```

mindspore.ops.LogicalXor

class mindspore.ops.LogicalXor

逐元素计算两个 Tensor 的逻辑异或运算。

警告：这是一个实验性 API，后续可能修改或删除。

详情请查看[mindspore.ops.logical_xor\(\)](#)。

输入：

- **x** (Union[Tensor, bool]) - 第一个输入是 bool 或数据类型可被隐式转换为 bool 的 Tensor。
- **y** (Union[Tensor, bool]) - 当第一个输入是 Tensor 的时候，第二个输入是 bool 或者数据类型可被隐式转换为 bool 的 Tensor。

输出：

Tensor，其 shape 与 *x* 和 *y* 广播后的 shape 相同，数据类型为 bool。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([True, False, True]), mindspore.bool_)
>>> y = Tensor(np.array([True, True, False]), mindspore.bool_)
>>> logical_xor = ops.LogicalXor()
>>> output = logical_xor(x, y)
>>> print(output)
[ False True True]
>>> x = Tensor(1, mindspore.bool_)
>>> y = Tensor(0, mindspore.bool_)
>>> output = ops.LogicalXor()(x, y)
>>> print(output)
True
>>> x = True
>>> y = Tensor(0, mindspore.bool_)
>>> output = ops.LogicalXor()(x, y)
>>> print(output)
True
>>> x = True
>>> y = Tensor(np.array([True, False]), mindspore.bool_)
>>> output = ops.LogicalXor()(x, y)
>>> print(output)
[False True]
```

mindspore.ops.Logit

class mindspore.ops.Logit (eps=-1.0)

逐元素计算 Tensor 的 logit 值。*x* 中的元素被截断到范围 [eps, 1-eps] 内。

警告： 这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.logit()`。

参数：

- **eps** (float, 可选) - epsilon 值。输入的数值界限被定义 [eps, 1-eps]。默认值：-1.0。

输入：

- **x** (Tensor) - Tensor 输入，其数据类型为 float16、float32 或 float64。

输出：

Tensor，具有与 x 相同的 shape。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([0.1, 0.2, 0.3]).astype(np.float32))
>>> op = ops.Logit(eps=1e-5)
>>> output = op(x)
>>> print(output)
[-2.1972246 -1.3862944 -0.8472978]
```

mindspore.ops.Mod

class mindspore.ops.Mod

将第一个输入 Tensor 逐元素除以第二个输入 Tensor，并取余。

x 和 y 的输入遵循隐式类型转换规则，使数据类型一致。输入必须是两个 Tensor 或一个 Tensor 和一个 Scalar。当输入是两个 Tensor 时，两个数据类型都不能是 bool，它们的 shape 可以广播。当输入是一个 Tensor 和一个 Scalar 时，Scalar 只能是一个常量。

$$out_i = x_i \% y_i$$

警告：

- 输入数据不支持 0。
- 当输出包含的元素个数超过 2048 时，该算子不能保证双千分之一的精度要求。
- 由于架构的差异，在 NPU 和 CPU 上生成的结果可能不一致。
- 如果 shape 表示为 $(D1, D2, ..., Dn)$ ，则 $D1 * D2 * ... * Dn \leq 1000000, n \leq 8$ 。

输入：

- **x** (Union[Tensor, numbers.Number, bool]) - 第一个输入是数值型、bool，或数据类型为数值型的 Tensor。
- **y** (Union[Tensor, numbers.Number, bool]) - 当第一个输入是 Tensor 时，第二个输入可以是数值型、bool，或数据类型为数值型的 Tensor。当第一个输入是数值型或 bool 时，第二个输入必须是数据类型为数值型的 Tensor。

输出：

Tensor，shape 与广播后的 shape 相同，数据类型为两个输入中精度较高或数据类型相对最高的类型。

异常：

- **TypeError** - 如果 x 和 y 都不是以下之一：Tensor、数值型、bool。
- **TypeError** - 如果 x 和 y 都不是 Tensor。
- **ValueError** - 如果 x 和 y 的 shape 不能相互广播。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([-4.0, 5.0, 6.0]), mindspore.float32)
>>> y = Tensor(np.array([3.0, 2.0, 3.0]), mindspore.float32)
>>> mod = ops.Mod()
>>> output = mod(x, y)
>>> print(output)
[-1.  1.  0.]
```

mindspore.ops.Mul

class mindspore.ops.Mul

```
prim = ops.Mul()
out = prim(input, other)
```

等价于

```
ops.mul(input, other)
```

更多详情请查看: `mindspore.ops.mul()`。

mindspore.ops.MulNoNan

class mindspore.ops.MulNoNan

逐元素计算输入乘积。如果 y 为零, 无论 x 取何值, 它都将返回 0。

x 和 y 的输入遵循隐式类型转换规则, 使数据类型一致。输入必须是两个 Tensor 或一个 Tensor 和一个 Scalar。当输入是两个 Tensor 时, 它们的 shape 可以被广播。当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常量。

$$output_{ij} = \begin{cases} 0, & y_{ij} = 0; \\ x_{ij} * y_{ij}, & otherwise. \end{cases}$$

说明: x 和 y 的 shape 应该相同, 或者可以广播。如果 y 是 NaN 或为无限的, 并且 x 是 0, 结果将是 NaN。

输入:

- **x** (Union[Tensor]) - 第一个输入是 Tensor, 其数据类型为 int32、int64、float16、float32、float64、complex64、complex128 或 Scalar。
- **y** (Union[Tensor]) - 第二个输入是 Tensor, 其数据类型为 int32、int64、float16、float32、float64、complex64、complex128 或 Scalar。

输出:

Tensor, shape 与输入 Tensor 广播后的 shape 相同, 数据类型是两个输入 Tensor 数据类型中精度较高的类型。

异常:

- **TypeError** - 如果 x 和 y 都不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1 : same data type and shape of two inputs, there are some 0 in y.
>>> x = Tensor(np.array([[[-1.0, 6.0, np.inf], [np.nan, -7.0, 4.0]]], mindspore.float32)
>>> y = Tensor(np.array([[[-1.0, 4.0, 0], [0, -3.0, 1.0]]], mindspore.float32)
>>> mul_no_nan = ops.MulNoNaN()
>>> output = mul_no_nan(x, y)
>>> print(output)
[[ 1. 24. 0.]
 [ 0. 21. 4.]]
>>> # case 2 : the shape of two inputs is same, there are some 0 in x, y.
>>> x = Tensor(np.array([[[-1.0, 6.0, 0], [0, np.nan, 4.0]]], mindspore.float32)
>>> y = Tensor(np.array([[[-1.0, 4.0, np.inf], [np.nan, 0, 1.0]]], mindspore.float32)
>>> output = mul_no_nan(x, y)
>>> print(output)
[[ 1. 24. nan]
 [nan 0. 4.]]
>>> print(output.dtype)
Float32
>>> # case 3 : the y is a scalar.
>>> x = Tensor(np.array([[[-1.0, 6.0, 0], [0, np.nan, 4.0]]], mindspore.float32)
>>> y = Tensor(0, mindspore.float32)
>>> output = mul_no_nan(x, y)
>>> print(output)
[[0. 0. 0.]
 [0. 0. 0.]]
```

mindspore.ops.Neg

class mindspore.ops.Neg

```
prim = ops.Neg()
out = prim(input)
```

等价于

```
ops.neg(input)
```

更多详情请查看: `mindspore.ops.neg()`。

mindspore.ops.NextAfter

class mindspore.ops.NextAfter

```
prim = ops.NextAfter()
out = prim(input, other)
```

等价于

```
ops.nextafter(input, other)
```

更多详情请查看：[mindspore.ops.nextafter\(\)](#)。

mindspore.ops.Polar

class mindspore.ops.Polar

将极坐标转化为笛卡尔坐标。

返回一个复数 Tensor，其元素是由输入极坐标构造的笛卡尔坐标。其中极坐标由极径 *abs* 和极角 *angle* 给定。

更多细节请参考[mindspore.ops.polar\(\)](#)。

$$y_i = abs_i * \cos(angle_i) + abs_i * \sin(angle_i) * j$$

警告： 这是一个实验性 API，后续可能修改或删除。

输入：

- **abs** (Tensor, float) - 极径，其输入 shape 可以是任意维度，其数据类型须为：float32。
- **angle** (Tensor, float) - 极角，其 shape 和数据类型与 *abs* 一致。

输出：

Tensor，其 shape 与 *abs* 一致，dtype 是 complex64。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> x1 = Tensor(np.array([1, 2]), mindspore.float32)
>>> x2 = Tensor(np.array([3, 4]), mindspore.float32)
>>> op_polar = ops.Polar()
>>> output = op_polar(x1, x2)
>>> print(output)
[-0.9899925 +0.14112002j -1.3072872-1.5136049j]
>>> x1 = Tensor(2.1, mindspore.float32)
>>> x2 = Tensor(2.1, mindspore.float32)
>>> output = op_polar(x1, x2)
>>> print(output)
(-1.0601765+1.8127397j)
```

mindspore.ops.Polygamma

class mindspore.ops.Polygamma

计算关于 x 的多伽马函数的 a 阶导数。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.polygamma()`。

输入：

- **a** (Tensor) - 多伽马函数求导的阶数， a 的 shape 为 $()$ ，支持的数据类型为 `int32` 和 `int64`。
- **x** (Tensor) - 用于计算多伽马函数的 a 阶导数的 Tensor，支持的数据类型为：`float16`、`float32` 和 `float64`。

输出：

Tensor，数据类型与 x 一致。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.0, -0.5]), mindspore.float32)
>>> a = Tensor(np.array(1), mindspore.int64)
>>> polygamma = ops.Polygamma()
>>> output = polygamma(a, x)
>>> print(output)
[1.644934 8.934802]
>>> a = Tensor(np.array(2), mindspore.int64)
>>> output = polygamma(a, x)
>>> print(output)
[-2.404114 -0.8287967]
>>> a = Tensor(np.array(3), mindspore.int64)
>>> output = polygamma(a, x)
>>> print(output)
[ 6.4939404 193.40909 ]
>>> a = Tensor(np.array(4), mindspore.int64)
>>> output = polygamma(a, x)
>>> print(output)
[-24.886265 -3.4742498]
```

mindspore.ops.Pow

class mindspore.ops.Pow

```
prim = ops.Pow()
out = prim(input, exponent)
```

等价于

```
ops.pow(input, exponent)
```

更多详情请查看: `mindspore.ops.pow()`。

mindspore.ops.Real

class mindspore.ops.Real

```
prim = ops.Real()
out = prim(input)
```

等价于

```
ops.real(input)
```

更多详情请查看: `mindspore.ops.real()`。

mindspore.ops.RealDiv

class mindspore.ops.RealDiv

第一个输入 Tensor 元素为分子，第二个输入 Tensor 元素为分母，逐元素进行浮点型除法运算。

更多参考详见`mindspore.ops.div()`。

输入:

- **x** (Union[Tensor, Number, bool]) - 第一个输入，为数值型，或 bool，或数据类型为数值型或 bool 的 Tensor。
- **y** (Union[Tensor, Number, bool]) - 第二个输入，为数值型，或 bool，或数据类型为数值型或 bool 的 Tensor。

输出:

Tensor，输出的 shape 与广播后的 shape 相同，数据类型取两个输入中精度较高或数字较高的。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> y = Tensor(np.array([4.0, 5.0, 6.0]), mindspore.float32)
>>> realdiv = ops.RealDiv()
>>> output = realdiv(x, y)
>>> print(output)
[0.25 0.4  0.5 ]
```


mindspore.ops.Reciprocal

class mindspore.ops.Reciprocal

返回输入 Tensor 的倒数。

$$out_i = \frac{1}{x_i}$$

输入:

- **x** (Tensor) - 输入 Tensor。

输出:

Tensor, shape 与 x 相同。

异常:

- **TypeError** - 如果 x 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.0, 2.0, 4.0]), mindspore.float32)
>>> reciprocal = ops.Reciprocal()
>>> output = reciprocal(x)
>>> print(output)
[1.  0.5 0.25]
```

mindspore.ops.Rint

class mindspore.ops.Rint

逐元素计算最接近输入数据的整数。

输入:

- **input_x** (Tensor) - 任意维度输入 Tensor, 数据必须是 float16、float32 或者 float64。

输出:

Tensor, shape 和数据类型与 *input_x* 相同。

异常:

- **TypeError** - 如果 *input_x* 的数据类型不是 float16、float32、float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([-1.6, -0.1, 1.5, 2.0]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> op = ops.Rint()
>>> output = op(input_x)
>>> print(output)
[-2.  0.  2.  2.]
>>> input_x = Tensor(np.array([[ -2.0,  -1.9,  -1.8,  -1.7,  -1.6],
...                           [ -2.0,  -1.9,  -1.8,  -1.7,  -1.6]]), mindspore.float32)
>>> output = op(input_x)
>>> print(output)
[[-2. -2. -2. -2. -2.]
 [-2. -2. -2. -2. -2.]]
```

mindspore.ops.Round

class mindspore.ops.Round

对输入数据进行四舍五入，得到最接近的整数数值。

$$out_i \approx input_i$$

说明：Ascend 平台支持的输入数据类型包括 bfloat16（Atlas 训练系列产品不支持）、float16、float32、float64、int32、int64。

输入：

- **input** (Tensor) - 输入 Tensor。
- **decimals** (int, 可选) - 要舍入到的小数位小数（默认值：0）。如果为负数，则指定小数点左侧的位数。支持输入单元素 Tensor 转换为 int。*input* 类型为 int32 或 int64 时，*decimals* 参数值必须为 0。

输出：

Tensor, shape 和数据类型与输入 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **RuntimeError** - *input* 为 int32 或 int64 类型时，*decimals* 值不为 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([0.8, 1.5, 2.3, 2.5, -4.5]), mindspore.float32)
>>> round = ops.Round()
>>> output = round(input)
>>> print(output)
[ 1.  2.  2.  2. -4.]
```

mindspore.ops.Rsqrt

class mindspore.ops.Rsqrt

```
prim = ops.Rsqrt()  
out = prim(input)
```

等价于

```
ops.rsqrt(input)
```

更多详情请查看: [mindspore.ops.rsqrt\(\)](#)。

mindspore.ops.Sign

class mindspore.ops.Sign

```
prim = ops.Sign()  
out = prim(input)
```

等价于

```
ops.sign(input)
```

更多详情请查看: [mindspore.ops.sign\(\)](#)。

mindspore.ops.Sin

class mindspore.ops.Sin

```
prim = ops.Sin()  
out = prim(input)
```

等价于

```
ops.sin(input)
```

更多详情请查看: [mindspore.ops.sin\(\)](#)。

mindspore.ops.Sinc

class mindspore.ops.Sinc

```
prim = ops.Sinc()  
out = prim(input)
```

等价于

```
ops.sinc(input)
```

更多详情请查看: [mindspore.ops.sinc\(\)](#)。

mindspore.ops.Sinh

class mindspore.ops.Sinh

```
prim = ops.Sinh()
out = prim(input)
```

等价于

```
ops.sinh(input)
```

更多详情请查看: [*mindspore.ops.sinh\(\)*](#) 。

mindspore.ops.Sqrt

class mindspore.ops.Sqrt

```
prim = ops.Sqrt()
out = prim(x)
```

等价于

```
ops.sqrt(x)
```

更多详情请查看: [*mindspore.ops.sqrt\(\)*](#) 。

mindspore.ops.Square

class mindspore.ops.Square

```
prim = ops.Square()
out = prim(input)
```

等价于

```
ops.square(input)
```

更多详情请查看: [*mindspore.ops.square\(\)*](#) 。

mindspore.ops.SquaredDifference

class mindspore.ops.SquaredDifference

第一个输入 Tensor 逐元素减去第二个输入 Tensor, 并返回其平方。

x 和 y 的输入遵循隐式类型转换规则, 使数据类型一致。输入必须是两个 Tensor 或一个 Tensor 和一个 Scalar。当输入是两个 Tensor 时, 它们的数据类型不能同时为 bool 类型, 并且它们的 shape 可以广播。当输入是一个 Tensor 和一个 Scalar 时, Scalar 只能是一个常量。

$$out_i = (x_i - y_i) * (x_i - y_i) = (x_i - y_i)^2$$

输入:

- **x** (Union[Tensor, Number, bool]) - 第一个输入, 为数值型, 或为 bool, 或为 Tensor。

- **y** (Union[Tensor, Number, bool]) - 第二个输入，通常为数值型，或为 Tensor，当第一个输入是 Tensor 时或为 bool。

输出：

Tensor，shape 与广播后的 shape 相同，数据类型为两个输入中精度较高或数字较高的类型。

异常：

- **TypeError** - 如果 x 和 y 不是数值型、bool 或 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1.0, 2.0, 3.0]), mindspore.float32)
>>> y = Tensor(np.array([2.0, 4.0, 6.0]), mindspore.float32)
>>> squared_difference = ops.SquaredDifference()
>>> output = squared_difference(x, y)
>>> print(output)
[1. 4. 9.]
```

mindspore.ops.SquareSumAll

class mindspore.ops.SquareSumAll

计算输入 Tensor 的平方和。

$$\begin{cases} out_x = \sum_0^N (x_i)^2 \\ out_y = \sum_0^N (y_i)^2 \end{cases}$$

说明：SquareSumAll 只支持 float16 和 float32 类型的输入值。

输入：

- **x** (Tensor) - SquareSumAll 的输入，其数据类型为数值型，shape: $(N, *)$ ，其中 $*$ 表示任意数量的附加维度。
- **y** (Tensor) - 数据类型和 shape 与 x 相同。

输出：

- **output_x** (Tensor) - 数据类型与 x 相同。
- **output_y** (Tensor) - 数据类型与 x 相同。

异常：

- **TypeError** - 如果 x 或 y 不是 Tensor。
- **ValueError** - 如果 x 与 y 的 shape 不一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import ops
>>> from mindspore import Tensor
>>> x = Tensor(np.array([0, 0, 2, 0]), mindspore.float32)
>>> y = Tensor(np.array([0, 0, 2, 4]), mindspore.float32)
>>> square_sum_all = ops.SquareSumAll()
>>> output = square_sum_all(x, y)
>>> print(output)
(Tensor(shape=[], dtype=Float32, value= 4),
 Tensor(shape=[], dtype=Float32, value= 20))
```

mindspore.ops.Sub

class mindspore.ops.Sub

逐元素用第一个输入 Tensor 减去第二个输入 Tensor。

更多参考详见 `mindspore.ops.sub()`。

说明:

- 当两个输入具有不同的 shape 时，它们的 shape 必须要能广播为一个共同的 shape。
- 两个输入不能同时为 bool 类型。[True, Tensor(True, bool_), Tensor(np.array([True]), bool_)] 等都为 bool 类型。
- 两个输入遵循隐式类型转换规则，使数据类型保持一致。

输入:

- **x** (Union[Tensor, number.Number, bool]) - 第一个输入，是一个 number.Number、bool 值或数据类型为 `number` 或 `bool_` 的 Tensor。
- **y** (Union[Tensor, number.Number, bool]) - 第二个输入，当第一个输入是 Tensor 时，第二个输入应该是一个 number.Number 或 bool 值，或数据类型为 `number` 或 `bool` 的 Tensor。

输出:

Tensor, shape 与两个输入广播后的 shape 相同，数据类型为两个输入中精度较高的类型。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1, 2, 3]), mindspore.int32)
>>> y = Tensor(np.array([4, 5, 6]), mindspore.int32)
>>> sub = ops.Sub()
>>> output = sub(x, y)
>>> print(output)
[-3 -3 -3]
```

mindspore.ops.Tan

class mindspore.ops.Tan

逐元素计算输入元素的正切值。

更多参考详见[mindspore.ops.tan\(\)](#)。

输入：

- **input** (Tensor) - 任意维度的输入 Tensor。

输出：

Tensor，数据 shape 与 *input* 相同。当输入类型为 bool、int8、uint8、int16、int32、int64 时，返回值类型为 float32。否则，返回值类型与输入类型相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> tan = ops.Tan()
>>> x = Tensor(np.array([-1.0, 0.0, 1.0]), mindspore.float32)
>>> output = tan(x)
>>> print(output)
[-1.5574081 0. 1.5574081]
```

mindspore.ops.Trunc

class mindspore.ops.Trunc

```
prim = ops.Trunc()
out = prim(input)
```

等价于

```
ops.trunc(input)
```

更多详情请查看：[mindspore.ops.trunc\(\)](#)。

mindspore.ops.TruncateDiv

class mindspore.ops.TruncateDiv

将第一个输入 Tensor 与第二个输入 Tensor 逐元素相除，结果将向 0 取整。相当于 C 语言风格的整数除法。

输入 *x* 和 *y* 应遵循隐式类型转换规则，以确保数据类型一致。当输入为两个 Tensor 时，数据类型不能同时为 bool 类型。当输入是一个 Tensor 和一个标量时，标量只能是一个常数。

说明：支持 shape 广播。

输入：

- **x** (Union[Tensor, Number, bool]) - Number 或 bool 类型的 Tensor。

- **y** (Union[Tensor, Number, bool]) - Number 或 bool 类型的 Tensor。 x 和 y 不能同时为 bool 类型。

输出：

Tensor, shape 为输入广播后的 shape, 数据类型为两个输入中精度较高的输入的类型。

异常：

- **TypeError** - x 和 y 数据类型不是以下之一：Tensor、Number 或 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([2, 4, -1]), mindspore.int32)
>>> y = Tensor(np.array([3, 3, 3]), mindspore.int32)
>>> truncate_div = ops.TruncateDiv()
>>> output = truncate_div(x, y)
>>> print(output)
[0 1 0]
```

mindspore.ops.TruncateMod

class mindspore.ops.TruncateMod

逐元素取模。

输入 x 和 y 应能遵循隐式类型转换规则使数据类型一致。当输入为两个 Tensor 时，它们必须能够广播到一个共同的 shape，且数据类型不能同时为 bool 类型。当输入是一个 Tensor 和一个标量时，标量只能是一个常数。

警告：

- 输入数值不能为 0。
- 当输入含有超过 2048 个元素，该操作不能保证千分之二的精度要求。
- 由于架构不同，该算子在 NPU 和 CPU 上的计算结果可能不一致。
- 若 shape 为 (D1、D2...、Dn)，则 $D1 * D2 * \dots * Dn \leq 1000000$ ， $n \leq 8$ 。

输入：

- **x** (Union[Tensor, numbers.Number, bool]) - Number 或 bool 类型的 Tensor。
- **y** (Union[Tensor, numbers.Number, bool]) - Number 或 bool 类型的 Tensor。

输出：

Tensor, shape 为输入进行广播后的 shape, 数据类型为两个输入中精度较高的输入的类型。

异常：

- **TypeError** - x 和 y 数据类型不是以下之一：Tensor、Number、bool。
- **TypeError** - x 和 y 均不是 Tensor。
- **ValueError** - x 和 y 的 shape 无法进行广播转换。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([2, 4, -1]), mindspore.int32)
>>> y = Tensor(np.array([3, 3, 3]), mindspore.int32)
>>> truncate_mod = ops.TruncateMod()
>>> output = truncate_mod(x, y)
>>> print(output)
[ 2  1 -1]
```

mindspore.ops.Xdivy

class mindspore.ops.Xdivy

将第一个输入 Tensor 除以第二个输入 Tensor。当 x 为零时，则返回零。

x 和 y 的输入遵循隐式类型转换规则，使数据类型一致。当输入是两个 Tensor 时，它们的数据类型不能同时 bool，它们的 shape 可以广播。当某个输入是 Scalar 时，Scalar 只能是一个常量。

输入：

- x (Union[Tensor, Number, bool]) - float、complex 或 bool 类型的 Tensor。
- y (Union[Tensor, Number, bool]) - float、complex 或 bool 类型的 Tensor。 x 和 y 不能同时为 bool 类型。

输出：

Tensor，shape 与广播后的 shape 相同，数据类型为两个输入中精度较高或数据类型较高的类型。

异常：

- **TypeError** - 如果 x 和 y 不是以下之一：Tensor、Number、bool。
- **TypeError** - 如果 x 和 y 的数据类型不是 float16、float32、float64、complex64、complex128、bool。
- **ValueError** - 如果 x 不能广播至 y 的 shape。
- **RuntimeError** - 如果 Parameter 的 x 、 y 需要进行数据类型转换，但是 Parameter 是不支持数据类型转换。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([2, 4, -1]), mindspore.float32)
>>> y = Tensor(np.array([2, 2, 2]), mindspore.float32)
>>> xdivy = ops.Xdivy()
>>> output = xdivy(x, y)
>>> print(output)
[ 1.  2. -0.5]
```

mindspore.ops.Xlogy

class mindspore.ops.Xlogy

计算第一个输入 Tensor 乘以第二个输入 Tensor 的对数。当 *input* 为零时，则返回零。

$$out_i = input_i \ln other_i$$

input 和 *other* 的输入遵循隐式类型转换规则，使数据类型一致。

输入：

- **input** (Tensor) - 第一个输入是数据类型为 `number` 或 `bool_` 的 Tensor。
- **other** (Tensor) - 第二个输入是数据类型为 `number` 或 `bool_` 的 Tensor。

输出：

- **y** (Tensor) - shape 是 *input* 和 *other* 广播后的 shape，数据类型为两个输入中精度较高或数值较高的类型。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *other* 不是 Tensor。
- **ValueError** - 如果 *input* 和 *other* 的 shape 不可广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.array([-5, 0, 4]), mindspore.float32)
>>> other = Tensor(np.array([2, 2, 2]), mindspore.float32)
>>> op = ops.Xlogy()
>>> output = op(input, other)
>>> print(output)
[-3.465736  0.          2.7725887]
```

mindspore.ops.Zeta

class mindspore.ops.Zeta

计算输入 Tensor 的 Hurwitz zeta 函数 $\zeta(x, q)$ 值。

$$\zeta(x, q) = \sum_{n=0}^{\infty} (q + n)^{-x}$$

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - Tensor，数据类型为：float32、float64。
- **q** (Tensor) - Tensor，数据类型与 *x* 一致。

输出：

Tensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - x 或 q 不是 Tensor。
- **TypeError** - x 的数据类型既不是 float32 也不是 float64。
- **TypeError** - q 的数据类型既不是 float32 也不是 float64。
- **ValueError** - x 和 q 的 shape 不同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([10.]), mindspore.float32)
>>> q = Tensor(np.array([1.]), mindspore.float32)
>>> zeta = ops.Zeta()
>>> z = zeta(x, q)
>>> print(z)
[1.0009946]
```

20.3.9 Reduction 算子

接口名	概述	支持平台
<code>mindspore.ops.Argmax</code>	返回输入 Tensor 在指定轴上的最大值索引。	Ascend GPU CPU
<code>mindspore.ops.ArgMaxWithValue</code>	在给定轴上计算输入 Tensor 的最大值，并且返回最大值和索引。	Ascend GPU CPU
<code>mindspore.ops.Argmin</code>	返回输入 Tensor 在指定轴上的最小值索引。	Ascend GPU CPU
<code>mindspore.ops.ArgMinWithValue</code>	在给定轴上计算输入 Tensor 的最小值，并且返回最小值和索引。	Ascend GPU CPU
<code>mindspore.ops.Median</code>	输出 Tensor 指定维度 <i>axis</i> 上的中值与其对应的索引。	GPU CPU
<code>mindspore.ops.ReduceAll</code>	更多详情请查看： <code>mindspore.ops.all()</code> 。	Ascend GPU CPU
<code>mindspore.ops.ReduceAny</code>	默认情况下，通过对指定维度所有元素进行逻辑或运算来移除该维度。	Ascend GPU CPU
<code>mindspore.ops.ReduceMax</code>	默认情况下，使用指定维度的最大值代替该维度的其他元素，以移除该维度。	Ascend GPU CPU
<code>mindspore.ops.ReduceMean</code>	默认情况下，使用指定维度的平均值代替该维度的其他元素，以移除该维度。	Ascend GPU CPU
<code>mindspore.ops.ReduceMin</code>	默认情况下，使用指定维度的最小值代替该维度的其他元素，以移除该维度。	Ascend GPU CPU
<code>mindspore.ops.ReduceProd</code>	默认情况下，使用指定维度所有元素的乘积代替该维度的其他元素，以移除该维度。	Ascend GPU CPU
<code>mindspore.ops.ReduceSum</code>	默认情况下，输出 Tensor 各维度上的和，以达到对所有维度进行归约的目的。	Ascend GPU CPU

mindspore.ops.Argmax

class mindspore.ops.Argmax (axis=-1, output_type=mstype.int32)

返回输入 Tensor 在指定轴上的最大值索引。

更多参考详见 `mindspore.ops.argmax()`。

参数：

- **axis** (int) - 指定 Argmax 计算轴。默认值：-1。
- **output_type** (`mindspore.dtype`) - 指定输出数据类型。可选值：mstype.int32、mstype.int64。默认值：mstype.int32。

输入：

- **input_x** (Tensor) - Argmax 的输入，shape：(N,*)，其中 * 表示任意数量的附加维度。

输出：

Tensor，输出为指定轴上输入 Tensor 最大值的索引。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[1, 20, 5], [67, 8, 9], [130, 24, 15]]).astype(np.float32))
>>> output = ops.Argmax(output_type=mstype.int32)(input_x)
>>> print(output)
[1 0 0]
```

mindspore.ops.ArgMaxWithValue

class mindspore.ops.ArgMaxWithValue (axis=0, keep_dims=False)

在给定轴上计算输入 Tensor 的最大值，并且返回最大值和索引。

说明：在 auto_parallel 和 semi_auto_parallel 模式下，不能使用第一个输出索引。

警告：

- 如果有多个最大值，则取第一个最大值的索引。
- **axis** 的取值范围为 [-dims, dims - 1]。“dims”为 *input* 的维度长度。

参考：`mindspore.ops.max()`。

参数：

- **axis** (int) - 指定计算维度。默认值：0。
- **keep_dims** (bool) - 表示是否减少维度，如果为 True，输出将与输入保持相同的维度；如果为 False，输出将减少维度。默认值：False。

输入：

- **input** (Tensor) - 输入任意维度的 Tensor。将输入 Tensor 的 shape 设为 $(input_1, input_2, \dots, input_N)$ 。

输出：

tuple (Tensor)，表示 2 个 Tensor 组成的 tuple，包含对应的索引和输入 Tensor 的最大值。

- **index** (Tensor) - 输入 Tensor 最大值的索引，数据类型为 int64。如果 *keep_dims* 为 True，则输出 Tensor 的 shape 为 $(input_1, input_2, \dots, input_{axis-1}, 1, input_{axis+1}, \dots, input_N)$ 。否则，shape 为 $(input_1, input_2, \dots, input_{axis-1}, input_{axis+1}, \dots, input_N)$ 。
- **values** (Tensor) - 输入 Tensor 的最大值，其 shape 与 *index* 相同，数据类型与 *input* 相同。

异常：

- **TypeError** - *keep_dims* 不是 bool。
- **TypeError** - *axis* 不是 int。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> index, output = ops.ArgMaxWithValue()(input_x)
>>> print(index, output)
3 0.7
>>> index, output = ops.ArgMaxWithValue(keep_dims=True)(input_x)
>>> print(index, output)
[3] [0.7]
```

mindspore.ops.Argmin

class mindspore.ops.**Argmin** (*axis=-1, output_type=mstype.int32*)

返回输入 Tensor 在指定轴上的最小值索引。

如果输入 Tensor 的 shape 为 $(x_1, \dots, x_N)$ ，则输出 Tensor 的 shape 为 $(x_1, \dots, x_{axis-1}, x_{axis+1}, \dots, x_N)$ 。

参数：

- **axis** (int, 可选) - 指定 Argmin 的计算轴。默认值：-1。
- **output_type** (*mindspore.dtype*, 可选) - 指定输出数据类型。可选值：mstype.int32 和 mstype.int64。默认值：mstype.int32。

输入：

- **input_x** (Tensor) - Argmin 的输入，shape 为 $(N, *)$ ，其中 * 表示任意数量的附加维度。

输出：

Tensor，表示指定轴上输入 Tensor 最小值的索引。

异常：

- **TypeError** - *axis* 不是 int。
- **TypeError** - *output_type* 既不是 int32 也不是 int64。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([2.0, 3.1, 1.2]), mindspore.float32)
>>> index = ops.Argmin()(input_x)
>>> print(index)
2
```

mindspore.ops.ArgMinWithValue

class mindspore.ops.ArgMinWithValue (*axis=0, keep_dims=False*)

在给定轴上计算输入 Tensor 的最小值，并且返回最小值和索引。

说明：在 auto_parallel 和 semi_auto_parallel 模式下，不能使用第一个输出索引。

警告：

- 如果有多个最小值，则取第一个最小值的索引。
- *axis* 的取值范围为 $[-\text{dims}, \text{dims} - 1]$ 。“dims”为 *input* 的维度长度。

参考 `mindspore.ops.min()`。

参数：

- **axis** (int) - 指定计算维度。默认值：0。
- **keep_dims** (bool) - 表示是否减少维度。如果为 True，则输出维度和输入维度相同。如果为 False，则减少输出维度。默认值：False。

输入：

- **input** (Tensor) - 输入任意维度的 Tensor。将输入 Tensor 的 shape 设为 $(input_1, input_2, \dots, input_N)$ 。不支持复数类型。

输出：

tuple (Tensor)，表示 2 个 Tensor 组成的 tuple，包含对应的索引和输入 Tensor 的最小值。

- **index** (Tensor) - 输入 Tensor 最小值的索引，数据类型为 int64。如果 *keep_dims* 为 True，则输出 Tensor 的 shape 为 $(input_1, input_2, \dots, input_{axis-1}, 1, input_{axis+1}, \dots, input_N)$ 。否则，shape 为 $(input_1, input_2, \dots, input_{axis-1}, input_{axis+1}, \dots, input_N)$ 。
- **values** (Tensor) - 输入 Tensor 的最小值，其 shape 与 *index* 相同，数据类型与 *input* 相同。

异常：

- **TypeError** - *input* 不是 Tensor。
- **TypeError** - *keep_dims* 不是 bool。
- **TypeError** - *axis* 不是 int。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([0.0, 0.4, 0.6, 0.7, 0.1]), mindspore.float32)
>>> index, output = ops.ArgMinWithValue()(x)
>>> print(index, output)
0 0.0
>>> index, output = ops.ArgMinWithValue(keep_dims=True)(x)
>>> print(index, output)
[0] [0.0]
```

mindspore.ops.Median

class mindspore.ops.Median(global_median=False, axis=0, keep_dims=False, ignore_nan=False)

输出 Tensor 指定维度 *axis* 上的中值与其对应的索引。如果 *global_median* 为 True, 则计算 Tensor 中所有元素的中值。

警告:

- 如果 *input* 的中值不唯一时 (输入中有重复值且重复值恰好为中值), *indices* 不一定包含第一个出现的中值。*indices* 包含的索引与该算子的具体实现方式和后端类型相关, 因此某些情况下, CPU 和 GPU 的返回值可能不相同。
- 如果 *global_median* 为 True, 第二个输出 *indices* 无意义。

参数:

- **global_median** (bool, 可选) - 是否计算 Tensor 中所有元素的中值。默认值: False。
- **axis** (int, 可选) - 进行中值计算的轴。默认值: 0。
- **keep_dims** (bool, 可选) - 是否保留 *axis* 指定的维度。默认值: False。
- **ignore_nan** (bool, 可选) - 是否忽略输入 Tensor 中的 NaN 值。为 False 时, 若输入范围 (由 *global_median* 确定) 中含有 NaN 值, 则 *values* 的相应元素为 NaN。为 True 时, 计算排除 NaN 后剩余元素的中值。默认值: False。

输入:

- **x** (Tensor) - 要计算中值的 Tensor。

输出:

- **y** (Tensor) - 中值, 数据类型与 *x* 相同。
 - 如果 *global_median* 为 True, *y* 只有一个元素。
 - 如果 *keep_dims* 为 True, *y* 的 *shape* 除了在 *axis* 维度上为 1 外与 *x* 一致。
 - 其他情况下, *y* 比 *x* 缺少 *axis* 指定的维度。
- **indices** (Tensor) - 中值的索引。shape 与 *y* 一致, 数据类型为 int64。

异常:

- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *global_median*、*keep_dims* 或 *ignore_nan* 被指定了一个非 bool 值。

- **TypeError** - *axis* 不是 int。
- **ValueError** - *axis* 不在 $[-x.\text{dim}, x.\text{dim}-1]$ 范围内。

支持平台：

GPU CPU

样例：

```
>>> # case 1 : common median compute
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> x = Tensor(np.array([[5, 1, 2],[3, 5, 7], [1, 6, 4]]).astype(np.int64))
>>> median = ops.Median(global_median=False, axis=0, keep_dims=False)
>>> y = median(x)
>>> print(y)
(Tensor(shape=[3], dtype=Int64, value= [3, 5, 4]), Tensor(shape=[3], dtype=Int64, value= [1, 2]))
>>> # case 2 : global median compute
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> x = Tensor(np.array([[1, 7, 6],[5, 1, 3],[9, 17, 1]]).astype(np.int32))
>>> median = ops.Median(global_median=True)
>>> y = median(x)
>>> print(y)
(Tensor(shape=[], dtype=Int32, value= 5), Tensor(shape=[], dtype=Int64, value= 0))
```

mindspore.ops.ReduceAll

class mindspore.ops.ReduceAll(*keep_dims=False*)

```
prim = ops.ReduceAll(keep_dims)
out = prim(input, axis)
```

等价于

```
ops.all(input, axis, keep_dims)
```

更多详情请查看：[*mindspore.ops.all\(\)*](#)。

mindspore.ops.ReduceAny

class mindspore.ops.ReduceAny(*keep_dims=False*)

默认情况下，通过对指定维度所有元素进行逻辑或运算来移除该维度。也可仅缩小该维度大小至 1。*keep_dims* 控制输出和输入的维度是否相同。

说明：Tensor 类型的 *axis* 仅用作兼容旧版本，不推荐使用。

参数：

- **keep_dims** (bool, 可选) - 如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值：False。

输入：

- **x** (Tensor[bool]) - bool 类型的输入 Tensor。

- **axis** (Union[int, tuple(int), list(int), Tensor], 可选) - 要规约计算的维度。默认值: (), 在所有维度上规约。只允许常量值。取值必须在 [-rank(x), rank(x)) 范围内。

输出:

bool 类型的 Tensor。

- 如果 *axis* 为 (), 且 *keep_dims* 为 False, 则输出一个零维 Tensor, 表示输入 Tensor 中所有元素的逻辑或运算结果。
- 如果 *axis* 为 int, 值为 2, 并且 *keep_dims* 为 False, 则输出 Tensor 的 shape 为: $(x_1, x_3, \dots, x_R)$ 。
- 如果 *axis* 为 tuple(int), 值为 (2, 3), 并且 *keep_dims* 为 False, 则输出 Tensor 的 shape 为: $(x_1, x_4, \dots, x_R)$ 。
- 如果 *axis* 为一维 Tensor, 值为 [2, 3], 并且 *keep_dims* 为 False, 则输出 Tensor 的 shape 为: $(x_1, x_4, \dots, x_R)$ 。

异常:

- **TypeError** - *keep_dims* 不是 bool 类型。
- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *axis* 不是以下数据类型之一: int、Tuple、List 或 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[True, False], [True, True]]))
>>> op = ops.ReduceAny(keep_dims=True)
>>> # case 1: Reduces a dimension by the "logical OR" of all elements in the dimension.
>>> output = op(x)
>>> print(output)
[[ True]]
>>> print(output.shape)
(1, 1)
>>> # case 2: Reduces a dimension along axis 0.
>>> output = op(x, 0)
>>> print(output)
[[ True True]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = op(x, 1)
>>> print(output)
[[True]
 [ True]]
>>> # case 4: input is a scalar.
>>> x = Tensor(True)
>>> op = ops.ReduceAny()
>>> output = op(x)
>>> print(output)
True
```

mindspore.ops.ReduceMax

class mindspore.ops.ReduceMax(*keep_dims=False*)

默认情况下，使用指定维度的最大值代替该维度的其他元素，以移除该维度。也可仅缩小该维度大小至 1。*keep_dims* 控制输出和输入的维度是否相同。

通过指定 *keep_dims* 参数，来控制输出和输入的维度是否相同。

说明：Tensor 类型的 *axis* 仅用作兼容旧版本，不推荐使用。

参数：

- **keep_dims** (bool) - 如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值：False。

输入：

- **x** (Tensor[Number]) - 输入 Tensor。
- **axis** (Union[int, tuple(int), list(int), Tensor]) - 要进行规约计算的维度。默认值：()，在所有维度上进行规约。只允许常量值。假设 *x* 的秩为 *r*，取值范围 [-*r*,*r*)。

输出：

与输入 *x* 具有相同数据类型的 Tensor。

- 如果 *axis* 为 ()，且 *keep_dims* 为 False，则输出一个零维 Tensor，表示输入 Tensor 中所有元素的最大值。
- 如果 *axis* 为 int，取值为 1，并且 *keep_dims* 为 False，则输出的 shape 为 ($x_0, x_2, \dots, x_R$)。
- 如果 *axis* 为 tuple(int) 或 list(int)，取值为 (1, 2)，并且 *keep_dims* 为 False，则输出 Tensor 的 shape 为 ($x_0, x_3, \dots, x_R$)。
- 如果 *axis* 为一维 Tensor，取值为 [1, 2]，并且 *keep_dims* 为 False，则输出 Tensor 的 shape 为 ($x_0, x_3, \dots, x_R$)。

异常：

- **TypeError** - *keep_dims* 不是 bool 类型。
- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *axis* 不是以下数据类型之一：int、Tuple、List 或 Tensor。
- **ValueError** - *axis* 超出范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> output = ops.ReduceMax(keep_dims=True)(x, 1)
>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by the maximum value of all elements in the dimension.
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
mindspore.float32)
```

(下页继续)

(续上页)

```

>>> output = ops.ReduceMax(keep_dims=True)(x, ())
>>> print(output)
[[[9.]]]
>>> print(output.shape)
(1, 1, 1)
>>> # case 2: Reduces a dimension along axis 0.
>>> output = ops.ReduceMax(keep_dims=True)(x, 0)
>>> print(output)
[[[7. 7. 7. 7. 7. 7.]
  [8. 8. 8. 8. 8. 8.]
  [9. 9. 9. 9. 9. 9.]]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = ops.ReduceMax(keep_dims=True)(x, 1)
>>> print(output)
[[[3. 3. 3. 3. 3. 3.]
  [6. 6. 6. 6. 6. 6.]
  [9. 9. 9. 9. 9. 9.]]]
>>> # case 4: Reduces a dimension along axis 2.
>>> output = ops.ReduceMax(keep_dims=True)(x, 2)
>>> print(output)
[[[1.]
  [2.]
  [3.]
  [4.]
  [5.]
  [6.]
  [7.]
  [8.]
  [9.]]]

```

mindspore.ops.ReduceMean

class mindspore.ops.ReduceMean (*keep_dims=False*)

默认情况下，使用指定维度的平均值代替该维度的其他元素，以移除该维度。也可仅缩小该维度大小至 1。

通过指定 *keep_dims* 参数，来控制输出和输入的维度是否相同。

说明：Tensor 类型的 *axis* 仅用作兼容旧版本，不推荐使用。

参数：

- **keep_dims** (bool) - 如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值：False。

输入：

- **x** (Tensor[Number]) - 输入 Tensor。
- **axis** (Union[int, tuple(int), list(int), Tensor]) - 要进行规约计算的维度。默认值：()，在所有维度上进行规约。只允许常量值。假设 *x* 的秩为 *r*，取值范围 [-*r*,*r*)。

输出：

与输入 *x* 具有相同数据类型的 Tensor。

- 如果 *axis* 为 ()，且 *keep_dims* 为 False，则输出一个零维 Tensor，表示输入 Tensor 中所有元素的平均值。
- 如果 *axis* 为 int，取值为 1，并且 *keep_dims* 为 False，则输出的 shape 为 (*x*₀, *x*₂, ..., *x*_{*R*})。

- 如果 *axis* 为 `tuple(int)` 或 `list(int)`, 取值为 (1, 2), 并且 *keep_dims* 为 `False`, 则输出 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。
- 如果 *axis* 为一维 Tensor, 取值为 [1, 2], 并且 *keep_dims* 为 `False`, 则输出 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。

异常:

- **TypeError** - *keep_dims* 不是 bool 类型。
- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *axis* 不是以下数据类型之一: `int`、`Tuple`、`List` 或 `Tensor`。
- **ValueError** - *axis* 超出范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> op = ops.ReduceMean(keep_dims=True)
>>> output = op(x, 1)
>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by averaging all elements in the dimension.
>>> x = Tensor(np.array([[[2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2], [2, 2, 2, 2, 2, 2]],
... [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
... [[6, 6, 6, 6, 6, 6], [8, 8, 8, 8, 8, 8], [10, 10, 10, 10, 10, 10]]]),
... mindspore.float32)
>>> output = op(x)
>>> print(output)
[[[5.]]]
>>> print(output.shape)
(1, 1, 1)
>>> # case 2: Reduces a dimension along the axis 0
>>> output = op(x, 0)
>>> print(output)
[[[4. 4. 4. 4. 4. 4.]
[5. 5. 5. 5. 5. 5.]
[6. 6. 6. 6. 6. 6.]]]
>>> # case 3: Reduces a dimension along the axis 1
>>> output = op(x, 1)
>>> print(output)
[[[2. 2. 2. 2. 2. 2.]
[5. 5. 5. 5. 5. 5.]
[8. 8. 8. 8. 8. 8.]]]
>>> # case 4: Reduces a dimension along the axis 2
>>> output = op(x, 2)
>>> print(output)
[[[ 2.]
[ 2.]
[ 2.]]
[[ 4.]
[ 5.]
[ 6.]]]
```

(下页继续)

(续上页)

```
[[ 6.]
 [ 8.]
 [10.]]]
```

mindspore.ops.ReduceMin

class mindspore.ops.ReduceMin(*keep_dims=False*)

默认情况下，使用指定维度的最小值代替该维度的其他元素，以移除该维度。也可仅缩小该维度大小至 1。

通过指定 *keep_dims* 参数，来控制输出和输入的维度是否相同。

说明：Tensor 类型的 *axis* 仅用作兼容旧版本，不推荐使用。

参数：

- **keep_dims** (bool) - 如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值：False。

输入：

- **x** (Tensor[Number]) - 输入 Tensor。
- **axis** (Union[int, tuple(int), list(int), Tensor]) - 要进行规约计算的维度。默认值：()，在所有维度上进行规约。只允许常量值。假设 *x* 的秩为 *r*，取值范围 [-*r*,*r*)。

输出：

与输入 *x* 具有相同数据类型的 Tensor。

- 如果 *axis* 为 ()，且 *keep_dims* 为 False，则输出一个零维 Tensor，表示输入 Tensor 中所有元素的最小值。
- 如果 *axis* 为 int，取值为 1，并且 *keep_dims* 为 False，则输出的 shape 为 $(x_0, x_2, \dots, x_R)$ 。
- 如果 *axis* 为 tuple(int) 或 list(int)，取值为 (1, 2)，并且 *keep_dims* 为 False，则输出 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。
- 如果 *axis* 为一维 Tensor，取值为 [1, 2]，并且 *keep_dims* 为 False，则输出 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。

异常：

- **TypeError** - *keep_dims* 不是 bool 类型。
- **TypeError** - *x* 不是 Tensor。
- **TypeError** - *axis* 不是以下数据类型之一：int、Tuple、List 或 Tensor。
- **ValueError** - *axis* 超出范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> op = ops.ReduceMin(keep_dims=True)
>>> output = op(x, 1)
```

(下页继续)

(续上页)

```

>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by the minimum value of all elements in the dimension.
>>> x = Tensor(np.array([[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]),
mindspore.float32)
>>> output = op(x)
>>> print(output)
[[[1.]]]
>>> print(output.shape)
(1, 1, 1)
>>> # case 2: Reduces a dimension along axis 0.
>>> output = op(x, 0)
>>> print(output)
[[[1. 1. 1. 1. 1. 1.]
  [2. 2. 2. 2. 2. 2.]
  [3. 3. 3. 3. 3. 3.]]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = op(x, 1)
>>> print(output)
[[[1. 1. 1. 1. 1. 1.]
  [4. 4. 4. 4. 4. 4.]
  [7. 7. 7. 7. 7. 7.]]]
>>> # case 4: Reduces a dimension along axis 2.
>>> output = op(x, 2)
>>> print(output)
[[[1.]
  [2.]
  [3.]]
  [[4.]
  [5.]
  [6.]]
  [[7.]
  [8.]
  [9.]]]

```

mindspore.ops.ReduceProd

class mindspore.ops.ReduceProd(keep_dims=False)

默认情况下，使用指定维度所有元素的乘积代替该维度的其他元素，以移除该维度。也可仅缩小该维度大小至 1。通过指定 `keep_dims` 参数，来控制输出和输入的维度是否相同。

说明：Tensor 类型的 `axis` 仅用作兼容旧版本，不推荐使用。

参数：

- **keep_dims** (bool) - 如果为 True，则保留缩小的维度，大小为 1。否则移除维度。默认值：False。

输入：

- **x** (Tensor[Number]) - 输入 Tensor。

- **axis** (Union[int, tuple(int), list(int), Tensor]) - 要进行规约计算的维度。默认值: (), 在所有维度上进行规约。只允许常量值。假设 x 的秩为 r , 取值范围 $[-r, r)$ 。

输出:

与输入 x 具有相同数据类型的 Tensor。

- 如果 **axis** 为 (), 且 **keep_dims** 为 False, 则输出一个零维 Tensor, 表示输入 Tensor 中所有元素的乘积。
- 如果 **axis** 为 int, 取值为 1, 并且 **keep_dims** 为 False, 则输出的 shape 为 $(x_0, x_2, \dots, x_R)$ 。
- 如果 **axis** 为 tuple(int) 或 list(int), 取值为 (1, 2), 并且 **keep_dims** 为 False, 则输出 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。
- 如果 **axis** 为一维 Tensor, 取值为 [1, 2], 并且 **keep_dims** 为 False, 则输出 Tensor 的 shape 为 $(x_0, x_3, \dots, x_R)$ 。

异常:

- **TypeError** - **keep_dims** 不是 bool 类型。
- **TypeError** - x 不是 Tensor。
- **TypeError** - **axis** 不是以下数据类型之一: int、Tuple、List 或 Tensor。
- **ValueError** - **axis** 超出范围。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> op = ops.ReduceProd(keep_dims=True)
>>> output = op(x, 1)
>>> result = output.shape
>>> print(result)
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by multiplying all elements in the dimension.
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
↳mindspore.float32)
>>> output = op(x)
>>> print(output)
[[[2.2833798e+33]]]
>>> print(output.shape)
(1, 1, 1)
>>> # case 2: Reduces a dimension along axis 0.
>>> output = op(x, 0)
>>> print(output)
[[[ 28.  28.  28.  28.  28.]
 [ 80.  80.  80.  80.  80.]
 [162. 162. 162. 162. 162.]]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = op(x, 1)
>>> print(output)
[[[ 6.  6.  6.  6.  6.]
 [120. 120. 120. 120. 120.]
```

(下页继续)

(续上页)

```
[[504. 504. 504. 504. 504. 504.]]]
>>> # case 4: Reduces a dimension along axis 2.
>>> output = op(x, 2)
>>> print(output)
[[[1.00000e+00]
[6.40000e+01]
[7.29000e+02]]
[[4.09600e+03]
[1.56250e+04]
[4.66560e+04]]
[[1.17649e+05]
[2.62144e+05]
[5.31441e+05]]]
```

mindspore.ops.ReduceSum

class mindspore.ops.ReduceSum(keep_dims=False, skip_mode=False)

默认情况下，输出 Tensor 各维度上的和，以达到对所有维度进行归约的目的。也可以对指定维度进行求和归约。

通过指定 keep_dims 参数，来控制输出和输入的维度是否相同。

说明：Tensor 类型的 axis 仅用作兼容旧版本，不推荐使用。

参数：

- **keep_dims** (bool, 可选) - 如果为 True，则保留计算维度，长度为 1。如果为 False，则不保留计算维度。默认值：False，输出结果会降低维度。
- **skip_mode** (bool, 可选) - 如果为 True，当 axis 为空 tuple 或空 list 时，不进行 ReduceSum 计算，当 axis 为其他值时，正常运算。如果为 False，则正常进行运算。默认值：False。

输入：

- **x** (Tensor[Number]) - 输入 Tensor。
- **axis** (Union[int, tuple(int), list(int), Tensor]) - 要减少的维度。默认值：()，当 skip_mode 为 False 时，缩小所有维度。只允许常量值，取值范围 [-rank(x), rank(x))。

输出：

Tensor，具有与输入 x 相同的 shape。

- 如果 axis 为 ()，且 keep_dims 为 False，skip_mode 为 False，则输出一个零维 Tensor，表示输入 Tensor 中所有元素的和。
- 如果 axis 为 ()，且 skip_mode 为 True，则不进行 ReduceSum 运算，输出 Tensor 等于输入 Tensor。
- 如果 axis 为 int，取值为 2，并且 keep_dims 为 False，则输出的 shape 为 $(x_1, x_3, \dots, x_R)$ 。
- 如果 axis 为 tuple(int) 或 list(int)，取值为 (2, 3)，并且 keep_dims 为 False，则输出的 shape 为 $(x_1, x_4, \dots, x_R)$ 。
- 如果 axis 为一维 Tensor，取值为 [2, 3]，并且 keep_dims 为 False，则输出 Tensor 的 shape 为 $(x_1, x_4, \dots, x_R)$ 。

异常：

- **TypeError** - keep_dims 不是 bool。
- **TypeError** - skip_mode 不是 bool。

- **TypeError** - x 不是 Tensor。
- **ValueError** - $axis$ 取值为 None。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.randn(3, 4, 5, 6).astype(np.float32))
>>> op = ops.ReduceSum(keep_dims=True)
>>> output = op(x, 1)
>>> output.shape
(3, 1, 5, 6)
>>> # case 1: Reduces a dimension by summing all elements in the dimension.
>>> x = Tensor(np.array([[[[1, 1, 1, 1, 1, 1], [2, 2, 2, 2, 2, 2], [3, 3, 3, 3, 3, 3]],
...                      [[4, 4, 4, 4, 4, 4], [5, 5, 5, 5, 5, 5], [6, 6, 6, 6, 6, 6]],
...                      [[7, 7, 7, 7, 7, 7], [8, 8, 8, 8, 8, 8], [9, 9, 9, 9, 9, 9]]]),
mindspore.float32)
>>> output = op(x)
>>> print(output)
[[[270.]]]
>>> print(output.shape)
(1, 1, 1)
>>> # case 2: Reduces a dimension along axis 0.
>>> output = op(x, 0)
>>> print(output)
[[[12. 12. 12. 12. 12. 12.]
[15. 15. 15. 15. 15. 15.]
[18. 18. 18. 18. 18. 18.]]]
>>> # case 3: Reduces a dimension along axis 1.
>>> output = op(x, 1)
>>> print(output)
[[[ 6.  6.  6.  6.  6.  6.]
[15. 15. 15. 15. 15. 15.]
[24. 24. 24. 24. 24. 24.]]]
>>> # case 4: Reduces a dimension along axis 2.
>>> output = op(x, 2)
>>> print(output)
[[[ 6.]
[12.]
[18.]]
[[24.]
[30.]
[36.]
[42.]
[48.]
[54.]]]
```

20.3.10 比较算子

接口名	概述	支持平台
<code>mindspore.ops.ApproximateEqual</code>	逐元素计算 $\text{abs}(x-y)$ ，如果小于 <code>tolerance</code> 则为 <code>True</code> ，否则为 <code>False</code> 。	Ascend GPU CPU
<code>mindspore.ops.Equal</code>	更多详情请查看: <code>mindspore.ops.equal()</code> 。	Ascend GPU CPU
<code>mindspore.ops.EqualCount</code>	计算两个 <code>Tensor</code> 的相同元素的数量。	Ascend GPU CPU
<code>mindspore.ops.Greater</code>	更多详情请查看: <code>mindspore.ops.greater()</code> 。	Ascend GPU CPU
<code>mindspore.ops.GreaterEqual</code>	更多详情请查看: <code>mindspore.ops.greater_equal()</code> 。	Ascend GPU CPU
<code>mindspore.ops.InTopK</code>	判断目标标签是否在前 k 个预测中。	Ascend GPU CPU
<code>mindspore.ops.IsFinite</code>	更多详情请查看: <code>mindspore.ops.isfinite()</code> 。	Ascend GPU CPU
<code>mindspore.ops.IsInf</code>	更多详情请查看: <code>mindspore.ops.isinf()</code> 。	Ascend CPU GPU
<code>mindspore.ops.IsNan</code>	判断输入数据每个位置上的值是否是 <code>NaN</code> 。	Ascend GPU CPU
<code>mindspore.ops.Less</code>	更多详情请查看: <code>mindspore.ops.less()</code> 。	Ascend GPU CPU
<code>mindspore.ops.LessEqual</code>	更多详情请查看: <code>mindspore.ops.less_equal()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Maximum</code>	更多详情请查看: <code>mindspore.ops.maximum()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Minimum</code>	更多详情请查看: <code>mindspore.ops.minimum()</code> 。	Ascend GPU CPU
<code>mindspore.ops.NotEqual</code>	更多详情请查看: <code>mindspore.ops.not_equal()</code> 。	Ascend GPU CPU
<code>mindspore.ops.TopK</code>	沿最后一个维度查找 k 个最大元素和对应的索引。	Ascend GPU CPU

mindspore.ops.ApproximateEqual

`class mindspore.ops.ApproximateEqual (tolerance=1e-05)`
逐元素计算 $\text{abs}(x-y)$ ，如果小于 `tolerance` 则为 `True`，否则为 `False`。

$$out_i = \begin{cases} \text{if } |x_i - y_i| < tolerance, & True \\ \text{if } |x_i - y_i| \geq tolerance, & False \end{cases}$$

`tolerance` 为相等的两元素间最大偏差。输入 x 和 y 会通过隐式数据类型转换使数据类型保持一致。如果数据类型不同，低精度的数据类型会被自动转换到高精度的数据类型。

参数：

- **tolerance** (float, 可选) - 两元素可被视为相等的最大偏差。默认值：1e-05。

输入：

- **x** (Tensor) - 输入 Tensor，需为以下数据类型：float16、float32。shape: $(N,*)$ ，其中 $*$ 表示任何数量的附加维度。其秩应小于 8。
- **y** (Tensor) - 输入 Tensor，shape 和数据类型与 x 相同。

输出：

Tensor, shape 与 x 相同, bool 类型。

异常:

- **TypeError** - *tolerance* 不是 float 类型。
- **TypeError** - x 与 y 之间的类型转换不被支持。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1, 2, 3]), mindspore.float32)
>>> y = Tensor(np.array([2, 3, 6]), mindspore.float32)
>>> approximate_equal = ops.ApproximateEqual(2.)
>>> output = approximate_equal(x, y)
>>> print(output)
[ True  True  False]
```

mindspore.ops.Equal

class mindspore.ops.Equal

```
prim = ops.Equal()
out = prim(input, other)
```

等价于

```
ops.equal(input, other)
```

更多详情请查看: [mindspore.ops.equal\(\)](#)。

mindspore.ops.EqualCount

class mindspore.ops.EqualCount

计算两个 Tensor 的相同元素的数量。

两个输入 Tensor 必须具有相同的数据类型和 shape。

输入:

- **x** (Tensor) - 第一个输入 Tensor。如果确定了 y 的数据类型和 shape, 则 x 必须与 y 相同, 反之亦然。 $(N, *)$, 其中 $*$ 表示任意数量的附加维度。
- **y** (Tensor) - 第二个输入 Tensor。如果确定了 x 的数据类型和 shape, 则 y 必须与 x 相同, 反之亦然。

输出:

Tensor, 数据类型与输入 Tensor 相同, shape 为 $(1,)$ 。

异常:

- **TypeError** - 如果 x 或 y 不是 Tensor。
- **ValueError** - 如果 x 与 y 的 shape 不相等。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1, 2, 3]), mindspore.int32)
>>> y = Tensor(np.array([1, 2, 4]), mindspore.int32)
>>> equal_count = ops.EqualCount()
>>> output = equal_count(x, y)
>>> print(output)
[2]
```

mindspore.ops.Greater

class mindspore.ops.Greater

```
prim = ops.Greater()
out = prim(input, other)
```

等价于

```
ops.greater(input, other)
```

更多详情请查看: [mindspore.ops.greater\(\)](#)。

mindspore.ops.GreaterEqual

class mindspore.ops.GreaterEqual

```
prim = ops.GreaterEqual()
out = prim(input, other)
```

等价于

```
ops.greater_equal(input, other)
```

更多详情请查看: [mindspore.ops.greater_equal\(\)](#)。

mindspore.ops.InTopK

class mindspore.ops.InTopK(*k*)

判断目标标签是否在前 *k* 个预测中。

更多参考详见[mindspore.ops.intopk\(\)](#)。

参数:

- **k** (int) - 指定在最后一维上参与比较的 top 元素的数量。

输入:

- **x1** (Tensor) - 二维 Tensor, 表示对样本的预测。数据类型支持 float16 或 float32。

- **x2 (Tensor)** - 一维 Tensor，表示样本的标签。数据类型为 int32。x2 的大小必须与 x1 第一维度的大小相同。x2 取值不可为负，且必须小于或等于 x1 第二维度的大小。

输出：

一维的 bool 类型 Tensor，与 x2 的 shape 相同。对于 x2 中的样本标签 i ，如果它在 x1 的前 k 个预测值中，则输出值为 True，否则为 False。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x1 = Tensor(np.array([[1, 8, 5, 2, 7], [4, 9, 1, 3, 5]]), mindspore.float32)
>>> x2 = Tensor(np.array([1, 3]), mindspore.int32)
>>> in_top_k = ops.InTopK(3)
>>> output = in_top_k(x1, x2)
>>> print(output)
[ True False]
```

mindspore.ops.IsFinite

class mindspore.ops.IsFinite

```
prim = ops.IsFinite()
out = prim(input)
```

等价于

```
ops.isfinite(input)
```

更多详情请查看：[mindspore.ops.isfinite\(\)](#)。

mindspore.ops.IsInf

class mindspore.ops.IsInf

```
prim = ops.IsInf()
out = prim(input)
```

等价于

```
ops.isinf(input)
```

更多详情请查看：[mindspore.ops.isinf\(\)](#)。

mindspore.ops.IsNan

class mindspore.ops.IsNan

判断输入数据每个位置上的值是否是 NaN。

更多参考详见[mindspore.ops.isnan\(\)](#)。

输入：

- **x** (Tensor) - 输入 Tensor。

输出：

Tensor, shape 与相同的输入, 数据的类型为 bool。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> is_nan = ops.IsNan()
>>> x = Tensor(np.array([np.log(-1), 1, np.log(0)]), mindspore.float32)
>>> output = is_nan(x)
>>> print(output)
[ True False False]
>>> x = Tensor(2.1, mindspore.float64)
>>> output = is_nan(x)
>>> print(output)
False
```

mindspore.ops.Less

class mindspore.ops.Less

```
prim = ops.Less()
out = prim(input, other)
```

等价于

```
ops.less(input, other)
```

更多详情请查看：[mindspore.ops.less\(\)](#)。

mindspore.ops.LessEqual

class mindspore.ops.LessEqual

```
prim = ops.LessEqual()
out = prim(input, other)
```

等价于

```
ops.less_equal(input, other)
```

更多详情请查看: `mindspore.ops.less_equal()`。

mindspore.ops.Maximum

class mindspore.ops.Maximum

```
prim = ops.Maximum()
out = prim(input, other)
```

等价于

```
ops.maximum(input, other)
```

更多详情请查看: `mindspore.ops.maximum()`。

mindspore.ops.Minimum

class mindspore.ops.Minimum

```
prim = ops.Minimum()
out = prim(input, other)
```

等价于

```
ops.minimum(input, other)
```

更多详情请查看: `mindspore.ops.minimum()`。

mindspore.ops.NotEqual

class mindspore.ops.NotEqual

```
prim = ops.NotEqual()
out = prim(input, other)
```

等价于

```
ops.not_equal(input, other)
```

更多详情请查看: `mindspore.ops.not_equal()`。

mindspore.ops.TopK

class mindspore.ops.TopK(*sorted=True*)

沿最后一个维度查找 k 个最大元素和对应的索引。

警告:

- 如果 *sorted* 设置为 `False`，它将使用 `aicpu` 运算符，性能可能会降低。另外，由于不同平台上的内存排布、遍历方式等不同，*sorted* 设置为 `False` 时计算结果的显示顺序可能会不一致。

如果 *input_x* 是一维 Tensor，则查找 Tensor 中 *k* 个最大元素，并将其值和索引输出为 Tensor。*values[k]* 是 *input_x* 中 *k* 个最大元素，其索引是 *indices[k]*。

对于多维矩阵，计算每行中最大的 *k* 个元素（沿最后一个维度的相应向量），因此：

$$values.shape = indices.shape = input.shape[:-1] + [k]$$

如果两个比较的元素相同，则优先返回索引值较小的元素。

参数：

- **sorted** (bool, 可选) - 如果为 True，则获取的元素将按值降序排序。如果为 False，则不对获取的元素进行排序。默认值：True。

输入：

- **input_x** (Tensor) - 需计算的输入，目前 GPU 支持零维输入，但是 Ascend 或者 CPU 不支持。支持的数据类型：
 - Ascend: int8、uint8、int32、int64、float16、float32。
 - GPU: float16、float32。
 - CPU: 所有数值型。
- **k** (Union(Tensor, int)) - 指定计算最大元素的数量。若 *k* 为 Tensor，其数据类型须为 int32，且只支持零维 Tensor 或 shape 为 (1,) 的一维 Tensor。

输出：

由 *values* 和 *indices* 组成的 tuple。

- **values** (Tensor) - 最后一个维度的每个切片中的 *k* 个最大元素。
- **indices** (Tensor) - *k* 个最大元素的对应索引。

异常：

- **TypeError** - 如果 *sorted* 不是 bool。
- **TypeError** - 如果 *input_x* 不是 Tensor。
- **TypeError** - 如果 *k* 不是 int。
- **TypeError** - 如果 *input_x* 的数据类型不被支持。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> import mindspore
>>> input_x = Tensor([1, 2, 3, 4, 5], mindspore.float16)
>>> k = 3
>>> values, indices = ops.TopK(sorted=True)(input_x, k)
>>> print((values, indices))
(Tensor(shape=[3], dtype=Float16, value= [ 5.0000e+00,  4.0000e+00,  3.0000e+00]),
↳Tensor(shape=[3],
dtype=Int32, value= [4, 3, 2]))
```


20.3.11 线性代数算子

接口名	概述	支持平台
<code>mindspore.ops.BatchMatMul</code>	基于 <code>batch</code> 维度的两个 <code>Tensor</code> 的矩阵乘法。	Ascend GPU CPU
<code>mindspore.ops.BiasAdd</code>	返回输入 <code>Tensor</code> 与偏置 <code>Tensor</code> 之和。	Ascend GPU CPU
<code>mindspore.ops.Ger</code>	计算两个一维 <code>Tensor</code> 的外积。	Ascend GPU CPU
<code>mindspore.ops.MatMul</code>	将矩阵 <code>a</code> 和矩阵 <code>b</code> 相乘。	Ascend GPU CPU
<code>mindspore.ops.MatrixInverse</code>	计算输入矩阵的逆矩阵。	Ascend GPU CPU
<code>mindspore.ops.Ormqr</code>	计算一个普通矩阵与 <code>Householder</code> 矩阵的乘积。	GPU
<code>mindspore.ops.Orgqr</code>	计算 <code>mindspore.ops.Geqrf</code> 返回的正交矩阵 <code>Q</code> 的显式表示。	Ascend GPU CPU
<code>mindspore.ops.Svd</code>	计算一个或多个矩阵的奇异值分解。	Ascend GPU CPU

mindspore.ops.BatchMatMul

class mindspore.ops.BatchMatMul (*transpose_a=False, transpose_b=False*)

基于 `batch` 维度的两个 `Tensor` 的矩阵乘法。

$$\text{output}[\dots, :, :] = \text{matrix}(x[\dots, :, :]) * \text{matrix}(y[\dots, :, :])$$

两个输入 `Tensor` 的秩必须不小于 2，如果是 GPU 或者 CPU 环境，两个输入 `Tensor` 必须具有相同的秩。

参数：

- **transpose_a** (bool, 可选) - 如果为 `True`，则在乘法之前转置 `x` 的最后两个维度。默认值：`False`。
- **transpose_b** (bool, 可选) - 如果为 `True`，则在乘法之前转置 `y` 的最后两个维度。默认值：`False`。

输入：

- **x** (`Tensor`) - 输入相乘的第一个 `Tensor`，其 `shape` 为 `(*B, N, C)`，其中 `*B` 表示批处理大小，可以是多维度，`N` 和 `C` 是最后两个维度的大小。如果 `transpose_a` 为 `True`，则其 `shape` 必须为 `(*B, C, N)`。
- **y** (`Tensor`) - 输入相乘的第二个 `Tensor`，其 `shape` 为 `(*B, C, M)`。如果 `transpose_b` 为 `True`，则其 `shape` 必须为 `(*B, M, C)`。

输出：

`Tensor`，输出 `Tensor` 的 `shape` 为 `(*B, N, M)`。

异常：

- **TypeError** - `transpose_a` 或 `transpose_b` 不是 `bool`。
- **ValueError** - `x` 的 `shape` 长度不等于 `y` 的 `shape` 长度或输入的 `shape` 长度小于 2。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.ones(shape=[2, 4, 1, 3]), mindspore.float32)
>>> y = Tensor(np.ones(shape=[2, 4, 3, 4]), mindspore.float32)
>>> batmatmul = ops.BatchMatMul()
>>> output = batmatmul(x, y)
```

(下页继续)

(续上页)

```
>>> print(output.shape)
(2, 4, 1, 4)
>>> x = Tensor(np.ones(shape=[2, 4, 3, 1]), mindspore.float32)
>>> y = Tensor(np.ones(shape=[2, 4, 3, 4]), mindspore.float32)
>>> batmatmul = ops.BatchMatMul(transpose_a=True)
>>> output = batmatmul(x, y)
>>> print(output.shape)
(2, 4, 1, 4)
```

mindspore.ops.BiasAdd

class mindspore.ops.BiasAdd(data_format='NCHW')

返回输入 Tensor 与偏置 Tensor 之和。相加前会把偏置 Tensor 广播成与输入 Tensor 的 shape 一致。

参数：

- **data_format** (str, 可选) - 输入和输出数据的格式。取值为 "NHWC"、"NCHW" 或 "NCDHW"。默认值: "NCHW"。

输入：

- **input_x** (Tensor) - 输入 Tensor。shape 可以有 2~5 个维度。支持数据类型：
 - Ascend/CPU：所有数值类型。
 - GPU：float16、float32、int8。
- **bias** (Tensor) - 偏置 Tensor，shape 为 (C)。C 必须与 *input_x* 的通道维度 C 相同。其数据类型与 *input_x* 一致。

输出：

Tensor，shape 和数据类型与 *input_x* 相同。

异常：

- **TypeError** - *data_format* 不是 str。
- **ValueError** - *data_format* 的值不在 ['NHWC', 'NCHW', 'NCDHW'] 范围内。
- **TypeError** - *input_x* 或 *bias* 不是 Tensor。
- **TypeError** - *input_x* 与 *bias* 的数据类型不一致。
- **TypeError** - *input_x* 的维度不在 [2, 5] 范围内。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.arange(6).reshape((2, 3)), mindspore.float32)
>>> bias = Tensor(np.random.random(3).reshape((3,)), mindspore.float32)
>>> bias_add = ops.BiasAdd()
>>> output = bias_add(input_x, bias)
>>> print(output.shape)
(2, 3)
```

mindspore.ops.Ger

class mindspore.ops.Ger

计算两个一维 Tensor 的外积。即输入 $x1$ 和 $x2$ 的外积。如果 $x1$ shape 为 $(m,)$ ， $x2$ shape 为 $(n,)$ ，那么输出就是一个 shape 为 (m,n) 的 Tensor。

更多参考详见 `mindspore.ops.ger()`。

输入：

- **x1** (Tensor) - 输入 1-D Tensor。
- **x2** (Tensor) - 输入 1-D Tensor，输入数据类型需和 $x1$ 保持一致。

输出：

Tensor，与 $x1$ 数据类型相同的输出 Tensor。如果 $x1$ shape 为 $(m,)$ ， $x2$ shape 为 $(n,)$ ，则输出的 shape 为 (m,n) 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> x1 = Tensor([1., 2., 3., 4.], mindspore.float32)
>>> x2 = Tensor([1., 2., 3.], mindspore.float32)
>>> ger = ops.Ger()
>>> output = ger(x1, x2)
>>> print(output)
[[ 1.  2.  3.]
 [ 2.  4.  6.]
 [ 3.  6.  9.]
 [ 4.  8. 12.]]
```

mindspore.ops.MatMul

class mindspore.ops.MatMul (transpose_a=False, transpose_b=False)

将矩阵 a 和矩阵 b 相乘。

$$(Output)_{ij} = \sum_{k=1}^P a_{ik} b_{kj} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{ip}b_{pj}, p \in N$$

其中， i, j 表示输出的第 i 行和第 j 列元素。

说明：

- 当 $N * M$ 不能被 16 整除时，算子在 Ascend 环境上的性能会比较差。
- 输入 Tensor 的数据类型必须一致。
- Ascend 上，暂不支持 float64 类型。

参数：

- **transpose_a** (bool, 可选) - 如果为 True，则在相乘之前转置 a 。默认值：False。
- **transpose_b** (bool, 可选) - 如果为 True，则在相乘之前转置 b 。默认值：False。

输入：

- **a** (Tensor) - 要相乘的第一个 Tensor。如果 *transpose_a* 为 `False`，则该 Tensor 的 shape 为 (N, C) ；否则，该 Tensor 的 shape 为 (C, N) 。
- **b** (Tensor) - 要相乘的第二个 Tensor。如果 *transpose_b* 为 `False`，则该 Tensor 的 shape 为 (C, M) ；否则，该 Tensor 的 shape 为 (M, C) 。

输出：

Tensor，输出 Tensor 的 shape 为 (N, M) 。

异常：

- **TypeError** - *transpose_a* 或 *transpose_b* 不是 bool。
- **TypeError** - *a* 的 dtype 和 *b* 的 dtype 不一致。
- **ValueError** - 矩阵 *a* 的列不等于矩阵 *b* 的行。
- **ValueError** - *a* 或 *b* 的维度不等于 2。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> a = Tensor(np.ones(shape=[1, 3]), mindspore.float32)
>>> b = Tensor(np.ones(shape=[3, 4]), mindspore.float32)
>>> matmul = ops.MatMul()
>>> output = matmul(a, b)
>>> print(output)
[[3. 3. 3. 3.]]
```

mindspore.ops.MatrixInverse

class mindspore.ops.MatrixInverse (*adjoint=False*)

计算输入矩阵的逆矩阵。如果输入矩阵不可逆，将产生错误或者返回一个未知结果。

说明：参数 *adjoint* 目前只支持 `False`，因为目前该算子不支持复数。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **adjoint** (bool) - 指定是否支持复数，`False` 表示为不支持复数。默认值：`False`。

输入：

- **x** (Tensor) - 输入需计算的矩阵，至少为二维矩阵，且最后两个维度大小相同，数据类型为 `float32`、`float64`。

输出：

Tensor，数据类型和 shape 与输入 *x* 相同。

异常:

- **TypeError** - *adjoint* 不是 bool。
- **TypeError** - *x* 的数据类型既不是 float32, 也不是 float64。
- **ValueError** - *x* 最后两个维度大小不同。
- **ValueError** - *x* 低于二维。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[-0.710504, -1.1207525],
...                      [-1.7651395, -1.7576632]],
...                      [[ 0.52412605,  1.9070215],
...                      [ 1.3384849,  1.4274558]]]), mindspore.float32)
>>> matrix_inverse = ops.MatrixInverse(adjoint=False)
>>> output = matrix_inverse(x)
>>> print(output)
[[[ 2.4095478 -1.5364188 ]
  [-2.419797  0.9740167 ]]
 [[-0.79111797  1.0569006 ]
  [ 0.74180895 -0.2904787 ]]]
```

mindspore.ops.Ormqr

class mindspore.ops.Ormqr (*left=True, transpose=False*)

计算一个普通矩阵与 Householder 矩阵的乘积。计算维度为 (m, n) 的矩阵 C (由 *other* 给出) 和一个矩阵 Q 的乘积, 其中 Q 由 Householder 反射系数 (*x, tau*) 表示, 通常由 `mindspore.ops.geqrf()` 计算生成。

更多细节请参考 `mindspore.ops.ormqr()`。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **left** (bool, 可选) - 决定了矩阵乘积运算的顺序。如果 *left* 为 True, 计算顺序为 `op(Q) * other`, 否则, 计算顺序为 `other * op(Q)`。默认值: True。
- **transpose** (bool, 可选) - 如果为 True, 对矩阵 Q 进行共轭转置变换, 否则, 不对矩阵 Q 进行共轭转置变换。默认值: False。

输入:

- **x** (Tensor) - shape `(*, mn, k)`, *mn* 的值取决于 *left*, 当 *left* 为 True 时, *mn* 的值等于 *m*, 否则 *mn* 的值等于 *n*。
* 表示 Tensor 在轴 0 上的长度为 0 或者大于 0 的 batch 维。
- **tau** (Tensor) - shape `(*, min(mn, k))`, 其中 * 表示 Tensor 在轴 0 上的长度为 0 或者大于 0 的 batch 维, 其类型与 *x* 相同。
- **other** (Tensor) - shape `(*, m, n)`, 其中 * 表示 Tensor 在轴 0 上的长度为 0 或者大于 0 的 batch 维, 其类型与 *x* 相同。

输出:

- **y** (Tensor) - 输出 Tensor, 数据类型与 `shape` 与 `other` 一致。

异常:

- **TypeError** - 如果 `x`, `tau` 或者 `other` 不是 Tensor。
- **TypeError** - 如果 `x`, `tau` 和 `other` 的 `dtype` 不是 `float64`、`float32`、`complex64` 或者 `complex128`。
- **ValueError** - 如果 `x` 或 `other` 的维度小于 2D。
- **ValueError** - `rank(x) - rank(tau) != 1`。
- **ValueError** - `tau.shape[-1] != x.shape[-2]`。
- **ValueError** - `other.shape[-2] != x.shape[-2]`。
- **ValueError** - 当 `left` 为 `True` 时, `other.shape[-2] < tau.shape[-1]`。
- **ValueError** - 当 `left` 为 `True` 时, `other.shape[-2] != x.shape[-2]`。
- **ValueError** - 当 `left` 为 `False` 时, `other.shape[-1] < tau.shape[-1]`。
- **ValueError** - 当 `left` 为 `False` 时, `other.shape[-1] != x.shape[-2]`。

支持平台:

GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[[-114.6, 10.9, 1.1], [-0.304, 38.07, 69.38], [-0.45, -0.17, 62]]],
↳mindspore.float32)
>>> tau = Tensor(np.array([1.55, 1.94, 3.0]), mindspore.float32)
>>> other = Tensor(np.array([[[-114.6, 10.9, 1.1],
...                           [-0.304, 38.07, 69.38],
...                           [-0.45, -0.17, 62]]], mindspore.float32)
>>> net = ops.Ormqr()
>>> y = net(x, tau, other)
>>> print(y)
[[ 63.82713   -13.823125 -116.28614 ]
 [ -53.659264  -28.157839  -70.42702 ]
 [ -79.54292    24.00183   -41.34253 ]]
```

mindspore.ops.Orgqr

class mindspore.ops.Orgqr

计算 `mindspore.ops.Geqrf` 返回的正交矩阵 Q 的显式表示。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多细节请参考 `mindspore.ops.orgqr()`。

输入:

- **x** (Tensor) - `shape(*, M, N)` 的 Tensor, 表示二维或者三维矩阵。数据类型为 `float32`、`float64`、`complex64` 或者 `complex128`。

- **tau** (Tensor) - Householder 转换的反射系数, 其 shape 为 $(*, K)$, 其中 K 小于等于 N 。数据类型与 x 一致。

输出:

Tensor, 数据类型与 shape 与 *input* 一致。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[ -114.6, 10.9, 1.1], [-0.304, 38.07, 69.38], [-0.45, -0.17, 62.]]),
↳mindspore.float32)
>>> tau = Tensor(np.array([1.55, 1.94, 0.0]), mindspore.float32)
>>> net = ops.Orgqr()
>>> y = net(x, tau)
>>> print(y)
[[-0.54999995 -0.2128925  0.8137956 ]
 [ 0.47119996 -0.8752807  0.08240613]
 [ 0.69749993  0.42560163  0.57772595]]
```

mindspore.ops.Svd

class mindspore.ops.Svd (full_matrices=False, compute_uv=True)

计算一个或多个矩阵的奇异值分解。

详情请查看 `mindspore.ops.svd()`。

参数:

- **full_matrices** (bool, 可选) - 如果为 True, 则计算完整的 U 和 V 。否则仅计算前 P 个奇异向量, 其中, P 为 M 和 N 中的较小值, M 和 N 分别是输入矩阵的行和列。默认值: False。
- **compute_uv** (bool, 可选) - 如果这个参数为 True, 则计算 U 和 V , 否则只计算 S 。默认值: True。

输入:

- **input** (Tensor) - 待分解的矩阵。shape 为 $(*, M, N)$, 支持的数据类型为 float32 和 float64。

输出:

- **s** (Tensor) - 奇异值。shape 为 $(*, P)$ 。
- **u** (Tensor) - 左奇异向量。如果 `compute_uv` 为 False, 则值为 0。shape 为 $(*, M, P)$ 。如果 `full_matrices` 为 True, 则 shape 为 $(*, M, M)$ 。
- **v** (Tensor) - 右奇异向量。如果 `compute_uv` 为 False, 则值为 0。shape 为 $(*, N, P)$ 。如果 `full_matrices` 为 True, 则 shape 为 $(*, N, N)$ 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
```

(下页继续)

(续上页)

```
>>> from mindspore import ops
>>> ms.set_device(device_target="CPU")
>>> svd = ops.Svd(full_matrices=True, compute_uv=True)
>>> a = Tensor(np.array([[1, 2], [-4, -5], [2, 1]]).astype(np.float32))
>>> s, u, v = svd(a)
>>> print(s)
[7.0652843 1.040081 ]
>>> print(u)
[[ 0.30821905 -0.48819482 0.81649697]
 [-0.90613353 0.11070572 0.40824813]
 [ 0.2896955 0.8656849 0.4082479 ]]
>>> print(v)
[[ 0.63863593 0.769509 ]
 [ 0.769509 -0.63863593]]
```

20.4 Tensor 操作算子

20.4.1 Tensor 创建

接口名	概述	支持平台
<code>mindspore.ops.Eps</code>	创建一个与输入数据类型和 <code>shape</code> 都相同的 Tensor，元素值为对应数据类型能表达的最小值。	Ascend GPU CPU
<code>mindspore.ops.Eye</code>	返回一个主对角线上元素为 1，其余元素为 0 的 tensor。	Ascend GPU CPU
<code>mindspore.ops.Fill</code>	Fill 接口已弃用，请使用 <code>mindspore.ops.FillV2</code> 。	弃用
<code>mindspore.ops.LinSpace</code>	创建一个一维 Tensor，其包含 <code>steps</code> 个元素，这些元素在区间 <code>[start, end]</code> 内均匀分布。	Ascend GPU CPU
<code>mindspore.ops.OneHot</code>	返回一个 one-hot 类型的 Tensor。	Ascend GPU CPU
<code>mindspore.ops.Ones</code>	创建一个值全为 1 的 Tensor。	Ascend GPU CPU
<code>mindspore.ops.OnesLike</code>	返回值为 1 的 Tensor， <code>shape</code> 和数据类型与输入相同。	Ascend GPU CPU
<code>mindspore.ops.Zeros</code>	创建一个值全为 0 的 Tensor。	Ascend GPU CPU
<code>mindspore.ops.ZerosLike</code>	返回值为 0 的 Tensor，其 <code>shape</code> 和数据类型与输入 Tensor 相同。	Ascend GPU CPU

mindspore.ops.Eps

class mindspore.ops.Eps

创建一个与输入数据类型和 `shape` 都相同的 Tensor，元素值为对应数据类型能表达的最小值。

更多参考详见`mindspore.ops.eps()`。

输入：

- **x** (Tensor) - 用于获取其数据类型能表达的最小值的任意维度的 Tensor。数据类型必须为 float16、float32 或者 float64。

输出：

Tensor, 具有与 x 相同的数据类型和 shape, 填充了 x 数据类型的最小值。

异常:

- **TypeError** - x 不是 Tensor。
- **TypeError** - x 的数据类型不是 float16, float32 或者 float64。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> x = Tensor([4, 1, 2, 3], mindspore.float32)
>>> output = ops.Eps()(x)
>>> print(output)
[1.1920929e-07 1.1920929e-07 1.1920929e-07 1.1920929e-07]
```

mindspore.ops.Eye

class mindspore.ops.Eye

返回一个主对角线上元素为 1, 其余元素为 0 的 tensor。

输入:

- **n** (int) - 返回的行数。
- **m** (int) - 返回的列数。
- **t** (mindspore.dtype) - 返回的数据类型。

输出:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> eye = mindspore.ops.Eye()
>>> output = eye(3, 3, mindspore.int32)
>>> print(output)
[[1 0 0]
 [0 1 0]
 [0 0 1]]
>>>
>>> output = eye(3, 4, mindspore.float32)
>>> print(output)
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]]
```

mindspore.ops.Fill

class mindspore.ops.Fill

Fill 接口已弃用, 请使用`mindspore.ops.FillV2`。

更多参考详见`mindspore.ops.fill()`。

支持平台:

Deprecated

mindspore.ops.LinSpace

class mindspore.ops.LinSpace

创建一个一维 Tensor, 其包含 `steps` 个元素, 这些元素在区间 `[start, end]` 内均匀分布。

$$\begin{aligned} \text{step} &= (\text{end} - \text{start}) / (\text{steps} - 1) \\ \text{output} &= [\text{start}, \text{start} + \text{step}, \text{start} + 2 * \text{step}, \dots, \text{end}] \end{aligned}$$

输入:

- **start** (Tensor) - 区间的起始值。
- **stop** (Tensor) - 区间的末尾值。
- **num** (Union[int, Tensor]) - 元素数量。

输出:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> start = mindspore.tensor(3, mindspore.float32)
>>> stop = mindspore.tensor(10, mindspore.float32)
>>> num = 5
>>> output = mindspore.ops.LinSpace()(start, stop, num)
>>> print(output)
[ 3.   4.75  6.5   8.25 10. ]
>>>
>>> start = mindspore.tensor(-10, mindspore.float32)
>>> stop = mindspore.tensor(10, mindspore.float32)
>>> num = 5
>>> output = mindspore.ops.LinSpace()(start, stop, num)
>>> print(output)
[-10.  -5.   0.   5.  10.]
>>>
>>> start = mindspore.tensor(-10, mindspore.float32)
>>> stop = mindspore.tensor(10, mindspore.float32)
>>> num = 1
>>> output = mindspore.ops.LinSpace()(start, stop, num)
>>> print(output)
[-10.]
```

mindspore.ops.OneHot

class mindspore.ops.OneHot (axis=-1)

返回一个 one-hot 类型的 Tensor。

生成一个新的 Tensor，由索引 *indices* 表示的位置取值为 *on_value*，而在其他所有位置取值为 *off_value*。

说明：如果输入索引为秩 N ，则输出为秩 $N+1$ 。新轴在 *axis* 处创建。当执行设备是 Ascend 时，如果 *on_value* 为 int64 类型，则 *indices* 也必须为 int64 类型，且 *on_value* 和 *off_value* 的取值只能是 1 和 0。

参数：

- **axis** (int) - 指定 one-hot 的计算维度。例如，如果 *indices* 的 shape 为 (N, C) ，*axis* 为 -1，则输出 shape 为 (N, C, D) ，如果 *axis* 为 0，则输出 shape 为 (D, N, C) 。默认值：-1。

输入：

- **indices** (Tensor) - 输入索引，shape 为 $(X_0, \dots, X_n)$ 的 Tensor。数据类型必须为 int32 或 int64。
- **depth** (Union[int, Tensor]) - 输入的 Scalar，定义 one-hot 的深度。
- **on_value** (Tensor) - 当 $indices[j] = i$ 时，用来填充输出的值。数据类型必须为 int32、int64、float16 或 float32。
- **off_value** (Tensor) - 当 $indices[j] \neq i$ 时，用来填充输出的值。数据类型与 *on_value* 的相同。

输出：

Tensor，one-hot 类型的 Tensor。shape 为 $(X_0, \dots, X_{axis}, \text{depth}, X_{axis+1}, \dots, X_n)$ ，输出数据类型与 *on_value* 的相同。

异常：

- **TypeError** - *axis* 或 *depth* 不是 int。
- **TypeError** - *indices* 的数据类型不是 int32 或者 int64。
- **TypeError** - *on_value* 的数据类型不是 int32、int64、float16 或者 float32。
- **TypeError** - *indices*、*on_value* 或 *off_value* 不是 Tensor。
- **ValueError** - *axis* 不在 $[-1, \text{ndim}]$ 范围内。
- **ValueError** - *depth* 小于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> indices = Tensor(np.array([0, 1, 2]), mindspore.int32)
>>> depth, on_value, off_value = 3, Tensor(1.0, mindspore.float32), Tensor(0.0, mindspore.
↳float32)
>>> onehot = ops.OneHot()
>>> output = onehot(indices, depth, on_value, off_value)
>>> print(output)
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

mindspore.ops.Ones

class mindspore.ops.Ones

创建一个值全为 1 的 Tensor。

更多参考详见[mindspore.ops.ones\(\)](#)。

警告：参数 *shape* 在后续版本中将不再支持 Tensor 类型的输入。

输入：

- **shape** (Union[tuple[int], list[int], int, Tensor]) - 指定输出 Tensor 的 shape。
- **type** ([mindspore.dtype](#)) - 指定输出 Tensor 的数据类型。

输出：

Tensor, shape 和 dtype 由输入定义

异常：

- **TypeError** - 如果 *shape* 不是一个 int, 或元素为 int 的元组/列表/Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import ops
>>> ones = ops.Ones()
>>> output = ones((2, 2), mindspore.float32)
>>> print(output)
[[1. 1.]
 [1. 1.]]
>>> output = ones((3, 3), mindspore.float32)
>>> print(output)
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

mindspore.ops.OnesLike

class mindspore.ops.OnesLike

返回值为 1 的 Tensor, shape 和数据类型与输入相同。

更多参考详见[mindspore.ops.ones_like\(\)](#)。

输入：

- **input_x** (Tensor) - 任意维度的 Tensor。

输出：

Tensor, 具有与 *input_x* 相同的 shape 和类型, 并填充了 1。

支持平台：

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[0, 1], [2, 1]]).astype(np.int32))
>>> output = ops.OnesLike()(input_x)
>>> print(output)
[[1 1]
 [1 1]]
```

mindspore.ops.Zeros

class mindspore.ops.Zeros

创建一个值全为 0 的 Tensor。

第一个参数指定 Tensor 的 shape，第二个参数指定填充值的数据类型。

警告：参数 *shape* 在后续版本中将不再支持 Tensor 类型的输入。

输入：

- **shape** (Union[tuple[int], list[int], int, Tensor]) - 指定输出 Tensor 的 shape。
- **type** (mindspore.dtype) - 指定输出 Tensor 的数据类型。

输出：

Tensor, shape 和 dtype 由输入定义。

异常：

- **TypeError** - 如果 *shape* 不是一个 int，或元素为 int 的元组/列表/Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import ops
>>> zeros = ops.Zeros()
>>> output = zeros((2, 2), mindspore.float32)
>>> print(output)
[[0. 0.]
 [0. 0.]]
```

mindspore.ops.ZerosLike

class mindspore.ops.ZerosLike
返回值为 0 的 Tensor，其 shape 和数据类型与输入 Tensor 相同。

输入：

- **input_x** (Tensor) - 任意维度的输入 Tensor。

输出：

Tensor，具有与 *input_x* 相同的 shape 和数据类型，并用 0 填充。

异常：

- **TypeError** - *input_x* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> zeroslike = ops.ZerosLike()
>>> input_x = Tensor(np.array([[0, 1], [2, 1]]).astype(np.float32))
>>> output = zeroslike(input_x)
>>> print(output)
[[0. 0.]
 [0. 0.]]
```

20.4.2 随机生成算子

接口名	概述	支持平台
<i>mindspore.ops.Bernoulli</i>	以 <i>p</i> 的概率随机将输出的元素设置为 0 或 1，服从伯努利分布。	GPU CPU
<i>mindspore.ops.Gamma</i>	根据概率密度函数分布生成随机正值浮点数 <i>x</i> 。	Ascend
<i>mindspore.ops.Multinomial</i>	返回从输入 Tensor 对应行进行多项式概率分布采样出的 Tensor。	Ascend GPU CPU
<i>mindspore.ops.MultinomialWithReplacement</i>	返回一个 Tensor，其中每行包含从重复采样的多项式分布中抽取的 <i>numsamples</i> 个索引。	CPU
<i>mindspore.ops.RandomCategorical</i>	从分类分布中抽取样本。	Ascend GPU CPU
<i>mindspore.ops.RandomChoiceWithMask</i>	对输入进行随机取样，返回取样索引和掩码。	Ascend GPU CPU
<i>mindspore.ops.RandomGamma</i>	根据概率密度函数分布生成随机正值浮点数 <i>x</i> 。	CPU
<i>mindspore.ops.RandomPoisson</i>	根据离散概率密度函数分布生成随机非负数浮点数 <i>i</i> 。	GPU CPU
<i>mindspore.ops.Randperm</i>	生成从 0 到 <i>n-1</i> 的 <i>n</i> 个不重复随机样本。	Ascend GPU
<i>mindspore.ops.RandpermV2</i>	更多详情请查看： <i>mindspore.ops.randperm()</i> 。	CPU
<i>mindspore.ops.StandardLaplace</i>	生成符合标准 Laplace (mean=0, lambda=1) 分布的随机数。	Ascend GPU CPU

下页继续

表 16 – 续上页

<code>mindspore.ops.StandardNormal</code>	根据标准正态 (高斯) 随机数分布生成随机数。	Ascend GPU CPU
<code>mindspore.ops.UniformInt</code>	生成均匀分布在区间 $[minval, maxval)$ 中的随机数。	Ascend GPU CPU
<code>mindspore.ops.UniformReal</code>	生成随机的浮点数, 这些数均匀分布在 $[0,1)$ 范围内。	Ascend GPU CPU

mindspore.ops.Bernoulli

class mindspore.ops.Bernoulli (*seed=-1, offset=0*)

以 p 的概率随机将输出的元素设置为 0 或 1, 服从伯努利分布。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多细节请参考 `mindspore.ops.bernoulli()`。

参数:

- **seed** (int, 可选) - 随机种子, 用于生成随机数, 数值范围是 -1 或正整数, -1 代表取当前时间戳。默认值: -1。
- **offset** (int, 可选) - 用于在生成随机数序列时改变起始位置。默认值: 0。

输入:

- **x** (Tensor) - Tensor 的输入。
- **p** (Union[Tensor, float], 可选) - 成功概率。 p 中每个值代表输出 Tensor 中对应位置为 1 的概率, 如果是 Tensor, 其 shape 必须与 x 一致, 数值范围在 0 到 1 之间。默认值: 0.5。

输出:

- **y** (Tensor) - shape 和数据类型与 x 相同。

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input_x = Tensor([0.1, 0.2, 0.3], mindspore.float32)
>>> bernoulli = ops.Bernoulli()
>>> output = bernoulli(input_x, Tensor([1.0]))
>>> print(output)
[1. 1. 1.]
>>> input_p = Tensor([0.0, 1.0, 1.0], mindspore.float32)
>>> output = bernoulli(input_x, input_p)
>>> print(output)
[0. 1. 1.]
```

mindspore.ops.Gamma

class mindspore.ops.Gamma (seed=0, seed2=0)

根据概率密度函数分布生成随机正值浮点数 x 。函数定义如下：

$$P(x|,) = \frac{\exp(-x/\beta)}{\Gamma(\alpha)} \cdot x^{\alpha-1}$$

说明：

- 随机种子：通过一些复杂的数学算法，可以得到一组有规律的随机数，而随机种子就是这个随机数的初始值。随机种子相同，得到的随机数就不会改变。
- 全局随机种子和算子层随机种子均未设置或均为 0 时：生成完全随机数。
- 全局随机种子已设置，但算子层随机种子未设置时：将全局随机种子与 0 进行拼接。
- 全局随机种子未设置，但算子层随机种子已设置时：将 0 与算子层随机种子进行拼接。
- 全局随机种子和算子层随机种子均已设置时：将全局随机种子与算子层随机种子进行拼接。

警告：Ascend 后端不支持随机数重现功能，*seed* 和 *seed2* 参数不起作用。

参数：

- **seed** (int, 可选) - 算子层的随机种子，用于生成随机数。必须是非负的。默认值：0。
- **seed2** (int, 可选) - 全局的随机种子，和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值：0。

输入：

- **shape** (tuple) - 待生成的随机 Tensor 的 shape。只支持常量值。
- **alpha** (Tensor) - α 为 Gamma 分布的 shape parameter，主要决定了曲线的形状。其值必须大于 0。数据类型为 float32。
- **beta** (Tensor) - β 为 Gamma 分布的 inverse scale parameter，主要决定了曲线有多陡。其值必须大于 0。数据类型为 float32。

输出：

Tensor。shape 是输入 *shape*, *alpha*, *beta* 广播后的 shape。数据类型为 float32。

异常：

- **TypeError** - *seed* 或 *seed2* 的数据类型不是 int。
- **TypeError** - *alpha* 或 *beta* 不是 Tensor。
- **TypeError** - *alpha* 或 *beta* 的数据类型不是 float32。
- **ValueError** - *shape* 不是常量值。

支持平台：

Ascend

样例：


```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> shape = (3, 1, 2)
>>> alpha = Tensor(np.array([[3, 4], [5, 6]]), mstype.float32)
>>> beta = Tensor(np.array([1.0]), mstype.float32)
>>> gamma = ops.Gamma(seed=3)
>>> output = gamma(shape, alpha, beta)
>>> result = output.shape
>>> print(result)
(3, 2, 2)
```

mindspore.ops.Multinomial

class mindspore.ops.**Multinomial** (*seed=0, seed2=0, dtype=mstype.int32*)

返回从输入 Tensor 对应行进行多项式概率分布采样出的 Tensor。

说明:

- 输入的行不要求和为 1(在这种情况下, 我们使用值作为权重), 但必须是非负的、有限的, 并且具有非零和。
- 随机种子: 通过一些复杂的数学算法, 可以得到一组有规律的随机数, 而随机种子就是这个随机数的初始值。随机种子相同, 得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都没设置或都设置为 0: 完全随机。
- 全局的随机种子设置了, 算子层的随机种子未设置: 采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置, 算子层的随机种子设置了: 使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了: 全局的随机种子和算子层的随机种子拼接。

警告: Ascend 后端不支持随机数重现功能, *seed0* 和 *seed2* 参数不起作用。

参数:

- **seed** (int, 可选) - 算子层的随机种子, 用于生成随机数。必须是非负的。默认值: 0。
- **seed2** (int, 可选) - 全局的随机种子, 和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值: 0。
- **dtype** (mindspore.dtype, 可选) - 输出数据类型, 必须是 int32 或者 int64, 默认: mstype.int32。

输入:

- **x** (Tensor) - 包含累加概率和的输入 Tensor, 必须是一维或二维。
- **num_samples** (int) - 要抽取的样本数。

输出:

Tensor, 具有与输入相同的行。每行的采样索引数为 *num_samples*。

异常:

- **TypeError** - 如果 *seed* 或者 *seed2* 不是 int 类型。
- **TypeError** - 如果 *num_sample* 不是 int 类型。

- **TypeError** - 如果 *dtype* 不是 `int32` 或者 `int64` 类型。
- **ValueError** - 如果 *seed* 或者 *seed2* 小于零。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor([[0., 9., 4., 0.]], mstype.float32)
>>> multinomial = ops.Multinomial(seed=10)
>>> output = multinomial(x, 2)
>>> print(output)
[[1 1]]
```

`mindspore.ops.MultinomialWithReplacement`

class `mindspore.ops.MultinomialWithReplacement` (*numsamples*, *replacement=False*)

返回一个 `Tensor`，其中每行包含从重复采样的多项式分布中抽取的 *numsamples* 个索引。与 *Multinomial* 不同，*MultinomialWithReplacement* 允许多次选择相同的结果。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.multinomial _with_replacement()`。

说明：输入的行不需要和为 1（在这种情况下，使用输入行元素的值作为权重），但必须是非负的、有限的，并且具有非零和。

参数：

- **numsamples** (int) - 抽取样本量，必须大于零。
- **replacement** (bool, 可选) - 是否有放回地抽取。默认值：False。

输入：

- **x** (`Tensor`) - 包含概率的累积和的输入 `Tensor`，必须为一维或二维。
- **seed** (`Tensor`) - 如果将随机种子设置为 -1，并将 *offset* 设置为 0，则随机数生成器将使用随机种子生成。否则，将使用给定的随机数种子。支持的 *dtype*：int64。
- **offset** (int) - 为避免种子冲突设置的偏移量。支持的 *dtype*：int64。

输出：

`Tensor`，具有与输入相同的行。每行的采样索引数为 *numsamples*。

支持平台：

CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor([[0., 9., 4., 0.]], mstype.float32)
>>> seed = Tensor(2, mstype.int64)
>>> offset = Tensor(5, mstype.int64)
>>> multinomialwithreplacement = ops.MultinomialWithReplacement(numsamples=2,
    ↳replacement=True)
>>> output = multinomialwithreplacement(x, seed, offset)
>>> print(output)
[[1 1]]
```

mindspore.ops.RandomCategorical

class mindspore.ops.RandomCategorical(dtype=mstype.int64)

从分类分布中抽取样本。

警告：Ascend 后端不支持随机数生成结果复现，seed 参数将失效。

参数：

- **dtype** (mindspore.dtype, 可选) - 输出的类型。其值必须是 mstype.int16、mstype.int32 或 mstype.int64。默认值：mstype.int64。

输入：

- **logits** (Tensor) - 输入 Tensor，是一个 shape 为 (batch_size, num_classes) 的二维 Tensor。
- **num_sample** (int) - 要抽取的样本数。只允许使用常量值。
- **seed** (int) - 随机种子值，仅支持常量值。默认值：0。

输出：

- **output** (Tensor) - 输出 Tensor，其 shape 为 (batch_size, num_samples)。

异常：

- **TypeError** - 如果 dtype 不是 mstype.int16、mstype.int32 或 mstype.int64。
- **TypeError** - 如果 logits 不是 Tensor。
- **TypeError** - 如果 num_sample 或者 seed 不是 int。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import nn, ops, Tensor
>>> class Net(nn.Cell):
...     def __init__(self, num_sample):
...         super(Net, self).__init__()
...         self.random_categorical = ops.RandomCategorical(mindspore.int64)
...         self.num_sample = num_sample
...     def construct(self, logits, seed=0):
```

(下页继续)

(续上页)

```

...     return self.random_categorical(logits, self.num_sample, seed)
...
>>> x = np.random.random((10, 5)).astype(np.float32)
>>> net = Net(8)
>>> output = net(Tensor(x))
>>> result = output.shape
>>> print(result)
(10, 8)

```

mindspore.ops.RandomChoiceWithMask

class mindspore.ops.RandomChoiceWithMask (count=256, seed=0, seed2=0)

对输入进行随机取样，返回取样索引和掩码。

更多参考详见[mindspore.ops.choice_with_mask\(\)](#)。

说明：

- 随机种子：通过一些复杂的数学算法，可以得到一组有规律的随机数，而随机种子就是这个随机数的初始值。随机种子相同，得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都未设置或都设置为 0：完全随机。
- 全局的随机种子设置了，算子层的随机种子未设置：采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置，算子层的随机种子设置了：使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了：全局的随机种子和算子层的随机种子拼接。

参数：

- **count** (int, 可选) - 取样数量，必须大于 0。默认值：256。
- **seed** (int, 可选) - 算子层的随机种子，用于生成随机数。必须是非负的。默认值：0。
- **seed2** (int, 可选) - 全局的随机种子，和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值：0。

输入：

- **input_x** (Tensor[bool]) - 输入 Tensor，bool 类型。秩必须大于等于 1 且小于等于 5。

输出：

两个 Tensor，第一个为索引，第二个为掩码。

- **index** (Tensor) - 二维 Tensor，shape 为 (count, input_x)。
- **mask** (Tensor) - 一维 Tensor，shape 为 (count)。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> rnd_choice_mask = ops.RandomChoiceWithMask()
>>> input_x = Tensor(np.ones(shape=[240000, 4]).astype(np.bool_))
>>> output_y, output_mask = rnd_choice_mask(input_x)
>>> result = output_y.shape
>>> print(result)
(256, 2)
>>> result = output_mask.shape
>>> print(result)
(256,)
```

mindspore.ops.RandomGamma

class mindspore.ops.**RandomGamma** (*seed=0, seed2=0*)

根据概率密度函数分布生成随机正值浮点数 x 。

说明:

- 随机种子：通过一些复杂的数学算法，可以得到一组有规律的随机数，而随机种子就是这个随机数的初始值。随机种子相同，得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都未设置或都设置为 0：完全随机。
- 全局的随机种子设置了，算子层的随机种子未设置：采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置，算子层的随机种子设置了：使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了：全局的随机种子和算子层的随机种子拼接。

参数：

- **seed** (int, 可选) - 算子层的随机种子，用于生成随机数。必须是非负的。默认值：0。
- **seed2** (int, 可选) - 全局的随机种子，和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值：0。

输入：

- **shape** (tuple) - 待生成的随机 Tensor 的 shape。只支持常量值。
- **alpha** (Tensor) - α 为 RandomGamma 分布的 shape parameter，主要决定了曲线的形状，其值必须大于 0。数据类型为 float32。

输出：

Tensor。shape 是输入 *shape*, *alpha* 相连后的 shape。数据类型与 *alpha* 一致。

异常：

- **TypeError** - *seed* 或 *seed2* 的数据类型不是 int。
- **TypeError** - *shape* 或 *alpha* 不是 Tensor。
- **TypeError** - *shape* 或 *alpha* 的数据类型不是 float32。
- **ValueError** - *shape* 不是常量值。

支持平台：

CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> shape = Tensor(np.array([3, 1, 2]), mstype.int32)
>>> alpha = Tensor(np.array([[3, 4], [5, 6]]), mstype.float32)
>>> gamma = ops.RandomGamma(seed=3)
>>> output = gamma(shape, alpha)
>>> result = output.shape
>>> print(result)
(3, 1, 2, 2, 2)
```

mindspore.ops.RandomPoisson

class mindspore.ops.**RandomPoisson** (*seed=0, seed2=0, dtype=mstype.int64*)

根据离散概率密度函数分布生成随机非负数浮点数 i 。函数定义如下：

$$P(i) = \frac{\exp(-)^i}{i!}$$

说明：

- 随机种子：通过一些复杂的数学算法，可以得到一组有规律的随机数，而随机种子就是这个随机数的初始值。随机种子相同，得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都没设置或都设置为 0：完全随机。
- 全局的随机种子设置了，算子层的随机种子未设置：采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置，算子层的随机种子设置了：使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了：全局的随机种子和算子层的随机种子拼接。

参数：

- **seed** (int, 可选) - 算子层的随机种子，用于生成随机数。必须是非负的。默认值：0。
- **seed2** (int, 可选) - 全局的随机种子，和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值：0。
- **dtype** (mindspore.dtype, 可选) - 输出数据类型，默认值：mstype.int64。

输入：

- **shape** (tuple) - 待生成的随机 Tensor 的 shape，是一个一维 Tensor。数据类型为 int32 或 int64。
- **rate** (Tensor) - *rate* 为 Poisson 分布的 μ 参数，决定数字的平均出现次数。数据类型是其中之一：[float16, float32, float64, int32, int64]。

输出：

Tensor。shape 是 $(*shape, *rate.shape)$ ，数据类型由参数 *dtype* 指定。

异常：

- **TypeError** - *shape* 不是 Tensor 或数据类型不是 int32 或 int64。
- **TypeError** - *dtype* 数据类型不是 int32 或 int64。
- **ValueError** - *shape* 不是一维 Tensor。

- **ValueError** - *shape* 的元素存在负数。

支持平台：

GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> shape = Tensor(np.array([2, 3]), mstype.int32)
>>> rate = Tensor(np.array([2, 2]), mstype.int32)
>>> seed = 0
>>> seed2 = 0
>>> random_poisson = ops.RandomPoisson(seed=seed, seed2=seed2)
>>> output = random_poisson(shape, rate)
>>> print(output.shape)
(2, 3, 2)
```

mindspore.ops.Randperm

class mindspore.ops.**Randperm**(*max_length=1, pad=-1, dtype=mstype.int32*)

生成从 0 到 *n*-1 的 *n* 个不重复随机样本。如果 *max_length* > *n*，则末尾的 *max_length*-*n* 个元素使用 *pad* 填充。

参数：

- **max_length** (int) - 取样数量，必须大于 0。默认值：1。
- **pad** (int) - 填充值。默认值：-1。
- **dtype** (mindspore.dtype) - 输出的数据类型。默认值：mstype.int32。

输入：

- **n** (Tensor) - *shape* 为 (1,) 的输入 Tensor，其数据类型为 int32 或 int64，须在 [0, *max_length*] 内取值。

输出：

- **output** (Tensor) - *shape*: (*max_length*,)，数据类型为 *dtype*。

异常：

- **TypeError** - *max_length* 或 *pad* 不是 int 类型。
- **TypeError** - *n* 不是 Tensor。
- **TypeError** - *n* 包含非 int 元素。
- **TypeError** - *n* 包含负数。
- **TypeError** - *dtype* 不被支持。
- **ValueError** - *n* 超出 *dtype* 的有效范围。
- **ValueError** - *n* 大于 *max_length*。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> # The result of every execution is different because this operator will generate n
    random samples.
>>> randperm = ops.Randperm(max_length=30, pad=-1)
>>> n = Tensor([20], dtype=mindspore.int32)
>>> output = randperm(n)
>>> print(output)
[15 6 11 19 14 16 9 5 13 18 4 10 8 0 17 2 1 12 3 7
 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
```

mindspore.ops.RandpermV2

class mindspore.ops.RandpermV2 (seed=0, offset=0, dtype=mstype.int64)

```
prim = ops.RandpermV2(seed, offset, dtype)
out = prim(n)
```

等价于

```
ops.randperm(n, seed, offset, dtype)
```

更多详情请查看：[mindspore.ops.randperm\(\)](#)。

mindspore.ops.StandardLaplace

class mindspore.ops.StandardLaplace (seed=0, seed2=0)

生成符合标准 Laplace (mean=0, lambda=1) 分布的随机数。其概率密度函数为：

$$f(x) = \frac{1}{2} \exp(-|x|)$$

说明：

- 随机种子：通过一些复杂的数学算法，可以得到一组有规律的随机数，而随机种子就是这个随机数的初始值。随机种子相同，得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都没设置或都设置为 0：完全随机。
- 全局的随机种子设置了，算子层的随机种子未设置：采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置，算子层的随机种子设置了：使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了：全局的随机种子和算子层的随机种子拼接。

警告： Ascend 后端不支持随机数重现功能，*seed* 和 *seed2* 参数不起作用。

参数：

- **seed** (int, 可选) - 算子层的随机种子，用于生成随机数。必须是非负的。默认值：0。
- **seed2** (int, 可选) - 全局的随机种子，和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值：0。

输入：

- **shape** (Union[tuple[int], Tensor[int]]) - 待生成的 Tensor 的 shape。当为 tuple 类型时，只支持常量值；当为 Tensor 类型时，支持动态 Shape。

输出：

Tensor。shape 为输入 *shape*。数据类型支持 float32。

异常：

- **TypeError** - *seed* 或 *seed2* 不是 int。
- **TypeError** - *shape* 既不是 tuple，也不是 Tensor。
- **ValueError** - *seed* 或 *seed2* 不是非负的 int。
- **ValueError** - *shape* 为 tuple 时，包含非正的元素。
- **ValueError** - *shape* 为秩不等于 1 的 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import ops
>>> shape = (4, 16)
>>> stdlaplace = ops.StandardLaplace(seed=2)
>>> output = stdlaplace(shape)
>>> result = output.shape
>>> print(result)
(4, 16)
```

mindspore.ops.StandardNormal

class mindspore.ops.**StandardNormal** (*seed=0, seed2=0*)

根据标准正态（高斯）随机数分布生成随机数。

更多参考详见 `mindspore.ops.standard_normal()`。

说明：

- 随机种子：通过一些复杂的数学算法，可以得到一组有规律的随机数，而随机种子就是这个随机数的初始值。随机种子相同，得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都没设置或都设置为 0：完全随机。
- 全局的随机种子设置了，算子层的随机种子未设置：采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置，算子层的随机种子设置了：使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了：全局的随机种子和算子层的随机种子拼接。

警告： Ascend 后端不支持随机数重现功能，*seed* 和 *seed2* 参数不起作用。

参数：

- **seed** (int, 可选) - 算子层的随机种子, 用于生成随机数。必须是非负的。默认值: 0。
- **seed2** (int, 可选) - 全局的随机种子, 和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值: 0。

输入:

- **shape** (tuple) - 目标随机数 Tensor 的 shape。只允许常量值。支持类型为: int32、int64。

输出:

Tensor。shape 为输入 *shape*。数据类型支持 float32。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import ops
>>> shape = (3, 4)
>>> stdnormal = ops.StandardNormal(seed=2)
>>> output = stdnormal(shape)
>>> print(output)
[[-1.3031056  0.64198005 -0.65207404 -1.767485 ]
 [-0.91792876  0.6508565  -0.9098478  -0.14092612]
 [ 0.7806437  1.1585592  1.9676613  -0.00440959]]
```

mindspore.ops.UniformInt

class mindspore.ops.UniformInt (seed=0, seed2=0)

生成均匀分布在区间 $[minval, maxval)$ 中的随机数。离散概率函数定义如下:

$$P(i|a, b) = \frac{1}{b - a + 1},$$

其中 a 为分布区间的最小值 *minval*, b 为分布区间的最大值 *maxval*。

说明:

- *minval* 中的数值在广播后必须严格小于 *maxval*。
- 随机种子: 通过一些复杂的数学算法, 可以得到一组有规律的随机数, 而随机种子就是这个随机数的初始值。随机种子相同, 得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都未设置或都设置为 0: 完全随机。
- 全局的随机种子设置了, 算子层的随机种子未设置: 采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置, 算子层的随机种子设置了: 使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了: 全局的随机种子和算子层的随机种子拼接。

警告: Ascend 后端不支持随机数重现功能, *seed* 和 *seed2* 参数不起作用。

参数:

- **seed** (int, 可选) - 算子层的随机种子, 用于生成随机数。必须是非负的。默认值: 0。

- **seed2** (int, 可选) - 全局的随机种子, 和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值: 0。

输入:

- **shape** (Union[tuple, Tensor]) - 目标 Tensor 的 shape。只允许常量值。
- **minval** (Tensor) - 分布参数 a 。决定可能生成的最小值, 数据类型为 int32。需为标量。
- **maxval** (Tensor) - 分布参数 b 。决定生成随机数的上限, 数据类型为 int32。需为标量。

输出:

Tensor。shape 为输入 *shape*, 数据类型支持 int32。

异常:

- **TypeError** - *seed* 或 *seed2* 不是 int 类型。
- **TypeError** - *shape* 不是 tuple 或 Tensor。
- **TypeError** - *minval* 或 *maxval* 不是 Tensor。
- **ValueError** - *shape* 不是常量值。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> shape = (2, 4)
>>> minval = Tensor(1, mstype.int32)
>>> maxval = Tensor(5, mstype.int32)
>>> uniform_int = ops.UniformInt(seed=10)
>>> output = uniform_int(shape, minval, maxval)
>>> result = output.shape
>>> print(result)
(2, 4)
```

mindspore.ops.UniformReal

class mindspore.ops.UniformReal (*seed=0, seed2=0*)

生成随机的浮点数, 这些数均匀分布在 [0,1) 范围内。

说明:

- 随机种子: 通过一些复杂的数学算法, 可以得到一组有规律的随机数, 而随机种子就是这个随机数的初始值。随机种子相同, 得到的随机数就不会改变。
- 全局的随机种子和算子层的随机种子都没设置或都设置为 0: 完全随机。
- 全局的随机种子设置了, 算子层的随机种子未设置: 采用全局的随机种子和 0 拼接。
- 全局的随机种子未设置, 算子层的随机种子设置了: 使用 0 和算子层的随机种子拼接。
- 全局的随机种子和算子层的随机种子都设置了: 全局的随机种子和算子层的随机种子拼接。
- 目前在 Ascend 平台上不支持 *shape* 为 Tensor, CPU/GPU 平台支持。当输入为 Tensor 的时候, 支持的数据类型:
 - GPU: int32、int64。

- CPU: int16、int32、int64。

警告: Ascend 后端不支持随机数重现功能, *seed* 和 *seed2* 参数不起作用。

参数:

- **seed** (int, 可选) - 算子层的随机种子, 用于生成随机数。必须是非负的。默认值: 0。
- **seed2** (int, 可选) - 全局的随机种子, 和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值: 0。

输入:

- **shape** (Union[tuple, Tensor]) - 待生成的 Tensor 的 shape。只支持常量值。

输出:

Tensor。它的 shape 为输入 *shape*。数据类型为 float32。

异常:

- **TypeError** - *seed* 或 *seed2* 不是 int。
- **TypeError** - *shape* 不是 tuple 或 Tensor。
- **ValueError** - *shape* 不是常量值。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import ops
>>> shape = (2, 2)
>>> uniformreal = ops.UniformReal(seed=2)
>>> output = uniformreal(shape)
>>> result = output.shape
>>> print(result)
(2, 2)
```

20.4.3 Array 操作

接口名	概述	支持平台
<code>mindspore.ops.AffineGrid</code>	基于一批仿射矩阵 theta 生成一个 2D 或 3D 的流场 (采样网格)。	Ascend GPU CPU
<code>mindspore.ops.BatchToSpace</code>	将批处理数据重新排列到空间数据中。	Ascend GPU
<code>mindspore.ops.BatchToSpaceND</code>	<code>mindspore.ops.BatchToSpaceND</code> 从 2.0 版本开始已被弃用, 并将在未来版本中被移除, 建议使用 <code>mindspore.ops.batch_to_space_nd()</code> 代替。	弃用
<code>mindspore.ops.BroadcastTo</code>	更多详情请查看: <code>mindspore.ops.broadcast_to()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Cast</code>	转换输入 Tensor 的数据类型。	Ascend GPU CPU

下页继续

表 17 – 续上页

<code>mindspore.ops.ChannelShuffle</code>	将 <code>shape</code> 为 $(*, C, H, W)$ 的 <code>Tensor</code> 的通道划分成 g 组, 并按如下方式重新排列 $(*, \frac{C}{g}, g, H*W)$, 同时在最终输出中保持原始 <code>Tensor</code> 的 <code>shape</code> 。	Ascend CPU
<code>mindspore.ops.Col2Im</code>	将一个行向量重新排列成图像。	Ascend GPU CPU
<code>mindspore.ops.Concat</code>	更多详情请查看: <code>mindspore.ops.cat()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Cummax</code>	更多详情请查看: <code>mindspore.ops.cummax()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Cummin</code>	返回输入 <code>Tensor</code> 在指定轴上的累积最小值与其对应的索引。	Ascend GPU CPU
<code>mindspore.ops.CumProd</code>	计算 x 沿着指定 <code>axis</code> 的元素累积积。	Ascend GPU CPU
<code>mindspore.ops.CumSum</code>	计算输入 <code>Tensor</code> 在指定轴上的累加和。	Ascend GPU CPU
<code>mindspore.ops.DataFormatDimMap</code>	返回源数据格式中的目标数据格式的维度索引。	Ascend GPU CPU
<code>mindspore.ops.DepthToSpace</code>	将深度数据重新排列到空间维度中。	Ascend GPU CPU
<code>mindspore.ops.Diag</code>	更多详情请查看: <code>mindspore.ops.diag()</code> 。	Ascend GPU CPU
<code>mindspore.ops.DType</code>	计算输入 <code>Tensor</code> 的数据类型, 且返回的数据类型为 <code>mindspore.dtype</code> 。	Ascend GPU CPU
<code>mindspore.ops.ExpandDims</code>	更多详情请查看: <code>mindspore.ops.expand_dims()</code> 。	Ascend GPU CPU
<code>mindspore.ops.FillDiagonal</code>	填充至少具有二维的 <code>Tensor</code> 的主对角线。	Ascend GPU CPU
<code>mindspore.ops.FillV2</code>	创建一个 <code>Tensor</code> , 其 <code>shape</code> 由 <code>shape</code> 指定, 其值则由 <code>value</code> 进行填充。	Ascend GPU CPU
<code>mindspore.ops.FloatStatus</code>	确定元素是否包含非数字 (NaN)、正无穷或负无穷。	GPU
<code>mindspore.ops.Fmax</code>	逐元素计算输入 <code>Tensor</code> 的最大值。	CPU
<code>mindspore.ops.Gather</code>	更多详情请查看: <code>mindspore.ops.gather()</code> 。	Ascend GPU CPU
<code>mindspore.ops.GatherD</code>	更多详情请查看: <code>mindspore.ops.gather_d()</code> 。	Ascend GPU CPU
<code>mindspore.ops.GatherNd</code>	更多详情请查看: <code>mindspore.ops.gather_nd()</code> 。	Ascend GPU CPU
<code>mindspore.ops.HammingWindow</code>	使用输入窗口长度计算汉明窗口函数。	Ascend GPU CPU
<code>mindspore.ops.Heaviside</code>	计算输入中每个元素的 <code>Heaviside</code> 步长函数。	Ascend GPU CPU
<code>mindspore.ops.HistogramFixedWidth</code>	返回一个 <code>rank</code> 为 1 的直方图, 该直方图中的每个组的值表示数量。	Ascend GPU
<code>mindspore.ops.Hypot</code>	将输入 <code>Tensor</code> 的逐个元素作为直角三角形的直角边, 并计算其斜边的值。	Ascend GPU CPU
<code>mindspore.ops.Identity</code>	更多详情请查看: <code>mindspore.ops.deepcopy()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Igamma</code>	计算正则化的下层不完全伽马函数。	Ascend GPU CPU
<code>mindspore.ops.Igammac</code>	计算正则化的上层不完全伽马函数。	Ascend GPU CPU
<code>mindspore.ops.Im2Col</code>	从一个 <code>batch</code> 的输入 <code>Tensor</code> 中提取滑动局部块。	Ascend GPU CPU
<code>mindspore.ops.IndexAdd</code>	将 <code>Tensor y</code> 加到 <code>Tensor x</code> 的指定 <code>axis</code> 和 <code>indices</code> 。	Ascend GPU CPU
<code>mindspore.ops.IndexFill</code>	按 <code>index</code> 中给定的顺序选择索引, 将输入 <code>Tensor x</code> 的 <code>dim</code> 维度下的元素用 <code>value</code> 的值填充。	Ascend GPU CPU
<code>mindspore.ops.IndexPut</code>	根据 <code>indices</code> 中的下标值, 使用 <code>x2</code> 中的数值替换 <code>x1</code> 中的相应元素的值。	Ascend CPU
<code>mindspore.ops.InplaceAdd</code>	将 <code>input_v</code> 添加到 <code>x</code> 的特定行。	Ascend GPU CPU

下页继续

表 17 – 续上页

<code>mindspore.ops.InplaceIndexAdd</code>	逐元素将一个 Tensor <i>updates</i> 添加到原 Tensor <i>var</i> 的指定轴和索引处。	Ascend CPU
<code>mindspore.ops.InplaceSub</code>	从 <i>x</i> 的特定行减去 <i>input_v</i> 。	Ascend GPU CPU
<code>mindspore.ops.InplaceUpdate</code>	InplaceUpdate 接口已废弃, 请使用接口 <code>mindspore.ops.InplaceUpdateV2</code> 替换, 废弃版本 2.0。	弃用
<code>mindspore.ops.InplaceUpdateV2</code>	根据 <i>indices</i> , 将 <i>x</i> 中的某些值更新为 <i>v</i> 。	GPU CPU
<code>mindspore.ops.InvertPermutation</code>	计算索引的逆置换。	Ascend GPU CPU
<code>mindspore.ops.IsClose</code>	返回一个 bool 型 Tensor , 表示 <i>input</i> 的每个元素与 <i>other</i> 的每个元素在给定容忍度内是否“接近”。	Ascend GPU CPU
<code>mindspore.ops.Lcm</code>	逐个元素计算输入 Tensor 的最小公倍数。	Ascend GPU CPU
<code>mindspore.ops.LeftShift</code>	将 Tensor 每个位置的值向左移动若干个比特位。	Ascend GPU CPU
<code>mindspore.ops.LogSpace</code>	返回一个大小为 <i>steps</i> 的 1-D Tensor , 其值从 $base^{start}$ 到 $base^{end}$, 以 <i>base</i> 为底数。	Ascend GPU CPU
<code>mindspore.ops.LuUnpack</code>	将 <i>LU_data</i> 和 <i>LU_pivots</i> 还原为 P, L, U 矩阵, 其中 P 为置换矩阵, L 为下三角矩阵, U 为上三角矩阵。	GPU CPU
<code>mindspore.ops.MaskedFill</code>	更多详情请查看: <code>mindspore.ops.masked_fill()</code> 。	Ascend GPU CPU
<code>mindspore.ops.MaskedScatter</code>	返回一个 Tensor 。	Ascend CPU
<code>mindspore.ops.MaskedSelect</code>	更多详情请查看: <code>mindspore.ops.masked_select()</code> 。	Ascend GPU CPU
<code>mindspore.ops.MatrixBandPart</code>	提取一个 Tensor 中每个矩阵的中心带, 中心带之外的所有值都设置为零。	Ascend GPU CPU
<code>mindspore.ops.MatrixDiagPartV3</code>	返回 Tensor 的对角线部分。	Ascend GPU CPU
<code>mindspore.ops.MatrixDiagV3</code>	构造以输入 Tensor 为对角线的矩阵。	Ascend GPU CPU
<code>mindspore.ops.MatrixSetDiagV3</code>	更新批处理矩阵对角线的值。	Ascend GPU CPU
<code>mindspore.ops.MatrixSolve</code>	求解线性方程组。	Ascend CPU
<code>mindspore.ops.Meshgrid</code>	从给定的 Tensor 生成网格矩阵。	Ascend GPU CPU
<code>mindspore.ops.Mvlgamma</code>	逐元素计算 <i>p</i> 维多元对数伽马函数值。	Ascend GPU CPU
<code>mindspore.ops.NanToNum</code>	更多详情请查看: <code>mindspore.ops.nan_to_num()</code> 。	Ascend CPU
<code>mindspore.ops.NonZero</code>	返回所有非零元素的索引位置。	Ascend GPU CPU
<code>mindspore.ops.ParallelConcat</code>	根据第一个维度连接输入 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.PopulationCount</code>	计算二进制数中 1 的个数。	Ascend GPU CPU
<code>mindspore.ops.RandomShuffle</code>	沿着 Tensor 的第一个维度进行随机打乱操作。	Ascend GPU CPU
<code>mindspore.ops.Range</code>	更多详情请查看: <code>mindspore.ops.range()</code> 。	GPU CPU
<code>mindspore.ops.Rank</code>	返回输入 Tensor 的秩。	Ascend GPU CPU
<code>mindspore.ops.Renorm</code>	对 Tensor 沿着指定维度 <i>dim</i> 进行重新规范化, 要求每个子 Tensor 的 <i>p</i> 范数不超过 <i>maxnorm</i> 。	Ascend GPU CPU
<code>mindspore.ops.Reshape</code>	更多详情请查看: <code>mindspore.ops.reshape()</code> 。	Ascend GPU CPU
<code>mindspore.ops.ReverseSequence</code>	对输入序列进行部分反转。	Ascend GPU CPU
<code>mindspore.ops.ReverseV2</code>	更多详情请查看: <code>mindspore.ops.flip()</code> 。	Ascend GPU CPU
<code>mindspore.ops.RightShift</code>	将 Tensor <i>input_x</i> 的每个元素右移 Tensor <i>input_y</i> 中对应位数。	Ascend GPU CPU

下页继续

表 17 – 续上页

<code>mindspore.ops.ScatterNd</code>	更多详情请查看: <code>mindspore.ops.scatter_nd()</code> 。	Ascend GPU CPU
<code>mindspore.ops.ScatterNdDiv</code>	将稀疏除法应用于张量中的单个值或切片。	GPU CPU
<code>mindspore.ops.ScatterNdMax</code>	对张量中的单个值或切片计算稀疏最大值。	Ascend GPU CPU
<code>mindspore.ops.ScatterNdMin</code>	对 Tensor 中的单个值或切片应用稀疏最小值。	Ascend GPU CPU
<code>mindspore.ops.ScatterNdMul</code>	对张量中的单个值或切片应用稀疏乘法。	GPU CPU
<code>mindspore.ops.SearchSorted</code>	返回元素能够插入输入 tensor 的位置索引, 以维持原 tensor 的递增顺序。	Ascend GPU CPU
<code>mindspore.ops.Select</code>	更多详情请查看: <code>mindspore.ops.select()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Shape</code>	返回输入 Tensor 的 shape 。	Ascend GPU CPU
<code>mindspore.ops.Size</code>	返回一个 Scalar , 类型为整数, 表示输入 Tensor 的大小, 即 Tensor 中元素的总数。	Ascend GPU CPU
<code>mindspore.ops.Slice</code>	根据指定 shape 对输入 Tensor 进行切片。	Ascend GPU CPU
<code>mindspore.ops.Sort</code>	根据指定的轴对输入 Tensor 的元素进行排序, 默认为升序排序。	Ascend GPU CPU
<code>mindspore.ops.SpaceToBatchND</code>	将空间维度划分为对应大小的块, 并在批次维度重排张量。	Ascend GPU CPU
<code>mindspore.ops.SpaceToDepth</code>	将空间维度分块, 增加 Tensor 深度。	Ascend GPU CPU
<code>mindspore.ops.SparseGatherV2</code>	基于指定的索引和 axis 返回输入 Tensor 的切片。	Ascend GPU
<code>mindspore.ops.Split</code>	根据指定的轴和分割数量对输入 Tensor 进行分割。	Ascend GPU CPU
<code>mindspore.ops.Squeeze</code>	返回删除指定 axis 中大小为 1 的维度后的 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.Stack</code>	在指定轴上对输入 Tensor 序列进行堆叠。	Ascend GPU CPU
<code>mindspore.ops.StridedSlice</code>	更多详情请查看: <code>mindspore.ops.strided_slice()</code> 。	Ascend GPU CPU
<code>mindspore.ops.TensorScatterAdd</code>	根据指定的更新值 updates 和输入索引 indices , 使用加法运算更新输入 Tensor 的值。	Ascend GPU CPU
<code>mindspore.ops.TensorScatterDiv</code>	根据指定的更新值 updates 和输入索引 indices , 进行除法运算更新输入 Tensor 的值。	GPU CPU
<code>mindspore.ops.TensorScatterMax</code>	根据指定的更新值 updates 和输入索引 indices , 通过最大值运算将结果赋值到输出 Tensor 中。	GPU CPU
<code>mindspore.ops.TensorScatterMin</code>	根据指定的更新值 updates 和输入索引 indices , 计算原值与更新值的较小值并更新原值, 返回更新后的 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.TensorScatterMul</code>	根据指定的更新值 updates 和输入索引 indices , 使用乘法运算更新输入 Tensor 的值。	GPU CPU
<code>mindspore.ops.TensorScatterSub</code>	根据指定的更新值 input_x 和输入索引 indices , 进行减法运算更新输入 Tensor 的值。	Ascend GPU CPU
<code>mindspore.ops.TensorScatterUpdate</code>	根据指定的更新值 update 和输入索引 indices , 通过更新操作更新输出 Tensor 的值。	Ascend GPU CPU
<code>mindspore.ops.TensorShape</code>	返回输入 Tensor 的 Shape 。	Ascend GPU CPU
<code>mindspore.ops.Tile</code>	按照给定的次数复制输入 Tensor 。	Ascend GPU CPU
<code>mindspore.ops.Trace</code>	更多详情请查看: <code>mindspore.ops.trace()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Transpose</code>	更多详情请查看: <code>mindspore.ops.transpose()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Tril</code>	返回单个或一批二维矩阵下三角形部分, 其他位置的元素将被置零。	Ascend GPU CPU

下页继续

表 17 – 续上页

<code>mindspore.ops.TrilIndices</code>	计算二维矩阵的下三角元素的索引，并将它们作为一个 Tensor 返回。	Ascend GPU CPU
<code>mindspore.ops.Triu</code>	更多详情请查看: <code>mindspore.ops.triu()</code> 。	Ascend GPU CPU
<code>mindspore.ops.TriuIndices</code>	计算 <code>row * col</code> 行列矩阵的上三角元素的索引，并将它们作为一个 <code>2xN</code> 的 Tensor 返回。	Ascend GPU CPU
<code>mindspore.ops.Unique</code>	返回输入 Tensor 的唯一元素以及其对应的每个值的索引。	Ascend GPU CPU
<code>mindspore.ops.UniqueConsecutive</code>	对输入 Tensor 中连续且重复的元素去重。	Ascend GPU CPU
<code>mindspore.ops.UniqueWithPad</code>	<code>ops.UniqueWithPad</code> 从 2.4 版本开始已被弃用，并将在未来版本中被移除。	弃用
<code>mindspore.ops.UnsortedSegmentMax</code>	沿分段计算输入 Tensor 的最大值。	Ascend GPU CPU
<code>mindspore.ops.UnsortedSegmentMin</code>	沿分段计算输入 Tensor 的最小值。	Ascend GPU CPU
<code>mindspore.ops.UnsortedSegmentProd</code>	沿分段计算输入 Tensor 元素的乘积。	Ascend GPU CPU
<code>mindspore.ops.UnsortedSegmentSum</code>	更多详情请查看: <code>mindspore.ops.unsorted_segment_sum()</code> 。	Ascend GPU CPU
<code>mindspore.ops.Unstack</code>	根据指定轴对输入矩阵进行分解，与 Stack 函数操作相反。	Ascend GPU CPU

mindspore.ops.AffineGrid

class `mindspore.ops.AffineGrid` (*align_corners=False*)
基于一批仿射矩阵 `theta` 生成一个 2D 或 3D 的流场（采样网格）。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见`mindspore.ops.affine_grid()`。

参数：

- **align_corners** (bool, 可选) - 在几何上，我们将输入的像素视为正方形而不是点。如果设置为 `True`，则极值 -1 和 1 指输入像素的中心。如果设置为 `False`，则极值 -1 和 1 指输入像素的边角，从而使采样与分辨率无关。默认值：`False`。

输入：

- **theta** (**Tensor**) - 仿射矩阵输入，数据类型为 `float16` 和 `float32`。当输入为 2D `grid` 时，其 `shape` 为 $(N, 2, 3)$ ；当输入为 3D `grid` 时，其 `shape` 为 $(N, 3, 4)$ 。
- **output_size** (tuple[int]) - 目标输出图像大小。指格式为 (N, C, H, W) 的 2D `grid` 或格式为 (N, C, D, H, W) 的 3D `grid` 的大小。

输出：

Tensor，其数据类型与 `theta` 相同，当输入为 2D `grid` 时，其 `shape` 为 $(N, H, W, 2)$ ；当输入为 3D `grid` 时，其 `shape` 为 $(N, D, H, W, 3)$ 。

支持平台：

Ascend GPU CPU

样例：


```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> affinegrid = ops.AffineGrid(align_corners=False)
>>> theta = Tensor([[[[0.8, 0.5, 0], [-0.5, 0.8, 0]]], mindspore.float32)
>>> out_size = (1, 3, 2, 3)
>>> output = affinegrid(theta, out_size)
>>> print(output)
[[[[-0.78333336 -0.06666666]
[-0.25         -0.4         ]
[ 0.28333336 -0.73333335]]]
[[-0.28333336  0.73333335]
[ 0.25         0.4         ]
[ 0.78333336  0.06666666]]]]]
```

`mindspore.ops.BatchToSpace`

class `mindspore.ops.BatchToSpace` (*block_size, crops*)

将批处理数据重新排列到空间数据中。

此操作将批处理维度 *N* 拆分为 *block_size* 大小的块 (blocks)，输出 Tensor 的维度 *N*，即为拆分后对应的块数。输出 Tensor 的 *H*、*W* 维分别是原 *H*、*W* 维和 *block_size* 在给定裁剪量情况下的乘积。

参数：

- **block_size** (int) - 指定拆分的块大小，其值不能小于 2。
- **crops** (Union[list(int), tuple(int)]) - 指定 *H* 和 *W* 维度上的裁剪值，包含 2 个列表。每个列表包含 2 个整数。所有值都必须不小于 0。*crops[i]* 表示指定空间维度 *i* 的裁剪值，该维度对应于输入维度 *i+2*。要求 $input\_shape[i+2] * block\_size > crops[i][0] + crops[i][1]$ 。

输入：

- **input_x** (Tensor) - 输入 Tensor。必须是四维，第零维度（维度 *n*）的大小必须可被 *block_size* 的乘积整除。数据类型为 float16 或 float32。

输出：

Tensor，数据类型与输入 Tensor 相同。假设输入 shape 为 (*n, c, h, w*)，经过 *block_size* 和 *crops* 计算后。输出 shape 将为 (*n', c', h', w'*)，其中

- $n' = n / (block\_size * block\_size)$
- $c' = c$
- $h' = h * block\_size - crops[0][0] - crops[0][1]$
- $w' = w * block\_size - crops[1][0] - crops[1][1]$

异常：

- **TypeError** - 如果 *block_size* 或 *crops* 的元素不是 int。
- **TypeError** - 如果 *crops* 既不是 list 也不是 tuple。
- **ValueError** - 如果 *block_size* 的值小于 2。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> block_size = 2
>>> crops = [[0, 0], [0, 0]]
>>> batch_to_space = ops.BatchToSpace(block_size, crops)
>>> input_x = Tensor(np.array([[[[1]]], [[2]]], [[3]]], [[4]])), mindspore.float32)
>>> output = batch_to_space(input_x)
>>> print(output)
[[[1.  2.]
  [3.  4.]]]]
```

mindspore.ops.BatchToSpaceND

class mindspore.ops.BatchToSpaceND (*block_shape, crops*)

mindspore.ops.BatchToSpaceND 从 2.0 版本开始已被弃用，并将在未来版本中被移除，建议使用 *mindspore.ops.batch_to_space_nd()* 代替。

支持平台：

Deprecated

mindspore.ops.BroadcastTo

class mindspore.ops.BroadcastTo (*shape*)

```
prim = ops.BroadcastTo(shape)
out = prim(input)
```

等价于

```
ops.broadcast_to(input, shape)
```

更多详情请查看：*mindspore.ops.broadcast_to()*。

mindspore.ops.Cast

class mindspore.ops.Cast

转换输入 Tensor 的数据类型。

说明：将复数转换为 bool 类型的时候，不考虑复数的虚部，只要实部不为零，返回 True，否则返回 False。

输入：

- **input** (Union[*Tensor*, *Number*]) - 输入要进行数据类型转换的 Tensor，其 shape 为 $(x_1, x_2, \dots, x_R)$ 。
- **dtype** (*dtype.Number*) - 指定转换的数据类型。仅支持常量值。

输出：

Tensor，其 shape 与 *input* 相同，即 $(x_1, x_2, \dots, x_R)$ 。

异常：

- **TypeError** - *input* 既不是 Tensor 也不是数值型。
- **TypeError** - *dtype* 不是数值型。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_np = np.random.randn(2, 3, 4, 5).astype(np.float32)
>>> input = Tensor(input_np)
>>> dtype = mindspore.int32
>>> cast = ops.Cast()
>>> output = cast(input, dtype)
>>> print(output.dtype)
Int32
>>> print(output.shape)
(2, 3, 4, 5)
```

mindspore.ops.ChannelShuffle

class mindspore.ops.ChannelShuffle(*group*)

将 shape 为 $(*, C, H, W)$ 的 Tensor 的通道划分成 g 组，并按如下方式重新排列 $(*, \frac{C}{g}, g, H * W)$ ，同时在最终输出中保持原始 Tensor 的 shape。

警告：这是一个实验性 API，后续可能修改或删除。

更多详情请查看：[*mindspore.ops.channel_shuffle*](#)。

参数：

- **group** (int) - 通道划分数目。

输入：

- **x** (Tensor) - 被划分的输入 Tensor。shape 为 $(*, C, H, W)$ ，数据类型为 float16、float32、int8、int16、int32、int64、uint8、uint16、uint32 或 uint64。

输出：

Tensor，数据类型与 x 相同，shape 为 $(*, C, H, W)$ 。

支持平台：

Ascend CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> group = 2
>>> x = Tensor(np.arange(1 * 4 * 2 * 2).reshape(1, 4, 2, 2).astype(np.int16))
>>> channel_shuffle_func = ops.ChannelShuffle(group)
>>> y = channel_shuffle_func(x)
```

(下页继续)

(续上页)

```
>>> print(y)
[[[ 0  1]
  [ 2  3]
  [ 8  9]
  [10 11]
  [ 4  5]
  [ 6  7]
  [12 13]
  [14 15]]]]
```

mindspore.ops.Col2Im

class mindspore.ops.Col2Im(*kernel_size, dilation=1, padding=0, stride=1*)

将一个行向量重新排列成图像。通常用于从一组图像补丁（或滑动局部块）中重建图像。

假设输入 Tensor 的 shape 为 $(N, C, \prod(\text{kernel\_size}), L)$ ， N 代表 Batch 数量， C 代表 Channel 数量， $\prod(\text{kernel\_size})$ 代表滑窗大小， L 代表滑窗总数。Col2Im 将这些局部块组合成 shape 为 $(N, C, \text{output\_size}[0], \text{output\_size}[1], \dots)$ 的 Tensor 作为输出，对于滑块间的重叠值采取求和策略。

输入与输出的 shape 之间的关系可以表示为：

$$L = \prod_d \left\lfloor \frac{\text{output\_size}[d] + 2 \times \text{padding}[d] - \text{dilation}[d] \times (\text{kernel\_size}[d] - 1) - 1}{\text{stride}[d]} + 1 \right\rfloor$$

式中 d 代表高度与宽度上两个维度，*dilation*、*padding* 和 *stride* 决定了滑窗如何滑动与检索元素。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **kernel_size** (Union[int, tuple[int], list[int]]) - 滑窗大小，由两个正整数组成，分别代表滑窗的高度与宽度。如果数据类型为 int，代表不同方向上的填充大小相等。取值必须由用户指定。
- **dilation** (Union[int, tuple[int], list[int]], 可选) - 滑窗之间的间距，由两个正整数组成，分别代表横向与纵向上滑窗移动时与上一个滑窗间的距离。如果数据类型为 int，代表不同方向上的填充大小相等。默认值：1。
- **padding** (Union[int, tuple[int], list[int]], 可选) - 滑窗取数前在输入 x 周围隐式填充 0（若需）的范围，由两个正整数组成，分别代表横向与纵向上的填充范围。如果数据类型为 int，代表不同方向上的填充范围大小相等。默认值：0。
- **stride** (Union[int, tuple[int], list[int]], 可选) - 滑窗移动的步长，由两个正整数组成，分别代表滑窗在横向与纵向上的移动步长。如果数据类型为 int，代表不同方向上的移动步长相等。默认值：1。

输入：

- **x** (Tensor) - 4D 输入 Tensor。
- **output_size** (Tensor) - 1D Tensor，输出 Tensor 的后两维的 shape，包含 2 个元素且其数据类型为 int32 或 int64。

输出：

4D Tensor，类型与输入 x 一致。

异常：

- **TypeError** - 输入 *kernel_size*、*dilation*、*padding*、*stride* 的数据类型不是 int、tuple[int] 或 list[int] 之一。
- **ValueError** - 输入 *kernel_size*、*dilation*、*padding*、*stride* 的值小于等于 0 或者其中元素的个数大于 2。

- **ValueError** - `x.shape[2] != kernel_size[0] * kernel_size[1]`。
- **ValueError** - `x.shape[3]` 与计算出的滑动块数量不匹配。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(input_data=np.random.rand(16, 16, 4, 25), dtype=mstype.float32)
>>> output_size = Tensor(input_data=[8, 8], dtype=mstype.int32)
>>> col2im = ops.Col2Im(kernel_size=[2, 2], dilation=[2, 2], padding=[2, 2], stride=[2, 2])
>>> y = col2im(x, output_size)
>>> print(y.shape)
(16, 16, 8, 8)
```

mindspore.ops.Concat

class mindspore.ops.**Concat** (*axis=0*)

```
prim = ops.Concat(axis)
out = prim(tensors)
```

等价于

```
ops.cat(tensors, axis)
```

更多详情请查看：[`mindspore.ops.cat\(\)`](#)。

mindspore.ops.Cummax

class mindspore.ops.**Cummax** (*axis*)

```
prim = ops.Cummax(axis)
out = prim(input)
```

等价于

```
ops.cummax(input, axis)
```

更多详情请查看：[`mindspore.ops.cummax\(\)`](#)。

mindspore.ops.Cummin

class mindspore.ops.Cummin(*axis*)

返回输入 Tensor 在指定轴上的累积最小值与其对应的索引。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.cummin()`。

参数：

- **axis** (int) - 算子操作的维度，维度的大小范围是 `[-input.ndim, input.ndim - 1]`。

输入：

- **input** (Tensor) - 输入 Tensor。

输出：

一个包含两个 Tensor 的元组 (values, indices)，分别表示累积最小值和对应索引。每个输出 Tensor 的 shape 和输入 Tensor 的 shape 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> import mindspore
>>> a = Tensor([-0.2284, -0.6628, 0.0975, 0.2680, -1.3298, -0.4220], mindspore.float32)
>>> axis = 0
>>> output = ops.Cummin(axis)(a)
>>> print(output[0])
[-0.2284 -0.6628 -0.6628 -0.6628 -1.3298 -1.3298]
>>> print(output[1])
[0 1 1 1 4 4]
```

mindspore.ops.CumProd

class mindspore.ops.CumProd(*exclusive=False, reverse=False*)

计算 x 沿着指定 axis 的元素累积积。

例如，如果输入是长度为 N 的 vector，则输出也是长度为 N 的 vector。其中的每一个元素为：

$$y_i = x_1 * x_2 * x_3 * \dots * x_i$$

参数：

- **exclusive** (bool, 可选) - 如果为 True，计算元素累积积时会排除末尾元素（见样例）。默认值：False。
- **reverse** (bool, 可选) - 如果为 True，计算元素累积积时会沿 axis 反转结果。默认值：False。

输入：

- **x** (Tensor[Number]) - 输入 Tensor。shape 为 $(N, *)$ ，其中 $*$ 为任意数量的额外维度。
- **axis** (int) - 沿此方向计算累积积。仅支持常量值。

输出：

Tensor, shape 和数据类型与 x 相同。

异常:

- **TypeError** - *exclusive* 或 *reverse* 不是 bool 类型。
- **TypeError** - *axis* 不是 int 类型。
- **ValueError** - *axis* 为 None。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> a, b, c, = 1, 2, 3
>>> x = Tensor(np.array([a, b, c]).astype(np.float32))
>>> op0 = ops.CumProd()
>>> output0 = op0(x, 0) # output=[a, a * b, a * b * c]
>>> op1 = ops.CumProd(exclusive=True)
>>> output1 = op1(x, 0) # output=[1, a, a * b]
>>> op2 = ops.CumProd(reverse=True)
>>> output2 = op2(x, 0) # output=[a * b * c, b * c, c]
>>> op3 = ops.CumProd(exclusive=True, reverse=True)
>>> output3 = op3(x, 0) # output=[b * c, c, 1]
>>> print(output0)
[1. 2. 6.]
>>> print(output1)
[1. 1. 2.]
>>> print(output2)
[6. 6. 3.]
>>> print(output3)
[6. 3. 1.]
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [5, 3, 5]]).astype(np.float32))
>>> output4 = op0(x, 0)
>>> output5 = op0(x, 1)
>>> print(output4)
[[ 1.  2.  3.]
 [ 4. 10. 18.]
 [20. 30. 90.]]
>>> print(output5)
[[ 1.  2.  6.]
 [ 4. 20. 120.]
 [ 5. 15. 75.]]
```

mindspore.ops.CumSum

class mindspore.ops.CumSum (*exclusive=False, reverse=False*)

计算输入 Tensor 在指定轴上的累加和。

$$y_i = x_1 + x_2 + x_3 + \dots + x_i$$

参数:

- **exclusive** (bool, 可选) - 表示输出结果的第一个元素是否与输入的第一个元素一致。如果为 False, 表示输出的第一个元素与输入的第二个元素一致。默认值: False。

- **reverse** (bool, 可选) - 如果为 True , 则逆向计算累加和。默认值: False 。

输入:

- **input** (Tensor) - 输入 Tensor, shape 为 $(N,*)$, 其中 * 为任意数量的额外维度。
- **axis** (int) - 指定要进行累加和计算的轴。仅支持常量值, 该值应在 $[-\text{rank}(\text{input}), \text{rank}(\text{input}))$ 范围内。

输出:

Tensor。输出 Tensor 的 shape 与输入 Tensor 的 shape 一致。

异常:

- **TypeError** - *exclusive* 或 *reverse* 不是 bool。
- **TypeError** - *axis* 不是 int。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[3, 4, 6, 10], [1, 6, 7, 9], [4, 3, 8, 7], [1, 3, 7, 9]]).astype(np.
    ↳ float32))
>>> cumsum = ops.CumSum()
>>> # case 1: along the axis 0
>>> y = cumsum(x, 0)
>>> print(y)
[[ 3.  4.  6. 10.]
 [ 4. 10. 13. 19.]
 [ 8. 13. 21. 26.]
 [ 9. 16. 28. 35.]]
>>> # case 2: along the axis 1
>>> y = cumsum(x, 1)
>>> print(y)
[[ 3.  7. 13. 23.]
 [ 1.  7. 14. 23.]
 [ 4.  7. 15. 22.]
 [ 1.  4. 11. 20.]]
>>> # Next demonstrate exclusive and reverse, along axis 1
>>> # case 3: exclusive = True
>>> cumsum = ops.CumSum(exclusive=True)
>>> y = cumsum(x, 1)
>>> print(y)
[[ 0.  3.  7. 13.]
 [ 0.  1.  7. 14.]
 [ 0.  4.  7. 15.]
 [ 0.  1.  4. 11.]]
>>> # case 4: reverse = True
>>> cumsum = ops.CumSum(reverse=True)
>>> y = cumsum(x, 1)
>>> print(y)
[[23. 20. 16. 10.]
 [23. 22. 16.  9.]
 [22. 18. 15.  7.]
 [20. 19. 16.  9.]]
```


mindspore.ops.DataFormatDimMap

class mindspore.ops.DataFormatDimMap (*src_format='NHWC', dst_format='NCHW'*)

返回源数据格式中的目标数据格式的维度索引。

参数：

- **src_format** (str) - 源数据格式中的可选值。格式可以是 'NHWC' 和 'NCHW'。默认值：'NHWC'。
- **dst_format** (str) - 目标数据格式中的可选值。格式可以是 'NHWC' 和 'NCHW'。默认值：'NCHW'。

输入：

- **input_x** (Tensor) - 输入 Tensor，每个元素都用作源数据格式的维度索引。建议值在 [-4, 4) 范围内，仅支持 int32。

输出：

输出 Tensor，返回给定目标数据格式的维度索引，与 *input_x* 具有相同的数据类型和 shape。

异常：

- **TypeError** - *src_format* 或 *dst_format* 不是 str。
- **TypeError** - *input_x* 不是数据类型为 int32 的 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input_x = Tensor([0, 1, 2, 3], mindspore.int32)
>>> dfdm = ops.DataFormatDimMap()
>>> output = dfdm(input_x)
>>> print(output)
[0 3 1 2]
```

mindspore.ops.DepthToSpace

class mindspore.ops.DepthToSpace (*block_size*)

将深度数据重新排列到空间维度中。

这是 SpaceToDepth 的反向操作。

输出 Tensor 的深度为 $input\_depth / (block\_size * block\_size)$ 。

输出 Tensor 的 *height* 维度为 $height * block\_size$ 。

输出 Tensor 的 *weight* 维度为 $weight * block\_size$ 。

输入 Tensor 的深度必须可被 $block\_size * block\_size$ 整除。数据格式为“NCHW”。

参数：

- **block_size** (int) - 用于划分深度数据的块大小。其值必须 ≥ 2 。

输入：

- **x** (Tensor) - 输入 Tensor。它必须为四维，其 shape 为 $(N, C_{in}, H_{in}, W_{in})$ ，数据类型为数值型。

输出：

Tensor, shape 为 $(N, C_{in}/block\_size^2, H_{in} * block\_size, W_{in} * block\_size)$ 。

异常:

- **TypeError** - 如果 *block_size* 不是 int。
- **ValueError** - 如果 *block_size* 小于 2。
- **ValueError** - 如果 *x* 的 shape 长度不等于 4。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.rand(1, 12, 1, 1), mindspore.float32)
>>> block_size = 2
>>> depth_to_space = ops.DepthToSpace(block_size)
>>> output = depth_to_space(x)
>>> print(output.shape)
(1, 3, 2, 2)
```

mindspore.ops.Diag

class mindspore.ops.Diag

```
prim = ops.Diag()
out = prim(input)
```

等价于

```
ops.diag(input)
```

更多详情请查看: [mindspore.ops.diag\(\)](#)。

mindspore.ops.DType

class mindspore.ops.DType

计算输入 Tensor 的数据类型, 且返回的数据类型为 *mindspore.dtype*。

输入:

- **input_x** (Tensor) - 输入 Tensor。

输出:

mindspore.dtype, 输入 Tensor 的数据类型。

异常:

- **TypeError** - 如果 *input_x* 不是 Tensor。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_tensor = Tensor(np.array([[2, 2], [2, 2]]), mindspore.float32)
>>> output = ops.DType()(input_tensor)
>>> print(output)
Float32
```

mindspore.ops.ExpandDims

class mindspore.ops.ExpandDims

```
prim = ops.ExpandDims()
out = prim(input_x, axis)
```

等价于

```
ops.expand_dims(input_x, axis)
```

更多详情请查看：[mindspore.ops.expand_dims\(\)](#)。

mindspore.ops.FillDiagonal

class mindspore.ops.FillDiagonal (*fill_value, wrap=False*)

填充至少具有二维的 Tensor 的主对角线。当输入的维度大于 2 时，所有维度必须等长。此函数原地修改输入 Tensor，并返回该 Tensor。

警告： 这是一个实验性 API，后续可能修改或删除。

参数：

- **fill_value** (float) - *input_x* 对角线的填充值。
- **wrap** (bool, 可选) - 对于一个高矩阵（即矩阵的行数大于列数），控制对角线元素是否继续延伸到剩余的行。具体效果详见下方代码样例。默认值：False。

输入：

- **input_x** (Tensor) - shape 为 $(x_1, x_2, \dots, x_R)$ 。

输出：

- **y** (Tensor) - Tensor，和输入 *input_x* 具有相同的 shape 和 dtype。

异常：

- **ValueError** - 如果 *input_x* 的维度没有大于 1。
- **ValueError** - 当维度大于 2 时，每个轴的大小不相等。

支持平台：

Ascend GPU CPU

样例：

```

>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]).astype(np.float32))
>>> fill_value = 9.9
>>> fill_diagonal = ops.FillDiagonal(fill_value)
>>> y = fill_diagonal(x)
>>> print(y)
[[9.9 2.  3. ]
 [4.  9.9 6. ]
 [7.  8.  9.9]]
>>> x = Tensor(np.array([[0, 0, 0], [1, 1, 1], [2, 2, 2], [3, 3, 3], [4, 4, 4], [5, 5, 5]]).
→.astype(np.int32))
>>> fill_value = 9.0
>>> fill_diagonal = ops.FillDiagonal(fill_value)
>>> y = fill_diagonal(x)
>>> print(y)
[[9 0 0]
 [1 9 1]
 [2 2 9]
 [3 3 3]
 [4 4 4]
 [5 5 5]]
>>> x = Tensor(np.array([[0, 0, 0], [1, 1, 1], [2, 2, 2], [3, 3, 3],
...                       [4, 4, 4], [5, 5, 5], [6, 6, 6]]).astype(np.int64))
>>> fill_value = 9.0
>>> wrap = True
>>> fill_diagonal = ops.FillDiagonal(fill_value, wrap)
>>> y = fill_diagonal(x)
>>> print(y)
[[9 0 0]
 [1 9 1]
 [2 2 9]
 [3 3 3]
 [9 4 4]
 [5 9 5]
 [6 6 9]]

```

mindspore.ops.FillV2

class mindspore.ops.FillV2

创建一个 Tensor，其 shape 由 *shape* 指定，其值则由 *value* 进行填充。

输入：

- **shape** (Union[Tuple[int], Tensor[int]]) - 1-D Tensor 或 Tuple，指定了输出 Tensor 的 shape。其数据类型必须是 int32 或 int64。
- **value** (Tensor) - *value* 是零维 Tensor，其值用于填充输出 *y*。

输出：

- **y** (Tensor) - Tensor，其 shape 和值如上所述。

异常：

- **TypeError** - 如果 *shape* 不是 1-D Tensor 或 Tuple。
- **TypeError** - 如果 *shape* 的数据类型不是 int32 或者 int64。

- **ValueError** - 如果 *value* 不是零维 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> fillv2 = ops.Fillv2()
>>> output = fillv2(Tensor([2, 3], mindspore.int32), Tensor(1, mindspore.float32))
>>> print(output)
[[1. 1. 1.]
 [1. 1. 1.]]
>>> output = fillv2(Tensor([3, 3], mindspore.int64), Tensor(0, mindspore.int32))
>>> print(output)
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

mindspore.ops.FloatStatus

class mindspore.ops.FloatStatus

确定元素是否包含非数字 (NaN)、正无穷或负无穷。0 表示正常，1 表示溢出。

输入：

- **x** (Tensor) - 输入 Tensor。数据类型必须为 float16、float32 或 float64。 $(N, *)$ ，其中 $*$ 表示任意数量的附加维度。

输出：

Tensor, shape 为 (1,)，数据类型为 *mindspore.dtype.float32*。

异常：

- **TypeError** - 如果 *x* 的数据类型不是 float16，float32 或 float64。

支持平台：

GPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> float_status = ops.FloatStatus()
>>> x = Tensor(np.array([np.log(-1), 1, np.log(0)]), mindspore.float32)
>>> result = float_status(x)
>>> print(result)
[1.]
```

mindspore.ops.Fmax

class mindspore.ops.Fmax

逐元素计算输入 Tensor 的最大值。

警告：这是一个实验性 API，后续可能修改或删除。

详情请查看[`mindspore.ops.fmax\(\)`](#)。

支持平台：

CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x1 = Tensor(np.array([1.0, 5.0, 3.0]), mindspore.float32)
>>> x2 = Tensor(np.array([4.0, 2.0, 6.0]), mindspore.float32)
>>> fmax = ops.Fmax()
>>> output = fmax(x1, x2)
>>> print(output)
[4. 5. 6.]
```

mindspore.ops.Gather

class mindspore.ops.Gather (*batch_dims=0*)

```
prim = ops.Gather(batch_dims)
out = prim(input_params, input_indices, axis)
```

等价于

```
ops.gather(input_params, input_indices, axis, batch_dims)
```

更多详情请查看：[`mindspore.ops.gather\(\)`](#)。

mindspore.ops.GatherD

class mindspore.ops.GatherD

```
prim = ops.GatherD()
out = prim(x, dim, index)
```

等价于

```
ops.gather_d(x, dim, index)
```

更多详情请查看：[`mindspore.ops.gather\_d\(\)`](#)。

mindspore.ops.GatherNd

class mindspore.ops.GatherNd

```
prim = ops.GatherNd()
out = prim(input_x, indices)
```

等价于

```
ops.gather_nd(input_x, indices)
```

更多详情请查看：[mindspore.ops.gather_nd\(\)](#)。

mindspore.ops.HammingWindow

class mindspore.ops.HammingWindow (*periodic=True, alpha=0.54, beta=0.46, dtype=mstype.float32*)

使用输入窗口长度计算汉明窗口函数。

$$w[n] = \alpha - \beta \cos\left(\frac{2\pi n}{N-1}\right),$$

其中， N 是全窗口尺寸。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **periodic** (bool, 可选) - 一个标志，表示返回的窗口是否修剪掉来自对称窗口的最后一个重复值。默认值：True。
 - 如果为 True，则返回的窗口作为周期函数，在上式中， $N = \text{length} + 1$ 。
 - 如果为 False，则返回一个对称窗口， $N = \text{length}$ 。
- **alpha** (float, 可选) - 加权系数，上式中的 α ，默认值：0.54。
- **beta** (float, 可选) - 加权系数，上式中的 β ，默认值：0.46。
- **dtype** ([mindspore.dtype](#), 可选) - 数据类型，可选值为 `mstype.float16`、`mstype.float32` 或 `mstype.float64`。默认值：`mstype.float32`。

输入：

- **length** (Tensor) - 一个 1D 的正整数 Tensor，控制返回窗口的大小。

输出：

Tensor，一个包含窗口的 1-D Tensor，其 shape 为 (length,)。

异常：

- **TypeError** - 如果 *length* 不是一个 Tensor。
- **TypeError** - 如果 *length* 的数据类型不是整型。
- **TypeError** - 如果 *periodic* 的数据类型不是 bool 类型。
- **TypeError** - 如果 *alpha* 的数据类型不是 float 类型。
- **TypeError** - 如果 *beta* 的数据类型不是 float 类型。

- **TypeError** - 如果 *dtype* 的取值不是 *mstype.float16*、*mstype.float32* 或 *mstype.float64*。
- **ValueError** - 如果 *length* 的维度不是 1。
- **ValueError** - 如果 *length* 的值是负数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> # case 1: periodic=True.
>>> length = Tensor(np.array([6]).astype(np.int32))
>>> hamming_window = ops.HammingWindow(periodic=True)
>>> y = hamming_window(length)
>>> print(y)
[0.08000001 0.31          0.77000004 1.          0.77000004 0.31          ]
>>> # case 2: periodic=False.
>>> length = Tensor(np.array([7]).astype(np.int32))
>>> hamming_window = ops.HammingWindow(periodic=False)
>>> y = hamming_window(length)
>>> print(y)
[0.08000001 0.31          0.77000004 1.          0.77000004 0.31          0.08000001]
```

mindspore.ops.Heaviside

class mindspore.ops.Heaviside

计算输入中每个元素的 Heaviside 步长函数。

$$\text{heaviside}(x, \text{values}) = \begin{cases} 0, & \text{if } x < 0 \\ \text{values}, & \text{if } x == 0 \\ 1, & \text{if } x > 0 \end{cases}$$

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 输入 Tensor，实数类型。
- **values** (Tensor) - 在 *x* 中为 0 的位置应用其值。*values* 可以同 *x* 进行广播。*x* 与 *values* 的数据类型应该相同。

输出：

Tensor，与 *x* 和 *values* 的数据类型相同。

异常：

- **TypeError** - 如果 *x* 或 *values* 不是 Tensor。
- **TypeError** - 如果 *x* 与 *values* 的数据类型不一致。
- **ValueError** - 如果两个输入的 *shape* 之间无法进行广播。

支持平台：

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([-1.5, 0., 2.]))
>>> values = Tensor(np.array([0.5]))
>>> heaviside = ops.Heaviside()
>>> y = heaviside(x, values)
>>> print(y)
[0.  0.5 1. ]
```

mindspore.ops.HistogramFixedWidth

class mindspore.ops.HistogramFixedWidth (nbins, dtype='int32')

返回一个 rank 为 1 的直方图，该直方图中的每个组的值表示数量。每个组的宽度相等，由输入 *range* 和参数 *nbins* 决定。

参数:

- **nbins** (int) - 直方图的组数，类型为正整数。
- **dtype** (str, 可选) - 可选属性。数据类型必须为 str。默认值: 'int32'。

输入:

- **x** (Tensor) - HistogramFixedWidth 的输入，为一个 Tensor。数据类型必须为 int32、float32、float16 或 float64。
- **range** (Tensor) - 数据类型与 *x* 相同，shape 为 (2,)。 $x \leq \text{range}[0]$ 将映射到 `histogram[0]`， $x \geq \text{range}[1]$ 将映射到 `histogram[-1]`。

输出:

1-D Tensor，其长度为 *nbins*，数据类型为 int32。

异常:

- **TypeError** - 如果 *dtype* 不是 str 或 *nbins* 不是 int。
- **ValueError** - 如果 *nbins* 小于 1。
- **ValueError** - 如果 *dtype* 不是 "int32"。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> x = Tensor([-1.0, 0.0, 1.5, 2.0, 5.0, 15], mindspore.float16)
>>> range_op = Tensor([0.0, 5.0], mindspore.float16)
>>> hist = ops.HistogramFixedWidth(5)
>>> output = hist(x, range_op)
>>> print(output)
[2 1 1 0 2]
```

mindspore.ops.Hypot

class mindspore.ops.Hypot

将输入 Tensor 的逐个元素作为直角三角形的直角边，并计算其斜边的值。两个输入的 shape 应该是可广播的，且它们的数据类型应该是其中之一：float32、float64。

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x1** (Tensor) - 第一个输入 Tensor。
- **x2** (Tensor) - 第二个输入 Tensor。

输出：

Tensor，shape 与广播后的 shape 相同，数据类型为两个输入中具有更高的精度的那一个。

异常：

- **TypeError** - 如果 *x1* 或 *x2* 的数据类型不是 float32 或 float64。
- **ValueError** - 如果两个输入的 shape 无法广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x1 = Tensor(np.array([3., 5., 7.]))
>>> x2 = Tensor(np.array([4., 12., 24.]))
>>> hypot_ = ops.Hypot()
>>> y = hypot_(x1, x2)
>>> print(y)
[ 5. 13. 25.]
>>> x1 = Tensor(2.1, mindspore.float32)
>>> x2 = Tensor(2.1, mindspore.float32)
>>> hypot_ = ops.Hypot()
>>> y = hypot_(x1, x2)
>>> print(y)
2.9698484
```

mindspore.ops.Identity

class mindspore.ops.Identity

```
prim = ops.Identity()
out = prim(input_x)
```

等价于

```
ops.deepcopy(input_x)
```

更多详情请查看：[mindspore.ops.deepcopy\(\)](#)。

mindspore.ops.lgamma

class mindspore.ops.Igamma

计算正则化的下层不完全伽马函数。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见[`mindspore.ops.lgamma\(\)`](#)。

输入：

- **a** (Tensor) - 输入 Tensor。
- **x** (Tensor) - 输入 Tensor，数据类型应该与 *a* 一致。

输出：

Tensor，数据类型与 *a* 和 *x* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> a = Tensor(np.array([2.0, 4.0, 6.0, 8.0]).astype(np.float32))
>>> x = Tensor(np.array([2.0, 3.0, 4.0, 5.0]).astype(np.float32))
>>> lgamma = ops.Igamma()
>>> output = lgamma(a, x)
>>> print (output)
[0.593994  0.35276785  0.21486944  0.13337152]
>>> a = Tensor(2.1, mindspore.float32)
>>> x = Tensor(2.1, mindspore.float32)
>>> lgamma = ops.Igamma()
>>> output = lgamma(a, x)
>>> print (output)
0.5917439
```

mindspore.ops.lgammac

class mindspore.ops.Igammac

计算正则化的上层不完全伽马函数。

更多参考详见[`mindspore.ops.lgammac\(\)`](#)。

输入：

- **a** (Tensor) - 输入 Tensor。
- **x** (Tensor) - 输入 Tensor，数据类型应该与 *a* 一致。

输出：

Tensor，数据类型与 *a* 和 *x* 相同。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> a = Tensor(np.array([2.0, 4.0, 6.0, 8.0]).astype(np.float32))
>>> x = Tensor(np.array([2.0, 3.0, 4.0, 5.0]).astype(np.float32))
>>> igammac = ops.Igammac()
>>> output = igammac(a, x)
>>> print (output)
[0.40600586 0.6472318 0.7851304 0.8666283 ]
>>> a = Tensor(2.1, mindspore.float32)
>>> x = Tensor(2.1, mindspore.float32)
>>> igammac = ops.Igammac()
>>> output = igammac(a, x)
>>> print (output)
0.40825662
```

mindspore.ops.Im2Col

class mindspore.ops.Im2Col (*ksizes, strides=1, dilations=1, pads=0*)

从一个 batch 的输入 Tensor 中提取滑动局部块。

考虑一个 batch 的输入 Tensor, 其 shape 为 $(N, C, *)$, 其中 N 是 batch 维度, C 是 channel 维度, $*$ 表示任意的空间维度。此操作将展平输入 x 空间维度内每个滑动的 $ksize$ 大小的滑块为输出的 4-D Tensor 中的一列 (如, 最后一维), 输出 Tensor 的 shape 为 $(N, C, \prod(\text{kernel\_size}), L)$, 其中 $C \times \prod(\text{kernel\_size})$ 表示每个滑块内的总元素数量 (一个滑块有 $\prod(\text{kernel\_size})$ 个空间位置, 每个位置都包含一个 C 通道的向量), 共有 L 个这样的滑块:

$$L = \prod_d \left\lfloor \frac{\text{spatial\_size}[d] + 2 \times \text{pads}[d] - \text{dilations}[d] \times (\text{kernel\_size}[d] - 1) - 1}{\text{strides}[d]} + 1 \right\rfloor,$$

其中, spatial_size 由输入 x 的空间维度 (上面的 $*$) 决定, d 遍历所有的空间维度。

因此, 在最后一个维度 (列维度) 上 $output$ 包含特定块内的所有值。

pads , strides 和 dilations 决定了滑块如何被取出。

说明: 目前, 只支持 4-D Tensor (一个 batch 的图像 Tensor)。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **ksizes** (Union[int, tuple[int], list[int]]) - 内核的大小。若为两个整数, 则分别代表高度和宽度; 若只有一个整数, 则同时代表高度和宽度。该参数必须被指定。
- **strides** (Union[int, tuple[int], list[int]], 可选) - 窗口的滑动步幅。若为两个整数, 则分别代表高度和宽度; 若只有一个整数, 则同时代表高度和宽度。默认值: 1。
- **dilations** (Union[int, tuple[int], list[int]], 可选) - 窗口的扩张系数。若为两个整数, 则分别代表高度和宽度; 若只有一个整数, 则同时代表高度和宽度。默认值: 1。
- **pads** (Union[int, tuple[int], list[int]], 可选) - 窗口的填充。必须是一个或两个整数来指定高宽和宽度方向的填充。默认值: 0。

- 如果是 1 个整数, 则 $pad\_height = pad\_width$ 。
- 如果是 2 个整数, 则 $pad\_height = pads[0]$, $pad\_width = pads[1]$ 。

输入:

- **x** (Tensor) - 输入 Tensor, 只支持 4-D Tensor(1 个 batch 的图像 Tensor)。

输出:

Tensor, 一个 4-D Tensor, 与输入 x 的数据类型相同。

异常:

- **TypeError** - 如果 $ksize$ s 的类型不在 Union[int, tuple[int], list[int]] 内。
- **TypeError** - 如果 $strides$ 的类型不在 Union[int, tuple[int], list[int]] 内。
- **TypeError** - 如果 $dilations$ 的类型不在 Union[int, tuple[int], list[int]] 内。
- **TypeError** - 如果 $pads$ 类型不在 Union[int, tuple[int], list[int]] 内。
- **ValueError** - 如果 $ksize$ s 的值不大于 0 或其元素数量大于 2。
- **ValueError** - 如果 $strides$ 的值不大于 0 或其元素数量大于 2。
- **ValueError** - 如果 $dilations$ 的值不大于 0 或其元素数量大于 2。
- **ValueError** - 如果 $pads$ 的值不大于 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(input_data=np.random.rand(4, 4, 32, 32), dtype=mstype.float64)
>>> im2col = ops.Im2Col(ksize=3, strides=1, dilations=1)
>>> y = im2col(x)
>>> print(y.shape)
(4, 4, 9, 900)
```

mindspore.ops.IndexAdd

class mindspore.ops.IndexAdd(*axis*, *use_lock=True*, *check_index_bound=True*)

将 Tensor y 加到 Tensor x 的指定 $axis$ 和 $indices$ 。 $axis$ 取值范围为 $[-len(x.dim), len(x.dim) - 1]$, $indices$ 取值范围为 $[0, len(x[axis]) - 1]$ 。

参数:

- **axis** (int) - 进行索引的 $axis$ 。
- **use_lock** (bool, 可选) - 是否对参数更新加锁保护。默认值: True。
- **check_index_bound** (bool, 可选) - 如果为 True 将对索引进行边界检查。默认值: True。

输入:

- **x** (Union[Parameter, Tensor]) - 加法计算的输入参数或 Tensor。
- **indices** (Tensor) - 沿 $axis$ 在指定 $indices$ 位置进行加法运算。数据类型支持 int32。 $indices$ 必须为一维且与 y 在 $axis$ 维度的尺寸相同。 $indices$ 取值范围应为 $[0, b)$, 其中 b 为 x 在 $axis$ 维度的尺寸。

- **y** (Tensor) - 被添加到 x 的输入 Tensor。必须与 x 的数据类型相同。除 $axis$ 之外的维度 shape 必须与 x 的 shape 相同。

输出：

Tensor, 与 x 的 shape 和数据类型相同。

异常：

- **TypeError** - $indices$ 或 y 不是 Tensor。
- **ValueError** - $axis$ 超过了 x 的秩。
- **ValueError** - x 与 y 的秩不相同。
- **ValueError** - $indices$ 不是一维或与 $y[axis]$ 的尺寸不同。
- **ValueError** - y 的 shape 与除 $axis$ 之外的维度的 x 的 shape 不同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, nn, ops, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.index_add = ops.IndexAdd(axis=1)
...         self.x = Parameter(Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.
→float32),
...                             name="name_x")
...         self.indices = Tensor(np.array([0, 2]), mindspore.int32)
...
...     def construct(self, y):
...         return self.index_add(self.x, self.indices, y)
...
>>> y = Tensor(np.array([[0.5, 1.0], [1.0, 1.5], [2.0, 2.5]]), mindspore.float32)
>>> net = Net()
>>> output = net(y)
>>> print(output)
[[ 1.5  2.   4. ]
 [ 5.   5.   7.5]
 [ 9.   8.  11.5]]
```

mindspore.ops.IndexFill

class mindspore.ops.IndexFill

按 $index$ 中给定的顺序选择索引，将输入 Tensor x 的 dim 维度下的元素用 $value$ 的值填充。

警告： 这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.index_fill()`。

输入：

- **x** (Tensor) - 输入 Tensor。

- **dim** (Union[int, Tensor]) - 填充输入 Tensor 的维度, 要求是一个 int 或者数据类型为 int32 或 int64 的零维 Tensor。
- **index** (Tensor) - 填充输入 Tensor 的索引, 数据类型为 int32。
- **value** (Union[bool, int, float, Tensor]) - 填充输入 Tensor 的值。

输出:

填充后的 Tensor。shape 和数据类型与输入 x 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> index_fill = ops.IndexFill()
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]).astype(np.float32))
>>> index = Tensor([0, 2], mindspore.int32)
>>> value = Tensor(-2.0, mindspore.float32)
>>> y = index_fill(x, 1, index, value)
>>> print(y)
[[-2.  2. -2.]
 [-2.  5. -2.]
 [-2.  8. -2.]]
```

mindspore.ops.IndexPut

class mindspore.ops.IndexPut (accumulate=0)

根据 *indices* 中的下标值, 使用 x_2 中的数值替换 x_1 中的相应元素的值。

参数:

- **accumulate** (int) - 如果 *accumulate* 被设置为 1, x_2 中的元素被累加到 x_1 的相应元素上; 如果为 0, 用 x_2 中的元素取代 x_1 的相应元素。默认值: 0。

输入:

- **x_1** (Tensor) - 被执行替换操作的 Tensor, 维度大于等于 1。
- **x_2** (Tensor) - 数据类型和 x_1 一致的一维的 Tensor。如果其 size 为 1, 则 shape 将被广播为 x_1 的 shape。
- **indices** (tuple[Tensor], list[Tensor]) - 元素类型是 int32 或者 int64, 用于对 x_1 中的元素进行索引。*indices* 中的 tensor 的秩应为 1-D, *indices* 中 tensor 的 size 应小于 x_1 的秩, *indices* 中的 tensor 应是可广播的。

输出:

Tensor, 其数据类型和 shape 与 x_1 相同。

异常:

- **TypeError** - 如果 x_1 的 dtype 与 x_2 的 dtype 不同。
- **TypeError** - 如果 *indices* 不是 tuple[Tensor] 或者 list[Tensor]。
- **TypeError** - 如果 *indices* 中的 tensor 的 dtype 不是 int32 或者 int64。
- **TypeError** - 如果 *indices* 中的 tensor 的 dtype 是不一致的。
- **TypeError** - 如果 *accumulate* 的 dtype 不是 int。

- **ValueError** - 如果 x_2 的秩不是 1-D。
- **ValueError** - 当 x_1 的 rank 与 $indices$ 的 size 相等时, 如果 x_2 的 size 不为 1 或者不为 $indices$ 中 Tensor 的最大 size。
- **ValueError** - 当 x_1 的 rank 大于 $indices$ 的 size 时, 如果 x_2 的 size 不为 1 或者不为 x_1 的最后一维的 shape。
- **ValueError** - 如果 $indices$ 中的 tensor 的秩不是 1-D。
- **ValueError** - 如果 $indices$ 中的 tensor 不是可广播的。
- **ValueError** - 如果 $indices$ 的 size 大于 x_1 的秩。
- **ValueError** - 如果 $accumulate$ 的值不是 0 或 1。

支持平台:

Ascend CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x1 = Tensor(np.array([[1, 2, 3], [4, 5, 6]]).astype(np.int32))
>>> x2 = Tensor(np.array([3]).astype(np.int32))
>>> indices = [Tensor(np.array([0, 0]).astype(np.int32)), Tensor(np.array([0, 1]).astype(np.
    ↪int32))]
>>> accumulate = 1
>>> op = ops.IndexPut(accumulate = accumulate)
>>> output = op(x1, x2, indices)
>>> print(output)
[[4 5 3]
 [4 5 6]]
```

mindspore.ops.InplaceAdd

class mindspore.ops.InplaceAdd (*indices*)

将 $input\_v$ 添加到 x 的特定行。即计算 $y = x; y[i,] += input\_v$ 。

更多参考详见 `mindspore.ops.inplace_add()`。

参数:

- **indices** (Union[int, tuple]) - 指定将 $input\_v$ 添加到 x 的哪些行。可以为 int 或 Tuple, 取值范围 $[0, \text{len}(x)]$ 。

输入:

- **x** (Tensor) - 输入 Tensor, shape 为: $(N, *)$, 其中 $*$ 表示任何数量的附加维度。
- **input_v** (Tensor) - 添加到 x 的 Tensor。除第一个维度之外 shape 必须与 x 的 shape 相同。第一个维度必须与 $indices$ 的长度相同。数据类型与 x 相同。

输出:

Tensor, 与 x 的 shape 和数据类型相同。

支持平台:

Ascend GPU CPU

样例:


```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> indices = (0, 1)
>>> x = Tensor(np.array([[1, 2], [3, 4], [5, 6]]), mindspore.float32)
>>> input_v = Tensor(np.array([[0.5, 1.0], [1.0, 1.5]]), mindspore.float32)
>>> inplaceAdd = ops.InplaceAdd(indices)
>>> output = inplaceAdd(x, input_v)
>>> print(output)
[[1.5 3. ]
 [4.  5.5]
 [5.  6. ]]
```

mindspore.ops.InplaceIndexAdd

class mindspore.ops.InplaceIndexAdd(*axis*)

逐元素将一个 Tensor *updates* 添加到原 Tensor *var* 的指定轴和索引处。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.inplace_index_add()`。

参数：

- **axis** (int) - 要执行添加操作的轴。应该在范围 `[0, len(var.dim))` 内。

输入：

- **var** (Union[Parameter, Tensor]) - 被添加的输入 Parameter 或 Tensor，数据类型为 uint8、int8、int16、int32、float16、float32 或 float64。
- **indices** (Tensor) - *axis* 指定轴上执行添加操作的索引。是一个 1D Tensor，shape 为 `(updates.shape[axis],)`，它的每个值应在范围 `[0, var.shape[axis]]` 之内，数据类型为 int32。
- **updates** (Tensor) - 要添加的输入 Tensor。必须与 *var* 具有相同的数据类型。除 *axis* 维度外，*updates* 与 *var* 的 shape 应一致。

输出：

Tensor，更新后的结果，其 shape 和 dtype 与 *var* 一致。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> var = Parameter(Tensor(np.array([[1, 2], [3, 4], [5, 6]]), mindspore.float32))
>>> indices = Tensor(np.array([0, 1]), mindspore.int32)
>>> updates = Tensor(np.array([[0.5, 1.0], [1.0, 1.5]]), mindspore.float32)
>>> inplaceIndexAdd = ops.InplaceIndexAdd(axis=0)
>>> var = inplaceIndexAdd(var, indices, updates)
>>> print(var)
[[1.5 3. ]]
```

(下页继续)

(续上页)

```
[4.  5.5]
[5.  6.  ]]
```

mindspore.ops.InplaceSub

class mindspore.ops.InplaceSub(*indices*)

从 x 的特定行减去 $input\_v$ 。即计算 $y = x; y[i, :] = input\_v$ 。

更多参考详见 `mindspore.ops.inplace_sub()`。

参数：

- **indices** (Union[int, tuple]) - 指定将 x 的哪些行减去 $input\_v$ 。可以为 int 或 Tuple，取值范围 $[0, \text{len}(x)]$ 。

输入：

- **x** (Tensor) - 输入 Tensor，shape 为：(N, *)，其中 * 表示任何数量的附加维度。
- **input_v** (Tensor) - 从 x 减去的 Tensor。除第一个维度之外 shape 必须与 x 的 shape 相同。第一个维度必须与 *indices* 的长度相同。数据类型与 x 相同。

输出：

Tensor，与 x 的 shape 和数据类型相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> indices = (0, 1)
>>> x = Tensor(np.array([[1, 2], [3, 4], [5, 6]]), mindspore.float32)
>>> input_v = Tensor(np.array([[0.5, 1.0], [1.0, 1.5]]), mindspore.float32)
>>> inplaceSub = ops.InplaceSub(indices)
>>> output = inplaceSub(x, input_v)
>>> print(output)
[[0.5 1. ]
 [2.  2.5]
 [5.  6.  ]]
```

mindspore.ops.InplaceUpdate

class mindspore.ops.InplaceUpdate(*indices*)

InplaceUpdate 接口已废弃，请使用接口 `mindspore.ops.InplaceUpdateV2` 替换，废弃版本 2.0。

支持平台：

Deprecated

`mindspore.ops.InplaceUpdateV2`

class `mindspore.ops.InplaceUpdateV2`

根据 *indices*, 将 *x* 中的某些值更新为 *v*。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.inplace_update()`。

输入：

- **x** (Tensor) - 待更新的 Tensor。数据类型支持 float16、float32 或 int32。
- **indices** (Union[int, tuple, Tensor]) - 指定将 *x* 的哪些行更新为 *v*。可以为 int 或 Tuple 或 Tensor，取值范围 $[0, \text{len}(x))$ 。
- **v** (Tensor) - 除第一个维度之外，*shape* 必须与 *x* 的 *shape* 相同，第一个维度必须与 *indices* 的长度相同。数据类型与 *x* 相同。

输出：

Tensor，与 *x* 的 *shape* 和数据类型相同。

支持平台：

GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> indices = (0, 1)
>>> x = Tensor(np.array([[1, 2], [3, 4], [5, 6]]), mindspore.float32)
>>> v = Tensor(np.array([[0.5, 1.0], [1.0, 1.5]]), mindspore.float32)
>>> inplace_update_v2 = ops.InplaceUpdateV2()
>>> output = inplace_update_v2(x, indices, v)
>>> print(output)
[[0.5 1. ]
 [1.  1.5]
 [5.  6. ]]
```

`mindspore.ops.InvertPermutation`

class `mindspore.ops.InvertPermutation`

计算索引的逆置换。

该算子主要用于计算索引的逆置换。*input_x* 是一个一维的整数 Tensor，一个以 0 开始的索引数组，并将每个值与其索引位置交换。换句话说，对于输出 Tensor 和输入 *input_x*，依赖此计算方法 $y[x[i]] = i$ ， $i \in [0, 1, \dots, \text{len}(x) - 1]$ 。

说明：这些值必须包括 0。不能有重复的值，并且不能为负值。

输入：

- **input_x** (Union(tuple[int], list[int])) - 输入由多个整数构造，即 $(y_1, y_2, \dots, y_S)$ 代表索引。值必须包括 0。不能有重复值或负值。只允许常量。最大值必须等于 *input_x* 的长度。

输出：

`tuple[int]`。输出的长度与 `input_x` 相同。

异常：

- **TypeError** - 如果 `input_x` 既不是 `tuple` 也不是 `list`。
- **TypeError** - 如果 `input_x` 的元素不是 `int`。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import ops
>>> invert = ops.InvertPermutation()
>>> input_data = (3, 4, 0, 2, 1)
>>> output = invert(input_data)
>>> print(output)
(2, 4, 3, 0, 1)
```

`mindspore.ops.IsClose`

class `mindspore.ops.IsClose` (`rtol=1e-05`, `atol=1e-08`, `equal_nan=False`)

返回一个 `bool` 型 `Tensor`，表示 `input` 的每个元素与 `other` 的每个元素在给定容忍度内是否“接近”。其中“接近”的数学公式为：

$$|input - other|atol + rtol * |other|$$

更多参考详见 `mindspore.ops.isclose()`。

参数：

- **rtol** (`float`, 可选) - 相对容忍度。默认值：1e-05。
- **atol** (`float`, 可选) - 绝对容忍度。默认值：1e-08。
- **equal_nan** (`bool`, 可选) - 若为 `True`，则两个 `NaN` 被视为相同。默认值：`False`。

输入：

- **input** (`Tensor`) - 对比的第一个输入。
- **other** (`Tensor`) - 对比的第二个输入。

输出：

`Tensor`，`shape` 与 `input` 和 `other` 广播后的 `shape` 相同，数据类型是 `bool`。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor
>>> from mindspore.ops import IsClose
>>> input = Tensor(np.array([1.3, 2.1, 3.2, 4.1, 5.1]), mindspore.float16)
```

(下页继续)

(续上页)

```
>>> other = Tensor(np.array([1.3, 3.3, 2.3, 3.1, 5.1]), mindspore.float16)
>>> isclose = IsClose()
>>> output = isclose(input, other)
>>> print(output)
[ True False False False  True]
```

mindspore.ops.Lcm

class mindspore.ops.Lcm

逐个元素计算输入 Tensor 的最小公倍数。两个输入的 shape 应该是可广播的，它们的数据类型应该是其中之一：int32、int64。

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x1** (Tensor) - 第一个输入 Tensor。
- **x2** (Tensor) - 第二个输入 Tensor。

输出：

Tensor，shape 与广播后的 shape 相同，数据类型为两个输入中具有更高的精度的那一个。

异常：

- **TypeError** - 如果 *x1* 或 *x2* 的数据类型不是 int32 或 int64。
- **ValueError** - 如果两个输入的 shape 无法广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x1 = Tensor(np.array([7, 8, 9]))
>>> x2 = Tensor(np.array([14, 6, 12]))
>>> lcm_ = ops.Lcm()
>>> y = lcm_(x1, x2)
>>> print(y)
[14 24 36]
```

mindspore.ops.LeftShift

class mindspore.ops.LeftShift

将 Tensor 每个位置的值向左移动若干个比特位。输入是两个 Tensor，它们的数据类型必须一致，并且它们的 shape 可以广播。输出不支持隐式类型转换。

$$out_i = x_i \ll y_i$$

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **x1** (Tensor) - 目标 Tensor，将根据 **x2** 对应位置的值向左移动相应的比特位，类型支持所有 int 和 uint 类型。
- **x2** (Tensor) - Tensor 必须具有与 **x1** 相同的数据类型，且其 shape 必须与 **x1** 相同或者可以与 **x1** 进行广播。

输出：

- **output** (Tensor) - 输出 Tensor，数据类型与 **x1** 相同。并且输出 Tensor 的 shape 与 **x1** 相同，或者和 **x1** 和 **x2** 广播后的 shape 相同。

异常：

- **TypeError** - 如果 **x1** 或 **x2** 的数据类型错误。
- **TypeError** - 如果 **x1** 或 **x2** 不是 Tensor。
- **ValueError** - 如果 **x1** 与 **x2** 无法广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> left_shift = ops.LeftShift()
>>> x1 = Tensor(np.array([1, 2, 3]).astype(np.int8))
>>> x2 = Tensor(np.array([0, 1, -1]).astype(np.int8))
>>> output = left_shift(x1, x2)
>>> print(output)
[1 4 0]
```

mindspore.ops.LogSpace

class mindspore.ops.LogSpace (steps=10, base=10, dtype=mstype.float32)

返回一个大小为 **steps** 的 1-D Tensor，其值从 $base^{start}$ 到 $base^{end}$ ，以 **base** 为底数。

$$step = (end - start) / (steps - 1)$$

$$output = [base^{start}, base^{start+1*step}, ..., base^{start+(steps-2)*step}, base^{end}]$$

警告：这是一个实验性 API，后续可能修改或删除。

参数:

- **steps** (int, 可选) - *steps* 必须为非负整数。默认值: 10。
- **base** (int, 可选) - *base* 必须为非负整数。默认值: 10。
- **dtype** (mindspore.dtype, 可选) - 输出的数据类型, 支持 `mstype.float16`、`mstype.float32` 或 `mstype.float64`。默认值: `mstype.float32`。

输入:

- **start** (Tensor) - 间隔的起始值, *shape* 为 0-D, 数据类型为 `float16`、`float32` 或 `float64` (对于 GPU) 类型。
- **end** (Tensor) - 间隔的结束值, *shape* 为 0-D, 数据类型为 `float16`、`float32` 或 `float64` (对于 GPU) 类型。

输出:

Tensor, *shape* 为 (*step*,), 数据类型由属性 *dtype* 设置。

异常:

- **TypeError** - 若 *input* 不是一个 Tensor。
- **TypeError** - 若 *steps* 不是一个整数。
- **TypeError** - 若 *base* 不是一个整数。
- **TypeError** - 若 *dtype* 不是 `mindspore.float16`、`mindspore.float32` 或 `mindspore.float64`。
- **ValueError** - 若 *steps* 不是非负整数。
- **ValueError** - 若 *base* 不是非负整数。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> logspace = ops.LogSpace(steps = 10, base = 10, dtype=mstype.float32)
>>> start = Tensor(1, mstype.float32)
>>> end = Tensor(10, mstype.float32)
>>> output = logspace(start, end)
>>> print(output)
[1.e+01 1.e+02 1.e+03 1.e+04 1.e+05 1.e+06 1.e+07 1.e+08 1.e+09 1.e+10]
```

mindspore.ops.LuUnpack

class mindspore.ops.LuUnpack (*unpack_data=True, unpack_pivots=True*)

将 *LU_data* 和 *LU_pivots* 还原为为 P, L, U 矩阵, 其中 P 为置换矩阵, L 为下三角矩阵, U 为上三角矩阵。通常情况下, *LU_data* 和 *LU_pivots* 是矩阵通过 LU 分解生成的。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多参考详见 `mindspore.ops.lu_unpack()`。

参数:

- **unpack_data** (bool, 可选) - 是否解压缩 *LU_data* 的标志。如果为 `False`, 则返回的 L 和 U 为 `None`。默认值: `True`。

- **unpack_pivots** (bool, 可选) - 是否将 *LU_pivots* 解压缩为置换矩阵 *P* 的标志。如果为 *False*，则返回的 *P* 为 *None*。默认值: *True*。

输入:

- **LU_data** (Tensor) - Tensor, 打包的 LU 分解数据, shape 为 $(*, M, N)$, 其中 $*$ 为 batch 维度, 其中 $*$ 是 batch 维度, 数据类型为 int8、uint8、int16、int32、int64、float16、float32、float64。*LU_data* 的维度必须等于或大于 2。
- **LU_pivots** (Tensor) - Tensor, 打包的 LU 分枢轴, shape 为 $(*, \min(M, N))$, 其中 $*$ 为 batch 维度, 其中 $*$ 是 batch 维度, 数据类型为 int8、uint8、int16、int32 或 int64。

输出:

- **pivots** (Tensor) - LU 分解的置换矩阵, shape 为 $(*, M, M)$, 数据类型与 *LU_data* 相同。
- **L** (Tensor) - LU 分解的 L 矩阵, 数据类型与 *LU_data* 相同。
- **U** (Tensor) - LU 分解的 U 矩阵, 数据类型与 *LU_data* 相同。

支持平台:

GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> LU_data = Tensor(np.array([[[-0.3806, -0.4872, 0.5536],
...                             [-0.1287, 0.6508, -0.2396],
...                             [0.2583, 0.5239, 0.6902]],
...                             [[0.6706, -1.1782, 0.4574],
...                             [-0.6401, -0.4779, 0.6701],
...                             [0.1015, -0.5363, 0.6165]]]), mstype.float32)
>>> LU_pivots = Tensor(np.array([[1, 3, 3],
...                               [2, 3, 3]]), mstype.int32)
>>> lu_unpack = ops.LuUnpack()
>>> pivots, L, U = lu_unpack(LU_data, LU_pivots)
>>> print(pivots)
[[[1. 0. 0.]
  [0. 0. 1.]
  [0. 1. 0.]]

  [[0. 0. 1.]
  [1. 0. 0.]
  [0. 1. 0.]]]
>>> print(L)
[[[ 1.      0.      0.      ]
  [-0.1287  1.      0.      ]
  [ 0.2583  0.5239  1.      ]]

  [[ 1.      0.      0.      ]
  [-0.6401  1.      0.      ]
  [ 0.1015 -0.5363  1.      ]]]
>>> print(U)
[[[-0.3806 -0.4872  0.5536]
  [ 0.      0.6508 -0.2396]
  [ 0.      0.      0.6902]]

  [[ 0.6706 -1.1782  0.4574]
```

(下页继续)

(续上页)

```
[ 0.      -0.4779  0.6701]
[ 0.       0.      0.6165]]]
```

mindspore.ops.MaskedFill

class mindspore.ops.MaskedFill

```
prim = ops.MaskedFill()
out = prim(input_x, mask, value)
```

等价于

```
ops.masked_fill(input_x, mask, value)
```

更多详情请查看: [mindspore.ops.masked_fill\(\)](#)。

mindspore.ops.MaskedScatter

class mindspore.ops.MaskedScatter

返回一个 Tensor。根据 *mask* 和 *updates* 更新输入 Tensor 的值。

警告： 这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - 待更新的 Tensor。
- **mask** (Tensor[bool]) - 指示应修改或替换哪些元素的掩码 Tensor，*mask* 和 *x* 的 shape 必须相等或者两者的 shape 可以广播。
- **updates** (Tensor) - 要散播到目标张量或数组中的值，其数据类型与 *x* 相同。*updates* 中的元素数量必须大于等于 *mask* 中值为 True 的元素的数量。

输出：

Tensor，其数据类型和 shape 与 *x* 相同。

异常：

- **TypeError** - 如果 *x*、*mask* 或者 *updates* 不是 Tensor。
- **TypeError** - 如果 *x* 的数据类型不被支持。
- **TypeError** - 如果 *mask* 的 dtype 不是 bool。
- **TypeError** - 如果 *x* 的维度数小于 *mask* 的维度数。
- **ValueError** - 如果 *mask* 不能广播到 *x*。
- **ValueError** - 如果 *updates* 中的元素数目小于 *mask* 中的 True 元素的数量。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([1., 2., 3., 4.]), mindspore.float32)
>>> mask = Tensor(np.array([True, True, False, True]), mindspore.bool_)
>>> updates = Tensor(np.array([5., 6., 7.]), mindspore.float32)
>>> output = ops.MaskedScatter()(input_x, mask, updates)
>>> print(output)
[5. 6. 3. 7.]
```

mindspore.ops.MaskedSelect

class mindspore.ops.MaskedSelect

```
prim = ops.MaskedSelect()
out = prim(input, mask)
```

等价于

```
ops.masked_select(input, mask)
```

更多详情请查看：[mindspore.ops.masked_select\(\)](#)。

mindspore.ops.MatrixBandPart

class mindspore.ops.MatrixBandPart

提取一个 Tensor 中每个矩阵的中心带，中心带之外的所有值都设置为零。

更多参考详见[mindspore.ops.matrix_band_part\(\)](#)。

警告： 这是一个实验性 API，后续可能修改或删除。

输入：

- **x** (Tensor) - x 的 shape 为 $(*, m, n)$ ，其中 $*$ 表示任意 batch 维度。
- **lower** (Union[int, Tensor]) - 要保留的下部子对角线数。其数据类型必须是 int32 或 int64。如果为负数，则保留整个下三角形。
- **upper** (Union[int, Tensor]) - 要保留的上部子对角线数。其数据类型必须是 int32 或 int64。如果为负数，则保留整个上三角形。

输出：

Tensor，其数据类型和维度必须和输入中的 x 保持一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> matrix_band_part = ops.MatrixBandPart()
>>> x = np.ones([2, 4, 4]).astype(np.float32)
>>> output = matrix_band_part(Tensor(x), 2, 1)
>>> print(output)
[[[1. 1. 0. 0.]
  [1. 1. 1. 0.]
  [1. 1. 1. 1.]
  [0. 1. 1. 1.]]
 [[1. 1. 0. 0.]
  [1. 1. 1. 0.]
  [1. 1. 1. 1.]
  [0. 1. 1. 1.]]]
```

mindspore.ops.MatrixDiagPartV3

class mindspore.ops.**MatrixDiagPartV3** (align='RIGHT_LEFT')

返回 Tensor 的对角线部分。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.matrix_diag_part()`。

参数：

- **align** (str, 可选) - 可选字符串，指定超对角线和次对角线的对齐方式。可选值："RIGHT_LEFT"、"LEFT_RIGHT"、"LEFT_LEFT"、"RIGHT_RIGHT"。默认值："RIGHT_LEFT"。
 - "RIGHT_LEFT" 表示将超对角线与右侧对齐（左侧填充行），将次对角线与左侧对齐（右侧填充行）。
 - "LEFT_RIGHT" 表示将超对角线与左侧对齐（右侧填充行），将次对角线与右侧对齐（左侧填充行）。
 - "LEFT_LEFT" 表示将超对角线和次对角线均与左侧对齐（右侧填充行）。
 - "RIGHT_RIGHT" 表示将超对角线和次对角线均与右侧对齐（左侧填充行）。

输入：

- **x** (Tensor) - 输入 Tensor，维度 r 需要满足 $r \geq 2$ 。
- **k** (Tensor) - int 或 int32 类型的 Tensor，对角线偏移量。正值表示超对角线，0 表示主对角线，负值表示次对角线。k 可以是单个整数（对于单个对角线）或一对整数，分别指定矩阵带的上界和下界，且 $k[0]$ 不得大于 $k[1]$ 。该值必须在 $(-x.shape[-2], x.shape[-1])$ 范围内。
- **padding_value** (Tensor) - 与 x 相同的数据类型的单值 Tensor，表示填充对角线带外区域的数值。

输出：

Tensor，与 x 的类型相同。

- 设 x 有 r 维 $(I, J, \dots, M, N)$ 。设 max_diag_len 为所有对角线长度中的最大值，则 $max\_diag\_len = \min(M + \min(k[1], 0), N + \min(-k[0], 0))$ 。
- 设 num_diags 为输出的维度数，则有 $num\_diags = k[1] - k[0] + 1$ 。如果 $num\_diags == 1$ ，则输出 Tensor 的维度为 $r - 1$ ，分别为 $(I, J, \dots, L, max\_diag\_len)$ 。否则，输出 Tensor 的维度为 r ，分别为 $(I, J, \dots, L, num\_diags, max\_diag\_len)$ 。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[1, 2, 3, 4],
...                      [5, 6, 7, 8],
...                      [9, 8, 7, 6]]), mindspore.float32)
>>> k = Tensor(np.array([1, 3]), mindspore.int32)
>>> padding_value = Tensor(np.array(9), mindspore.float32)
>>> matrix_diag_part_v3 = ops.MatrixDiagPartV3(align='RIGHT_LEFT')
>>> output = matrix_diag_part_v3(x, k, padding_value)
>>> print(output)
[[9. 9. 4.]
 [9. 3. 8.]
 [2. 7. 6.]]
>>> print(output.shape)
(3, 3)
```

mindspore.ops.MatrixDiagV3

class mindspore.ops.**MatrixDiagV3** (align='RIGHT_LEFT')

构造以输入 Tensor 为对角线的矩阵。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多参考详见 `mindspore.ops.matrix_diag()`。

参数:

- **align** (str, 可选) - 可选字符串, 指定超对角线和次对角线的对齐方式。可选值: "RIGHT_LEFT"、"LEFT_RIGHT"、"LEFT_LEFT"、"RIGHT_RIGHT"。默认值: "RIGHT_LEFT"。
 - "RIGHT_LEFT" 表示将超对角线与右侧对齐 (左侧填充行), 将次对角线与左侧对齐 (右侧填充行)。
 - "LEFT_RIGHT" 表示将超对角线与左侧对齐 (右侧填充行), 将次对角线与右侧对齐 (左侧填充行)。
 - "LEFT_LEFT" 表示将超对角线和次对角线均与左侧对齐 (右侧填充行)。
 - "RIGHT_RIGHT" 表示将超对角线和次对角线均与右侧对齐 (左侧填充行)。

输入:

- **x** (Tensor) - 对角线 Tensor。
- **k** (Union[int, Tensor], 可选) - 对角线偏移。int32 类型的 Tensor。正值表示超对角线, 0 表示主对角线, 负值表示次对角线。k 可以是单个整数 (对于单个对角线) 或一对整数, 分别指定矩阵带的上界和下界, 且 $k[0]$ 不得大于 $k[1]$ 。该值必须在 $(-\text{num\_rows}, \text{num\_cols})$ 范围内。默认值: 0。
- **num_rows** (Union[int, Tensor], 可选) - 输出 Tensor 的行数。int32 类型的单值 Tensor, 若该值为 -1, 则表示输出 Tensor 的最内层矩阵是一个方阵, 实际行数将由其他输入推导, 即 $\text{num\_rows} = x.\text{shape}[-1] - \min(k[1], 0)$; 否则, 该值必须大于或等于 $x.\text{shape}[-1] - \min(k[1], 0)$ 。默认值: -1。
- **num_cols** (Union[int, Tensor], 可选) - 输出 Tensor 的列数。int32 类型的单值 Tensor, 若该值为 -1, 则表示输出 Tensor 的最内层矩阵是一个方阵, 实际列数将由其他输入推导, 即 $\text{num\_cols} = x.\text{shape}[-1] + \max(k[0], 0)$; 否则, 该值必须大于或等于 $x.\text{shape}[-1] - \min(k[1], 0)$ 。默认值: -1。

- **padding_value** (Union[int, float, Tensor], 可选) - 填充对角线带外区域的数值, 是一个数据类型与 x 相同的单值 Tensor。默认值: 0。

输出:

Tensor, 与 x 的数据类型相同。设 x 有 r 维 $(I, J, \dots, M, N)$, 当只给出一条对角线 (k 是整数或 $k[0]==k[1]$) 时, 输出 Tensor 的维度是 $r+1$, shape 为 $(I, J, \dots, M, \text{num\_rows}, \text{num\_cols})$ 。否则, 输出 Tensor 的维度是 r , shape 为 $(I, J, \dots, \text{num\_rows}, \text{num\_cols})$ 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[8, 9, 0],
...                      [1, 2, 3],
...                      [0, 4, 5]]), mindspore.float32)
>>> k = Tensor(np.array([-1, 1]), mindspore.int32)
>>> num_rows = Tensor(np.array(3), mindspore.int32)
>>> num_cols = Tensor(np.array(3), mindspore.int32)
>>> padding_value = Tensor(np.array(11), mindspore.float32)
>>> matrix_diag_v3 = ops.MatrixDiagV3(align='LEFT_RIGHT')
>>> output = matrix_diag_v3(x, k, num_rows, num_cols, padding_value)
>>> print(output)
[[ 1.  8. 11.]
 [ 4.  2.  9.]
 [11.  5.  3.]]
>>> print(output.shape)
(3, 3)
```

mindspore.ops.MatrixSetDiagV3

class mindspore.ops.MatrixSetDiagV3 (align='RIGHT_LEFT')

更新批处理矩阵对角线的值。给定输入 x 和对角线 $diagonal$, 此操作返回一个 Tensor。该 Tensor 最内层矩阵的对角线的值将被 $diagonal$ 中的值替换。

如果某些对角线比 max_diag_len 短, 则需要被填充, 其中 max_diag_len 指对角线的最长长度。 $diagonal$ 的维度 $\text{shape}[-2]$ 必须等于对角线个数 num_diags , $\text{num\_diags} = k[1] - k[0] + 1$, $diagonal$ 的维度 $\text{shape}[-1]$ 必须等于最长对角线值 max_diag_len , $\text{max\_diag\_len} = \min(x.\text{shape}[-2] + \min(k[1], 0), x.\text{shape}[-1] + \min(-k[0], 0))$ 。

设 x 是一个 n 维 Tensor, shape 为: $(d_1, d_2, \dots, d_{n-2}, d_{n-1}, d_n)$ 。当 k 是一个整数或 $k[0] == k[1]$ 时, $diagonal$ 为 $n-1$ 维 Tensor, shape 为 $(d_1, d_2, \dots, d_{n-2}, \text{max\_diag\_len})$ 。否则, $diagonal$ 与 x 维度一致, 其 shape 为 $(d_1, d_2, \dots, d_{n-2}, \text{num\_diags}, \text{max\_diag\_len})$ 。

警告: 这是一个实验性 API, 后续可能修改或删除。

参数:

- **align** (str, 可选) - 可选字符串, 指定超对角线和次对角线的对齐方式。可选值: "RIGHT_LEFT"、"LEFT_RIGHT"、"LEFT_LEFT"、"RIGHT_RIGHT"。默认值: "RIGHT_LEFT"。
 - "RIGHT_LEFT" 表示将超对角线与右侧对齐 (左侧填充行), 将次对角线与左侧对齐 (右侧填充行)。
 - "LEFT_RIGHT" 表示将超对角线与左侧对齐 (右侧填充行), 将次对角线与右侧对齐 (左侧填充行)。

- "LEFT_LEFT" 表示将超对角线和次对角线均与左侧对齐（右侧填充行）。
- "RIGHT_RIGHT" 表示将超对角线和次对角线均与右侧对齐（左侧填充行）。

输入：

- **x** (Tensor) - n 维 Tensor，其中 $n \geq 2$ 。
- **diagonal** (Tensor) - 输入对角线 Tensor，具有与 x 相同的数据类型。当 k 是整数或 $k[0] == k[1]$ 时，其为维度 $n - 1$ ，否则，其维度为 n 。
- **k** (Tensor) - int32 类型的 Tensor。对角线偏移量。正值表示超对角线，0 表示主对角线，负值表示次对角线。 k 可以是单个整数（对于单个对角线）或一对整数，分别指定矩阵带的上界和下界，且 $k[0]$ 不得大于 $k[1]$ 。其值必须在 $(-x.shape[-2], x.shape[-1])$ 中。采用图模式时，输入 k 必须是常量 Tensor。

输出：

Tensor，数据类型和 shape 与 x 相同。

异常：

- **TypeError** - 若任一输入不是 Tensor。
- **TypeError** - x 与 *diagonal* 数据类型不同。
- **TypeError** - k 的数据类型不为 int32。
- **ValueError** - *align* 取值不在合法值集合内。
- **ValueError** - k 的维度不为 0 或 1。
- **ValueError** - x 的维度不大于等于 2。
- **ValueError** - k 的大小不为 1 或 2。
- **ValueError** - 当 k 的大小为 2 时， $k[1]$ 小于 $k[0]$ 。
- **ValueError** - 对角线 *diagonal* 的维度与输入 x 的维度不匹配。
- **ValueError** - 对角线 *diagonal* 的 shape 与输入 x 不匹配。
- **ValueError** - 对角线 *diagonal* 的维度 $shape[-2]$ 不等于与对角线个数 num_diags ， $num\_diags = k[1] - k[0] + 1$ 。
- **ValueError** - k 的取值不在 $(-x.shape[-2], x.shape[-1])$ 范围内。
- **ValueError** - 对角线 *diagonal* 的维度 $shape[-1]$ 不等于最长对角线长度 max_diag_len ， $max\_diag\_len = \min(x.shape[-2] + \min(k[1], 0), x.shape[-1] + \min(-k[0], 0))$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[7, 7, 7, 7],
...                      [7, 7, 7, 7],
...                      [7, 7, 7, 7]]), mindspore.float32)
>>> diagonal = Tensor(np.array([[0, 9, 1],
...                             [6, 5, 8],
...                             [1, 2, 3],
...                             [4, 5, 0]]), mindspore.float32)
>>> k = Tensor(np.array([-1, 2]), mindspore.int32)
```

(下页继续)

(续上页)

```
>>> matrix_set_diag_v3 = ops.MatrixSetDiagV3(align='RIGHT_LEFT')
>>> output = matrix_set_diag_v3(x, diagonal, k)
>>> print(output)
[[1. 6. 9. 7.]
 [4. 2. 5. 1.]
 [7. 5. 3. 8.]]
>>> print(output.shape)
(3, 4)
```

mindspore.ops.MatrixSolve

class mindspore.ops.MatrixSolve (*adjoint=False*)

求解线性方程组。

参数：

- **adjoint** (bool, 可选) - 指明是否使用矩阵的伴随矩阵进行求解。默认值：False，使用转置矩阵进行求解。

输入：

- **matrix** (Tensor) - Tensor，线性方程组系数组成的矩阵，其 shape 为 $(..., M, M)$ 。
- **rhs** (Tensor) - Tensor，线性方程组结果值组成的矩阵，其 shape 为 $(..., M, K)$ ，*rhs* 与 *matrix* 的类型必须一致。

输出：

Tensor，线性方程组解组成的矩阵，与 *rhs* 的 shape 及类型均相同。

异常：

- **TypeError** - 如果 *adjoint* 不是 bool 型。
- **TypeError** - 如果 *matrix* 的类型不是以下之一：mstype.float16、mstype.float32、mstype.float64、mstype.complex64、mstype.complex128。
- **TypeError** - 如果 *rhs* 与 *matrix* 的类型不一致。
- **ValueError** - 如果 *matrix* 的秩小于 2。
- **ValueError** - 如果 *matrix* 和 *rhs* 的维度不同。
- **ValueError** - 如果 *matrix* 的最内两维不同。
- **ValueError** - 如果 *rhs* 的最内两维与 *matrix* 不匹配。

支持平台：

Ascend CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> matrix = Tensor(np.array([[1.0 , 4.0],
...                           [2.0 , 7.0]]), mindspore.float32)
>>> rhs = Tensor(np.array([[1.0] , [3.0]]), mindspore.float32)
>>> matrix_solve = ops.MatrixSolve(adjoint = False)
>>> output = matrix_solve(matrix, rhs)
>>> print(output)
[[5.0]
 [-1.0]]
```

mindspore.ops.Meshgrid

class mindspore.ops.**Meshgrid**(*indexing='xy'*)

从给定的 Tensor 生成网格矩阵。给定 N 个一维 Tensor，对每个 Tensor 做扩张操作，返回 N 个 N 维的 Tensor。

更多参考详见 `mindspore.ops.meshgrid()`。

参数：

- **indexing** (str, 可选) - 以笛卡尔坐标 'xy' 或者矩阵 'ij' 索引作为输出。对于长度为 M 和 N 的二维输入，取值为 'xy' 时，输出的 shape 为 (N, M) ，取值为 'ij' 时，输出的 shape 为 (M, N) 。对于长度为 M, N 和 P 的三维输入，取值为 'xy' 时，输出的 shape 为 (N, M, P) ，取值为 'ij' 时，输出的 shape 为 (M, N, P) 。默认值：'xy'。

输入：

- **inputs** (Union(tuple[Tensor], list[Tensor])) - 静态图下为 N 个一维 Tensor，输入的 Tensor 个数应大于 1。动态图下为 N 个零维或一维 Tensor，输入的 Tensor 个数应大于 0。数据类型为 Number。

输出：

Tensor， N 个 N 维 Tensor 对象的元组。数据类型与输入相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([1, 2, 3, 4]).astype(np.int32))
>>> y = Tensor(np.array([5, 6, 7]).astype(np.int32))
>>> z = Tensor(np.array([8, 9, 0, 1, 2]).astype(np.int32))
>>> inputs = (x, y, z)
>>> meshgrid = ops.Meshgrid(indexing='xy')
>>> output = meshgrid(inputs)
>>> print(output)
(Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2],
  [3, 3, 3, 3, 3],
  [4, 4, 4, 4, 4]],
 [[1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2],
  [3, 3, 3, 3, 3],
  [4, 4, 4, 4, 4]],
 [[1, 1, 1, 1, 1],
  [2, 2, 2, 2, 2],
  [3, 3, 3, 3, 3],
  [4, 4, 4, 4, 4]]]),
 Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5],
  [5, 5, 5, 5, 5]],
 [[6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6],
  [6, 6, 6, 6, 6]],
 [[7, 7, 7, 7, 7],
```

(下页继续)

(续上页)

```

    [7, 7, 7, 7, 7],
    [7, 7, 7, 7, 7],
    [7, 7, 7, 7, 7]]]),
Tensor(shape=[3, 4, 5], dtype=Int32, value=
[[[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]],
 [[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]],
 [[8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2],
  [8, 9, 0, 1, 2]]]])

```

mindspore.ops.Mvlgamma

class mindspore.ops.Mvlgamma(*p*)
 逐元素计算 *p* 维多元对数伽马函数值。

警告：这是一个实验性 API，后续可能修改或删除。

更多细节请参考 `mindspore.ops.mvlgamma()`。

参数：

- **p** (int) - 进行计算的维度，必须大于等于 1。

输入：

- **x** (Tensor) - 多元对数伽马函数的输入 Tensor，支持数据类型为 float32 和 float64。其 shape 为 (N,*), 其中 * 为任意数量的附加维度。x 中每个元素的值必须大于 $(p-1)/2$ 。

输出：

Tensor, shape 和数据类型与 *x* 一致。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[3, 4, 5], [4, 2, 6]]), mindspore.float32)
>>> op = ops.Mvlgamma(p=3)
>>> y = op(x)
>>> print(y)
[[ 2.694925   5.402975   9.140645 ]
 [ 5.402975   1.5963125 13.640454 ]]

```

mindspore.ops.NanToNum

class mindspore.ops.NanToNum(*nan=None, posinf=None, neginf=None*)

```
prim = ops.NanToNum(nan, posinf, neginf)
out = prim(input)
```

等价于

```
ops.nan_to_num(input, nan, posinf, neginf)
```

更多详情请查看: [mindspore.ops.nan_to_num\(\)](#)。

mindspore.ops.NonZero

class mindspore.ops.NonZero

返回所有非零元素的索引位置。

输入:

- **input** (Tensor) - 输入 Tensor。
 - Ascend: 其秩可以等于 0, O2 模式除外。
 - CPU/GPU: 其秩应大于等于 1。

输出:

二维 Tensor, 数据类型为 int64, 包含所有输入中的非零元素的索引位置。如果 *input* 的维数为 *D*, *input* 中的非零个数为 *N*, 则输出的 shape 为 (N, D) 。

异常:

- **TypeError** - 如果 *input* 不是 Tensor。
- **RuntimeError** - 在 CPU 或者 GPU 或者 Ascend 的 O2 模式中, 如果 *input* 的维度为 0。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> input = mindspore.tensor([1, 0, 2, 0, 3], mindspore.int32)
>>> output = mindspore.ops.NonZero()(input)
>>> print(output)
[[0]
 [2]
 [4]]
```

mindspore.ops.ParallelConcat

class mindspore.ops.ParallelConcat

根据第一个维度连接输入 Tensor。

Concat 和 ParallelConcat 之间的区别在于, Concat 要求在操作开始之前需要计算所有的输入, 但不要求在构图期间知道输入 shape。ParallelConcat 将在输入片段可用时, 将其复制到输出中, 在某些情况下, 这可以提供性能优势。

说明: 输入 Tensor 在第一个维度要求长度为 1。

输入:

- **values** (tuple, list) - 由 Tensor 组成的 tuple 或 list。其元素的数据类型和 shape 必须相同, 且每个 Tensor 的 rank 不能小于 1。CPU 上支持数据类型为数值型, Ascend 上支持数据类型为除去 [float64, complex64, complex128] 三种数据类型外的数值型。

输出:

Tensor, 数据类型与 values 相同。

异常:

- **TypeError** - 如果输入不是 Tensor。
- **TypeError** - 如果 values 中各个 Tensor 的数据类型和 shape 不相同。
- **ValueError** - 如果任意一个输入 Tensor 的 shape 第一维长度不是 1。
- **ValueError** - 如果 values 中的 Tensor 的 rank 小于 1。
- **ValueError** - 如果输入各 Tensor 的 shape 不一致。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> data1 = Tensor(np.array([[0, 1]]).astype(np.int32))
>>> data2 = Tensor(np.array([[2, 1]]).astype(np.int32))
>>> op = ops.ParallelConcat()
>>> output = op((data1, data2))
>>> print(output)
[[0 1]
 [2 1]]
```

mindspore.ops.PopulationCount

class mindspore.ops.PopulationCount

计算二进制数中 1 的个数。

更多参考详见 `mindspore.ops.population_count()`。

输入:

- **input_x** (Tensor) - 任意维度的 Tensor。Ascend 平台支持的数据类型为 int16、uint16, CPU 和 GPU 平台支持的数据类型为 int8、int16、int32、int64、uint8、uint16、uint32、uint64。

输出:

Tensor, shape 与 *input_x* 相同, 数据类型为 uint8。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> input_x = Tensor([0, 1, 3], mindspore.int16)
>>> output = ops.PopulationCount()(input_x)
>>> print(output)
[0 1 2]
```

mindspore.ops.RandomShuffle

class mindspore.ops.**RandomShuffle** (*seed=0, seed2=0*)

沿着 Tensor 的第一个维度进行随机打乱操作。

说明:

- 随机种子: 通过一些复杂的数学算法, 可以得到一组有规律的随机数, 而随机种子就是这个随机数的初始值。随机种子相同, 得到的随机数就不会改变。
 - 全局的随机种子和算子层的随机种子都没设置或都设置为 0: 完全随机。
 - 全局的随机种子设置了, 算子层的随机种子未设置: 采用全局的随机种子和 0 拼接。
 - 全局的随机种子未设置, 算子层的随机种子设置了: 使用 0 和算子层的随机种子拼接。
 - 全局的随机种子和算子层的随机种子都设置了: 全局的随机种子和算子层的随机种子拼接。
-

参数:

- **seed** (int, 可选) - 算子层的随机种子, 用于生成随机数。必须是非负的。默认值: 0。
- **seed2** (int, 可选) - 全局的随机种子, 和算子层的随机种子共同决定最终生成的随机数。必须是非负的。默认值: 0。

输入:

- **x** (tuple) - 需要进行打乱操作的 Tensor。

输出:

Tensor。输出的 shape 和类型与输入 *x* 相同。

异常:

- **TypeError** - 如果 *seed* 或 *seed2* 的数据类型不是 int。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([1, 2, 3, 4]), mstype.float32)
>>> shuffle = ops.RandomShuffle(seed=1, seed2=1)
>>> output = shuffle(x)
>>> print(output.shape)
(4,)
```

mindspore.ops.Range

class mindspore.ops.Range (maxlen=1000000)

```
prim = ops.Range(maxlen)
out = prim(start, end, step)
```

等价于

```
ops.range(start, end, step, maxlen)
```

更多详情请查看: [mindspore.ops.range\(\)](#)。

mindspore.ops.Rank

class mindspore.ops.Rank

返回输入 Tensor 的秩。

详情请查看[mindspore.ops.rank\(\)](#)。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_tensor = Tensor(np.array([[2, 2], [2, 2]]), mindspore.float32)
>>> rank = ops.Rank()
>>> output = rank(input_tensor)
>>> print(output)
2
>>> print(type(output))
<class 'int'>
```

mindspore.ops.Renorm

class mindspore.ops.Renorm(*p*, *dim*, *maxnorm*)

对 Tensor 沿着指定维度 *dim* 进行重新规范化, 要求每个子 Tensor 的 *p* 范数不超过 *maxnorm*。如果子 Tensor 的 *p* 范数小于 *maxnorm*, 则其值不需要改变。否则, 子 Tensor 需要修改为相应位置的原始值除以子 Tensor 的 *p* 范数, 然后再乘以 *maxnorm*。

更多参考详见 [mindspore.ops.renorm\(\)](#)。

参数:

- **p** (int) - 范数计算的幂。
- **dim** (int) - 获得子 Tensor 的维度。
- **maxnorm** (float32) - 给定的最大范数。

输入:

- **x** (Tensor) - 输入 Tensor, 类型为 float32 或者 float16。

输出:

Tensor, shape 和 type 与输入 Tensor 一致。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[1, 1, 1], [2, 2, 2], [3, 3, 3]]), mindspore.float32)
>>> y = ops.Renorm(p=1, dim=0, maxnorm=5.)(x)
>>> print(y)
[[1.          1.          1.          ]
 [1.66666666 1.66666666 1.66666666 ]
 [1.66666667 1.66666667 1.66666667 ]]
```

mindspore.ops.Reshape

class mindspore.ops.Reshape

```
prim = ops.Reshape()
out = prim(input, shape)
```

等价于

```
ops.reshape(input, shape)
```

更多详情请查看: [mindspore.ops.reshape\(\)](#)。

mindspore.ops.ReverseSequence

class mindspore.ops.ReverseSequence (seq_dim, batch_dim=0)

对输入序列进行部分反转。

参数:

- **seq_dim** (int) - 指定反转的维度, 此值为必填参数。
- **batch_dim** (int, 可选) - 指定切片维度。默认值: 0。

输入:

- **x** (Tensor) - 输入需反转的数据, 其数据类型支持包括 **bool** 在内的所有数值型。
- **seq_lengths** (Tensor) - 指定反转长度, 为一维向量, 其数据类型为 **int32** 或 **int64**。

输出:

Tensor, shape 和数据类型与输入 *x* 相同。

异常:

- **TypeError** - *seq_dim* 或 *batch_dim* 不是 **int**。
- **ValueError** - 如果 $\text{len}(\text{seq\_lengths}) \neq \text{x.shape}[\text{batch\_dim}]$ 。
- **ValueError** - 如果 $\text{batch\_dim} == \text{seq\_dim}$ 。
- **ValueError** - 如果 $\text{seq\_dim} < 0$ 或 $\text{seq\_dim} \geq \text{len}(\text{x.shape})$ 。
- **ValueError** - 如果 $\text{batch\_dim} < 0$ 或 $\text{batch\_dim} \geq \text{len}(\text{x.shape})$ 。
- **RuntimeError** - 如果 *seq_lengths* 中的任意一个值小于 0。
- **RuntimeError** - 如果 *seq_lengths* 中的任意一个值大于 $\text{x.shape}[\text{seq\_dim}]$ 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.float32)
>>> seq_lengths = Tensor(np.array([1, 2, 3]))
>>> reverse_sequence = ops.ReverseSequence(seq_dim=1)
>>> output = reverse_sequence(x, seq_lengths)
>>> print(output)
[[1. 2. 3.]
 [5. 4. 6.]
 [9. 8. 7.]]
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.float32)
>>> seq_lengths = Tensor(np.array([1, 2, 3]))
>>> reverse_sequence = ops.ReverseSequence(seq_dim=0, batch_dim=1)
>>> output = reverse_sequence(x, seq_lengths)
>>> print(output)
[[1. 5. 9.]
 [4. 2. 6.]
 [7. 8. 3.]]
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.float32)
>>> seq_lengths = Tensor(np.array([2, 2, 3]))
```

(下页继续)

(续上页)

```

>>> reverse_sequence = ops.ReverseSequence(seq_dim=1)
>>> output = reverse_sequence(x, seq_lengths)
>>> print(output)
[[2. 1. 3.]
 [5. 4. 6.]
 [9. 8. 7.]]
>>> x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]), mindspore.float32)
>>> seq_lengths = Tensor(np.array([3, 2, 3]))
>>> reverse_sequence = ops.ReverseSequence(seq_dim=1)
>>> output = reverse_sequence(x, seq_lengths)
>>> print(output)
[[3. 2. 1.]
 [5. 4. 6.]
 [9. 8. 7.]]
>>> x = Tensor(np.array([[1, 2, 3, 4], [5, 6, 7, 8]]), mindspore.float32)
>>> seq_lengths = Tensor(np.array([4, 4]))
>>> reverse_sequence = ops.ReverseSequence(seq_dim=1)
>>> output = reverse_sequence(x, seq_lengths)
>>> print(output)
[[4. 3. 2. 1.]
 [8. 7. 6. 5.]]

```

mindspore.ops.ReverseV2

class mindspore.ops.ReverseV2 (axis)

```

prim = ops.ReverseV2(axis)
out = prim(input)

```

等价于

```
ops.flip(input, axis)
```

更多详情请查看: [mindspore.ops.flip\(\)](#)。

mindspore.ops.RightShift

class mindspore.ops.RightShift

将 Tensor *input_x* 的每个元素右移 Tensor *input_y* 中对应位数。输入为两个 Tensor，数据类型需要保持一致，他们之间的 shape 可以广播。

$$out_i = x_i \gg y_i$$

警告： 这是一个实验性 API，后续可能修改或删除。

输入：

- **input_x** (Tensor) - 被操作 Tensor，将被逐元素位右移 *input_y* 位。支持所有 int 和 uint 类型。
- **input_y** (Tensor) - 右移位数。数据类型必须和 *input_x* 一致。

输出：

- **output** (Tensor) - 输出 Tensor，数据类型和 *input_x* 一致。

异常：

- **TypeError** - *input_x* 或者 *input_y* 不是 Tensor。
- **TypeError** - *input_x* 和 *input_y* 不能发生广播。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([1, 2, 3]).astype(np.uint8))
>>> input_y = Tensor(np.array([1, 1, 1]).astype(np.uint8))
>>> output = ops.RightShift()(input_x, input_y)
>>> print(output)
[0 1 1]
```

mindspore.ops.ScatterNd

class mindspore.ops.ScatterNd

```
prim = ops.ScatterNd()
out = prim(indices, updates, shape)
```

等价于

```
ops.scatter_nd(indices, updates, shape)
```

更多详情请查看：[mindspore.ops.scatter_nd\(\)](#)。

mindspore.ops.ScatterNdDiv

class mindspore.ops.ScatterNdDiv(*use_locking=False*)

将稀疏除法应用于张量中的单个值或切片。

使用给定值通过除法运算和输入索引更新 *input_x* 的值。为便于使用更新后的值，函数返回 *input_x* 的副本。

警告： 这是一个实验性 API，后续可能修改或删除。

更多参考详见[mindspore.ops.scatter_nd_div\(\)](#)。

参数：

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值：False。

输入：

- **input_x** (Union[Parameter, Tensor]) - 输入参数，数据类型是 Parameter 或 Tensor。
- **indices** (Tensor) - 指定除法操作的索引，数据类型为 mindspore.int32 或 mindspore.int64。索引的 rank 必须至少为 2，并且 *indices.shape[-1] <= len(shape)*。

- **updates** (Tensor) - 指定与 *input_x* 进行除法操作的 Tensor, 数据类型与 *input_x* 相同, shape 为 *indices.shape[:-1] + x.shape[indices.shape[-1]:]*。

输出:

Tensor, 更新后的 *input_x*, shape 和数据类型与 *input_x* 相同。

支持平台:

GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8]), mindspore.float32), name=
↳ "x")
>>> indices = Tensor(np.array([[2], [4], [1], [7]]), mindspore.int32)
>>> updates = Tensor(np.array([6, 7, 8, 9]), mindspore.float32)
>>> use_locking = False
>>> scatter_nd_div = ops.ScatterNdDiv(use_locking)
>>> output = scatter_nd_div(input_x, indices, updates)
>>> print(output)
[1.          0.25          0.5          4.          0.71428573  6.
 7.          0.88888889 ]
>>> input_x = Parameter(Tensor(np.ones((4, 4, 4)), mindspore.float32))
>>> indices = Tensor(np.array([[0], [2]]), mindspore.int32)
>>> updates = Tensor(np.array([[[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]]),
↳ mindspore.float32)
>>> use_locking = False
>>> scatter_nd_div = ops.ScatterNdDiv(use_locking)
>>> output = scatter_nd_div(input_x, indices, updates)
>>> print(output)
[[[1.          1.          1.          1.          ]
  [0.5          0.5          0.5          0.5          ]
  [0.33333334  0.33333334  0.33333334  0.33333334]
  [0.25          0.25          0.25          0.25          ]]
 [[1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]]
 [[0.2          0.2          0.2          0.2          ]
  [0.16666667  0.16666667  0.16666667  0.16666667]
  [0.14285715  0.14285715  0.14285715  0.14285715]
  [0.125          0.125          0.125          0.125          ]]
 [[1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]
  [1.          1.          1.          1.          ]]]]
```

mindspore.ops.ScatterNdMax

class mindspore.ops.ScatterNdMax (use_locking=False)

对张量中的单个值或切片计算稀疏最大值。

使用给定值通过最大值运算和输入索引更新 **Parameter** 或 **Tensor** 值。在更新完成后输出 *input_x*，这有利于更加方便地使用更新后的值。

更多参考详见 `mindspore.ops.scatter_nd_max()`。

参数：

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值: False。

输入：

- **input_x** (Union[Parameter, Tensor]) - 输入参数，数据类型为 **Parameter** 或 **Tensor**。
- **indices** (Tensor) - 指定最大值计算的索引，数据类型为 `mindspore.int32` 或 `mindspore.int64`。索引的 rank 必须至少为 2，并且 `indices.shape[-1] <= len(shape)`。
- **updates** (Tensor) - 指定与 *input_x* 计算最大值的 **Tensor**，数据类型与 *input_x* 相同，shape 为 `indices.shape[:-1] + x.shape[indices.shape[-1]:]`。

输出：

Tensor，更新后的 *input_x*，shape 和数据类型与 *input_x* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8]), mindspore.float32), name=
↳ "x")
>>> indices = Tensor(np.array([[2], [4], [1], [7]]), mindspore.int32)
>>> updates = Tensor(np.array([6, 7, 8, 9]), mindspore.float32)
>>> scatter_nd_max = ops.ScatterNdMax()
>>> output = scatter_nd_max(input_x, indices, updates)
>>> print(output)
[ 1.  8.  6.  4.  7.  6.  7.  9.]
>>> input_x = Parameter(Tensor(np.ones((4, 4, 4)), mindspore.int32))
>>> indices = Tensor(np.array([[0], [2]]), mindspore.int32)
>>> updates = Tensor(np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]])),
↳ mindspore.int32)
>>> scatter_nd_max = ops.ScatterNdMax()
>>> output = scatter_nd_max(input_x, indices, updates)
>>> print(output)
[[[1 1 1 1]
  [2 2 2 2]
  [3 3 3 3]
  [4 4 4 4]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]
 [[5 5 5 5]
```

(下页继续)

(续上页)

```
[6 6 6 6]
[7 7 7 7]
[8 8 8 8]]
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]]
```

mindspore.ops.ScatterNdMin

class mindspore.ops.ScatterNdMin (use_locking=False)

对 Tensor 中的单个值或切片应用稀疏最小值。

使用给定值通过最小值运算和输入索引更新 *input_x* 的值。为便于使用更新后的值，函数返回 *input_x* 的副本。

更多参考详见 `mindspore.ops.scatter_nd_min()`。

参数：

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值: False。

输入：

- **input_x** (Union[Parameter, Tensor]) - 输入参数，数据类型为 Parameter 或 Tensor。
- **indices** (Tensor) - 指定最小值操作的索引，数据类型为 mindspore.int32 或 mindspore.int64。索引的 rank 必须至少为 2，并且 `indices.shape[-1] <= len(shape)`。
- **updates** (Tensor) - 指定与 *input_x* 操作的 Tensor，数据类型与 *input_x* 相同，shape 为 `indices.shape[:-1] + x.shape[indices.shape[-1]:]`。

输出：

Tensor，更新后的 *input_x*，shape 和数据类型与 *input_x* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.ones(8) * 10, mindspore.float32), name="x")
>>> indices = Tensor(np.array([[2], [4], [1], [7]]), mindspore.int32)
>>> updates = Tensor(np.array([6, 7, 8, 9]), mindspore.float32)
>>> use_locking = False
>>> scatter_nd_min = ops.ScatterNdMin(use_locking)
>>> output = scatter_nd_min(input_x, indices, updates)
>>> print(output)
[10.  8.  6. 10.  7. 10. 10.  9.]
>>> input_x = Parameter(Tensor(np.ones((4, 4, 4)) * 10, mindspore.int32))
>>> indices = Tensor(np.array([[0], [2]]), mindspore.int32)
>>> updates = Tensor(np.array([[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                           [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]),
↪mindspore.int32)
>>> use_locking = False
>>> scatter_nd_min = ops.ScatterNdMin(use_locking)
```

(下页继续)

(续上页)

```
>>> output = scatter_nd_min(input_x, indices, updates)
>>> print(output)
[[[ 1  1  1  1]
   [ 2  2  2  2]
   [ 3  3  3  3]
   [ 4  4  4  4]]
 [[10 10 10 10]
  [10 10 10 10]
  [10 10 10 10]
  [10 10 10 10]]
 [[ 5  5  5  5]
   [ 6  6  6  6]
   [ 7  7  7  7]
   [ 8  8  8  8]]
 [[10 10 10 10]
  [10 10 10 10]
  [10 10 10 10]
  [10 10 10 10]]]
```

mindspore.ops.ScatterNdMul

class mindspore.ops.ScatterNdMul (use_locking=False)

对张量中的单个值或切片应用稀疏乘法。

使用给定值通过乘法运算和输入索引更新 Parameter 或 Tensor 值。在更新完成后输出 *input_x*，这有利于更加方便地使用更新后的值。

警告： 这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.scatter_nd_mul()`。

参数：

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值: False。

输入：

- **input_x** (Union[Parameter, Tensor]) - 输入参数，数据类型为 Parameter 或 Tensor。
- **indices** (Tensor) - 指定乘法操作的索引，数据类型为 mindspore.int32 或 mindspore.int64。索引的 rank 必须至少为 2，并且 `indices.shape[-1] <= len(shape)`。
- **updates** (Tensor) - 指定与 *input_x* 进行乘法操作的 Tensor，数据类型与 *input_x* 相同，shape 为 `indices.shape[:-1] + x.shape[indices.shape[-1]::]`。

输出：

Tensor，更新后的 *input_x*，shape 和数据类型与 *input_x* 相同。

支持平台：

GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8]), mindspore.float32), name=
↳ "x")
>>> indices = Tensor(np.array([[2], [4], [1], [7]]), mindspore.int32)
>>> updates = Tensor(np.array([6, 7, 8, 9]), mindspore.float32)
>>> scatter_nd_mul = ops.ScatterNdMul()
>>> output = scatter_nd_mul(input_x, indices, updates)
>>> print(output)
[ 1. 16. 18.  4. 35.  6.  7. 72.]
>>> input_x = Parameter(Tensor(np.ones((4, 4, 4)), mindspore.int32))
>>> indices = Tensor(np.array([[0], [2]]), mindspore.int32)
>>> updates = Tensor(np.array([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]]),
↳ mindspore.int32)
>>> scatter_nd_mul = ops.ScatterNdMul()
>>> output = scatter_nd_mul(input_x, indices, updates)
>>> print(output)
[[[1 1 1 1]
  [2 2 2 2]
  [3 3 3 3]
  [4 4 4 4]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]
 [[5 5 5 5]
  [6 6 6 6]
  [7 7 7 7]
  [8 8 8 8]]
 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]

```

mindspore.ops.SearchSorted

class mindspore.ops.**SearchSorted** (*dtype=mstype.int64, right=False*)

返回元素能够插入输入 **tensor** 的位置索引，以维持原 **tensor** 的递增顺序。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.searchsorted()`。

参数：

- **dtype** (`mindspore.dtype`, 可选) - 输出的数据类型。可选值为： `mstype.int32` 和 `mstype.int64`。默认 `mstype.int64`。
- **right** (bool, 可选) - 搜索策略。如果为 `True`，则返回找到的最后一个合适的索引；如果为 `False`，则返回第一个合适的索引。默认 `False`。

输入：

- **sorted_sequence** (Tensor) - 输入 **tensor**。如果未提供 *sorter*，最内层的维度上须为递增的序列。

- **values** (Tensor) - 要插入元素的值。
- **sorter** (Tensor, 可选) - 按 *sorted_sequence* 最内层维度升序排序的索引序列, 与未排序的 *sorted_sequence* 共同使用。CPU 和 GPU 只支持 None。默认 None。

输出:

Tensor

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> sorted_sequence = mindspore.tensor([1, 2, 2, 3, 4, 5, 5], mindspore.float32)
>>> values = mindspore.tensor([2], mindspore.float32)
>>> mindspore.ops.SearchSorted()(sorted_sequence, values)
Tensor(shape=[1], dtype=Int64, value= [1])
```

mindspore.ops.Select

class mindspore.ops.Select

```
prim = ops.Select()
out = prim(condition, input, other)
```

等价于

```
ops.select(condition, input, other)
```

更多详情请查看: [mindspore.ops.select\(\)](#)。

mindspore.ops.Shape

class mindspore.ops.Shape

返回输入 Tensor 的 shape。

获取更多详情请查看[mindspore.ops.shape\(\)](#)。

输入:

- **input_x** (Tensor) - Tensor 的 shape 是 $(x_1, x_2, \dots, x_R)$ 。

输出:

tuple[int], 输出是由多个整数组成的 tuple。 $(x_1, x_2, \dots, x_R)$ 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.ones(shape=[3, 2, 1]), mindspore.float32)
>>> shape = ops.Shape()
```

(下页继续)

(续上页)

```
>>> output = shape(input_x)
>>> print(output)
(3, 2, 1)
```

mindspore.ops.Size

class mindspore.ops.Size

返回一个 `Scalar`，类型为整数，表示输入 `Tensor` 的大小，即 `Tensor` 中元素的总数。

更多参考详见 `mindspore.ops.size()`。

输入：

- **input_x** (`Tensor`) - 输入参数，`shape` 为 $(x_1, x_2, \dots, x_R)$ 。数据类型为 `number`。

输出：

`int`，表示 `input_x` 元素大小的 `Scalar`。它的值为 $size = x_1 * x_2 * \dots * x_R$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[2, 2], [2, 2]]), mindspore.float32)
>>> size = ops.Size()
>>> output = size(input_x)
>>> print(output)
4
```

mindspore.ops.Slice

class mindspore.ops.Slice

根据指定 `shape` 对输入 `Tensor` 进行切片。

更多参考详见 `mindspore.ops.slice()`。

输入：

- **input_x** (`Tensor`) - `Slice` 的输入，任意维度的 `Tensor`。其 `shape` 为 $(N, *)$ ，其中 $*$ 为任意数量的额外维度。
- **begin** (`Union[tuple, list]`) - 切片的起始位置。只支持常量值 (≥ 0)。
- **size** (`Union[tuple, list]`) - 切片的大小。只支持常量值。

输出：

`Tensor`，`shape` 与输入 `size` 相同，数据类型与输入 `input_x` 的相同。

支持平台：

Ascend GPU CPU

样例：


```

>>> from mindspore import Tensor
>>> from mindspore import ops
>>> import numpy as np
>>> data = Tensor(np.array([[[1, 1, 1], [2, 2, 2]],
...                        [[3, 3, 3], [4, 4, 4]],
...                        [[5, 5, 5], [6, 6, 6]]]).astype(np.int32))
>>> slice_op = ops.Slice()
>>> output = slice_op(data, (1, 0, 0), (1, 1, 3))
>>> print(output)
[[[3 3 3]]]
>>> output = slice_op(data, (1, 0, 0), (1, 1, 2))
>>> print(output)
[[[3 3]]]
>>> output = slice_op(data, (1, 0, 0), (1, 1, 1))
>>> print(output)
[[[3]]]
>>> output = slice_op(data, (1, 1, 0), (1, 1, 3))
>>> print(output)
[[[4 4 4]]]
>>> output = slice_op(data, (1, 0, 1), (1, 1, 2))
>>> print(output)
[[[3 3]]]

```

mindspore.ops.Sort

class mindspore.ops.Sort (*axis=-1, descending=False*)

根据指定的轴对输入 Tensor 的元素进行排序，默认为升序排序。

警告： 目前仅支持 float16、uint8、int8、int16、int32、int64 数据类型。如果使用 float32 类型可能导致数据精度损失。

参数：

- **axis** (int, 可选) - 指定排序的轴。默认值：-1，表示指定最后一维。当前 Ascend 后端只支持对最后一维进行排序。
- **descending** (bool, 可选) - 指定排序方式。如果 *descending* 为 True，则根据 value 对元素进行降序排序。默认值：False。

输入：

- **x** (Tensor) - 输入 Tensor。

输出：

- **y1** (Tensor) - Tensor，其值为排序后的值，shape 和数据类型与输入相同。
- **y2** (Tensor) - 输入 Tensor，其元素的索引。数据类型为 int32。

异常：

- **TypeError** - *axis* 不是 int。
- **TypeError** - *descending* 不是 bool。
- **ValueError** - 当 *axis* 取值不在 [-len(x.shape), len(x.shape)) 范围内。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[8, 2, 1], [5, 9, 3], [4, 6, 7]]), mindspore.float16)
>>> sort = ops.Sort()
>>> output = sort(x)
>>> # The output below is based on the Ascend platform.
>>> print(output)
(Tensor(shape=[3, 3], dtype=Float16, value=
[[ 1.0000e+00,  2.0000e+00,  8.0000e+00],
 [ 3.0000e+00,  5.0000e+00,  9.0000e+00],
 [ 4.0000e+00,  6.0000e+00,  7.0000e+00]], Tensor(shape=[3, 3], dtype=Int32, value=
[[2, 1, 0],
 [2, 0, 1],
 [0, 1, 2]]))
```

mindspore.ops.SpaceToBatchND

class mindspore.ops.SpaceToBatchND (block_shape, paddings)

将空间维度划分为对应大小的块，并在批次维度重排张量。

此操作将输入的空间维度 (Space) $[1, \dots, M]$ 划分为形状为 *block_shape* 的块网格，并将这些块在批次维度 (Batch，默认是第零维) 中交错排列。如此，输出在空间维度上的截面是输入在对应空间维度上截面的一个网格，而输出的批次维度的大小为输入的批次维度的大小乘以空间维度分解成块网格的数量。在划分成块之前，输入的空间维度会根据 *paddings* 填充零。

参数:

- **block_shape** (Union[list(int), tuple(int), int]) - 块形状描述空间维度为分割的个数，取值需大于或者等于 1。如果 *block_shape* 为 list 或者 tuple，其长度 *M* 为空间维度的长度。如果 *block_shape* 为整数，那么所有空间维度分割的个数均为 *block_shape*。在 Ascend 后端 *M* 必须为 2。
- **paddings** (Union[tuple, list]) - 空间维度的填充大小。包含 *M* 个 List，每一个 List 包含 2 个整形值，且各值须大于或者等于 0。*paddings[i]* 为对空间维度 *i* 的填充，对应输入 Tensor 的维度 *i+offset*，*offset* 为空间维度在输入 Tensor 维度中的偏移量，其中 *offset=N-M*，*N* 是输入维度数。对空间维度 *i*，*input_shape[i+offset]+paddings[i][0]+paddings[i][1]* 必须能被 *block_shape[i]* 整除。

输入:

- **input_x** (Tensor) - SpaceToBatchND 的输入，Ascend 平台必须为四维。

输出:

Tensor，经过划分排列之后的结果。假设输入的 shape 为 $(n, c_1, \dots, c_k, w_1, \dots, w_M)$ ，且算子的属性为 *block_shape* 和 *paddings*，那么输出的形状为 $(n', c_1, \dots, c_k, w'_1, \dots, w'_M)$ ，其中

$$n' = n * (block\_shape[0] * \dots * block\_shape[M - 1])$$

$$w'_i = (w_i + paddings[i - 1][0] + paddings[i - 1][1]) // block\_shape[i - 1]$$

异常:

- **TypeError** - 如果 *block_shape* 不是 list，tuple 或者 int。
- **TypeError** - 如果 *paddings* 不是 list 或者 tuple。

- **ValueError** - 如果当 *block_shape* 为 list 或 tuple, *block_shape* 不是一维。
- **ValueError** - 如果 Ascend 平台上 *block_shape* 长度不是 2。
- **ValueError** - 如果 *paddings* 的形状不是 (M, 2), 其中 M 为 *block_shape* 的长度。
- **ValueError** - 如果 *block_shape* 的元素不是大于或者等于一的整数。
- **ValueError** - 如果 *paddings* 的元素不是非负的整数。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> block_shape = [2, 2]
>>> paddings = [[0, 0], [0, 0]]
>>> space_to_batch_nd = ops.SpaceToBatchND(block_shape, paddings)
>>> input_x = Tensor(np.array([[[[1, 2], [3, 4]]]]), mindspore.float32)
>>> output = space_to_batch_nd(input_x)
>>> print(output)
[[[1.]]]
[[2.]]]
[[3.]]]
[[4.]]]
```

mindspore.ops.SpaceToDepth

class mindspore.ops.SpaceToDepth (*block_size*)

将空间维度分块, 增加 Tensor 深度。

输出 Tensor 的高度为 *height/block_size*。

输出 Tensor 的宽度为 *weight/block_size*。

输出 Tensor 的深度为 *block_size * block_size * input_depth*。

输入 Tensor 的高度和宽度必须可被 *block_size* 整除。格式为“NCHW” (batch_size, 深度, 高度, 宽度)。

参数:

- **block_size** (int) - 用于划分空间维度的子块的大小。必须 ≥ 2 。

输入:

- **x** (Tensor) - 四维 Tensor。数据类型为 Number。

输出:

四维 Tensor, 数据类型与 *x* 相同, shape: (*N*, (*C_{in}* * *block_size* * 2), *H_{in}*/*block_size*, *W_{in}*/*block_size*)。

异常:

- **TypeError** - *block_size* 不是 int 类型。
- **ValueError** - *block_size* 小于 2。
- **ValueError** - *x* 的维度不为 4。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.random.rand(1, 3, 2, 2), mindspore.float32)
>>> block_size = 2
>>> space_to_depth = ops.SpaceToDepth(block_size)
>>> output = space_to_depth(x)
>>> print(output.shape)
(1, 12, 1, 1)
```

mindspore.ops.SparseGatherV2

class mindspore.ops.SparseGatherV2

基于指定的索引和 axis 返回输入 Tensor 的切片。

输入:

- **input_params** (Tensor) - 被切片的 Tensor。shape: $(x_1, x_2, \dots, x_R)$ 。
- **input_indices** (Tensor) - shape: $(y_1, y_2, \dots, y_S)$ 。指定切片的索引，取值须在 $[0, \text{input\_params.shape[axis]}]$ 范围内。
- **axis** (Union(int, Tensor[int])) - 进行索引的 axis。axis 是 Tensor 的时候，size 必须是 1。

输出:

Tensor, shape: $(z_1, z_2, \dots, z_N)$ 。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_params = Tensor(np.array([[1, 2, 7, 42], [3, 4, 54, 22], [2, 2, 55, 3]]),
↳mindspore.float32)
>>> input_indices = Tensor(np.array([1, 2]), mindspore.int32)
>>> axis = 1
>>> out = ops.SparseGatherV2()(input_params, input_indices, axis)
>>> print(out)
[[2. 7.]
 [4. 54.]
 [2. 55.]]
```

mindspore.ops.Split

class mindspore.ops.**Split** (*axis=0, output_num=1*)

根据指定的轴和分割数量对输入 Tensor 进行分割。

更多参考详见[mindspore.ops.split\(\)](#)。

参数：

- **axis** (int) - 指定分割轴。默认值：0。
- **output_num** (int) - 指定分割数量。其值为正整数。默认值：1。

输入：

- **input_x** (Tensor) - Tensor 的 shape 为 $(x_0, x_1, \dots, x_{R-1})$ ，其中 $R \geq 1$ 。

输出：

tuple[Tensor]，每个输出 Tensor 的 shape 相同，为 $(x_0, x_1, \dots, x_{axis/output\_num}, \dots, x_{R-1})$ 。数据类型与 *input_x* 的相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> split = ops.Split(1, 2)
>>> x = Tensor(np.array([[1, 1, 1, 1], [2, 2, 2, 2]]), mindspore.int32)
>>> print(x)
[[1 1 1 1]
 [2 2 2 2]]
>>> output = split(x)
>>> print(output)
(Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 1],
 [2, 2]]), Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 1],
 [2, 2]]))
>>> split = ops.Split(1, 4)
>>> output = split(x)
>>> print(output)
(Tensor(shape=[2, 1], dtype=Int32, value=
[[1],
 [2]]), Tensor(shape=[2, 1], dtype=Int32, value=
[[1],
 [2]]), Tensor(shape=[2, 1], dtype=Int32, value=
[[1],
 [2]]), Tensor(shape=[2, 1], dtype=Int32, value=
[[1],
 [2]]))
```

mindspore.ops.Squeeze

class mindspore.ops.Squeeze (axis=())

返回删除指定 *axis* 中大小为 1 的维度后的 Tensor。

更多参考详见 `mindspore.ops.squeeze()`。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **axis** (Union[int, tuple(int)]) - 指定待删除 shape 的维度索引，它会删除给定 *axis* 参数中所有大小为 1 的维度。如果指定了维度索引，其数据类型必须为 int32 或 int64。默认值：()。

输入：

- **input** (Tensor) - 用于计算 Squeeze 的输入 Tensor，shape 为 $(x_1, x_2, \dots, x_R)$ 。

输出：

Tensor，shape 为 $(x_1, x_2, \dots, x_S)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input = Tensor(np.ones(shape=[3, 2, 1]), mindspore.float32)
>>> squeeze = ops.Squeeze(2)
>>> output = squeeze(input)
>>> print(output)
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```

mindspore.ops.Stack

class mindspore.ops.Stack (axis=0)

在指定轴上对输入 Tensor 序列进行堆叠。

更多参考详见 `mindspore.ops.stack()`。

参数：

- **axis** (int, 可选) - 指定堆叠运算的轴。取值范围为 $[-(R+1), R+1]$ 。默认值：0。

输入：

- **input_x** (Union[tuple, list]) - 输入多个 Tensor 对象组成的 tuple 或 list，每个 Tensor 具有相同 shape 和数据类型。

输出：

堆叠运算后的 Tensor，数据类型和 *input_x* 的相同。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> import numpy as np
>>> data1 = Tensor(np.array([0, 1]).astype(np.float32))
>>> data2 = Tensor(np.array([2, 3]).astype(np.float32))
>>> stack = ops.Stack()
>>> output = stack([data1, data2])
>>> print(output)
[[0. 1.]
 [2. 3.]]
```

mindspore.ops.StridedSlice

class mindspore.ops.StridedSlice (begin_mask=0, end_mask=0, ellipsis_mask=0, new_axis_mask=0, shrink_axis_mask=0)

```
prim = ops.StridedSlice(begin_mask, end_mask, ellipsis_mask, new_axis_mask, shrink_axis_mask)
out = prim(input_x, begin, end, strides)
```

等价于

```
ops.strided_slice(input_x, begin, end, strides, begin_mask, end_mask, ellipsis_mask, new_
→axis_mask, shrink_axis_mask)
```

更多详情请查看: [mindspore.ops.strided_slice\(\)](#)。

mindspore.ops.TensorScatterAdd

class mindspore.ops.TensorScatterAdd

根据指定的更新值 *updates* 和输入索引 *indices*, 使用加法运算更新输入 Tensor 的值。当同一索引有不同更新值时, 更新的结果将是累积加法的结果。此操作与 [mindspore.ops.ScatterNdAdd](#) 类似, 但更新后的结果是返回一个新的输出 Tensor, 而不是直接更新 *input_x*。

更多参考详见 [mindspore.ops.tensor_scatter_add\(\)](#)。

$$output[indices] = input\_x + update$$

输入:

- **input_x** (Tensor) - 输入 Tensor。*input_x* 的维度必须不小于 *indices.shape[-1]*。
- **indices** (Tensor) - 输入 Tensor 的索引, 数据类型为 int32 或 int64。其 rank 必须至少为 2。
- **updates** (Tensor) - 指定与 *input_x* 相加的 Tensor, 其数据类型与 *input_x* 相同, 并且 shape 应等于 *indices.shape[:-1] + input_x.shape[indices.shape[-1]:]*。

输出:

Tensor, shape 和数据类型与输入 *input_x* 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]), mindspore.float32)
>>> indices = Tensor(np.array([[0, 0], [0, 0]]), mindspore.int32)
>>> updates = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> # Next, demonstrate the approximate operation process of this operator:
>>> # 1, indices[0] = [0, 0], indices[1] = [0, 0]
>>> # 2, And input_x[0, 0] = -0.1
>>> # 3, So input_x[indices] = [-0.1, -0.1]
>>> # 4, Satisfy the above formula: input_x[indices].shape=(2) == updates.shape=(2)
>>> op = ops.TensorScatterAdd()
>>> # 5, Perform the addition operation for the first time:
>>> #     first_input_x = input_x[0][0] + updates[0] = [[0.9, 0.3, 3.6], [0.4, 0.5, -3.2]]
>>> # 6, Perform the addition operation for the second time:
>>> #     second_input_x = input_x[0][0] + updates[1] = [[3.1, 0.3, 3.6], [0.4, 0.5, -3.2]]
>>> output = op(input_x, indices, updates)
>>> print(output)
[[ 3.1  0.3  3.6]
 [ 0.4  0.5 -3.2]]
```

mindspore.ops.TensorScatterDiv

class mindspore.ops.TensorScatterDiv

根据指定的更新值 *updates* 和输入索引 *indices*，进行除法运算更新输入 Tensor 的值。当同一索引有不同更新值时，更新的结果将是累积除法的结果。此操作更新后会返回一个新的输出 Tensor，而不是直接原地更新 *input_x*。

更多参考详见 `mindspore.ops.tensor_scatter_div()`。

$$\text{output}[\text{indices}] = \text{input\_x} \div \text{update}$$

输入：

- **input_x** (Tensor) - 输入 Tensor。*input_x* 的维度必须不小于 `indices.shape[-1]`。
- **indices** (Tensor) - 输入 Tensor 的索引，数据类型为 int32 或 int64。其 rank 必须至少为 2。
- **updates** (Tensor) - 指定与 *input_x* 相除的 Tensor，其数据类型与 *input_x* 相同，并且 shape 应等于 `indices.shape[-1] + input_x.shape[indices.shape[-1] :]`。

输出：

Tensor，shape 和数据类型与输入 *input_x* 相同。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]), mindspore.float32)
>>> indices = Tensor(np.array([[0, 0], [0, 0]]), mindspore.int32)
>>> updates = Tensor(np.array([1.0, 2.0]), mindspore.float32)
>>> # Next, demonstrate the approximate operation process of this operator:
>>> # 1, indices[0] = [0, 0], indices[1] = [0, 0]
```

(下页继续)

(续上页)

```

>>> # 2, And input_x[0, 0] = -0.1
>>> # 3, So input_x[indices] = [-0.1, -0.1]
>>> # 4, Satisfy the above formula: input_x[indices].shape=(2) == updates.shape=(2)
>>> op = ops.TensorScatterDiv()
>>> # 5, Perform the division operation for the first time:
>>> #     first_input_x = input_x[0][0] / updates[0] = [[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]
>>> # 6, Perform the division operation for the second time:
>>> #     second_input_x = input_x[0][0] * updates[1] = [[-0.05, 0.3, 3.6], [0.4, 0.5, -3.
↪2]]
>>> output = op(input_x, indices, updates)
>>> print(output)
[[-0.05  0.3  3.6 ]
 [ 0.4   0.5 -3.2 ]]

```

mindspore.ops.TensorScatterMax

class mindspore.ops.TensorScatterMax

根据指定的更新值 *updates* 和输入索引 *indices*，通过最大值运算将结果赋值到输出 Tensor 中。

更多参考详见 `mindspore.ops.tensor_scatter_max()`。

输入：

- **input_x** (Tensor) - 输入 Tensor。*input_x* 的维度必须不小于 *indices.shape[-1]*。
- **indices** (Tensor) - 输入 Tensor 的索引，数据类型为 int32 或 int64。其 rank 必须至少为 2。
- **updates** (Tensor) - 指定与 *input_x* 取最大值的 Tensor，其数据类型与 *input_x* 相同，并且 shape 应等于 *indices.shape[: -1] + input_x.shape[indices.shape[-1] :]*。

输出：

Tensor, shape 和数据类型与输入 *input_x* 相同。

支持平台：

GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]], mindspore.float32)
>>> indices = Tensor(np.array([[0, 0], [0, 0]]), mindspore.int32)
>>> updates = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> # Next, demonstrate the approximate operation process of this operator:
>>> # 1, indices[0] = [0, 0], indices[1] = [0, 0]
>>> # 2, And input_x[0, 0] = -0.1
>>> # 3, So input_x[indices] = [-0.1, -0.1]
>>> # 4, Satisfy the above formula: input_x[indices].shape=(2) == updates.shape=(2)
>>> op = ops.TensorScatterMax()
>>> # 5, Perform the max operation for the first time:
>>> #     first_input_x = Max(input_x[0][0], updates[0]) = [[1.0, 0.3, 3.6], [0.4, 0.5, -3.
↪2]]
>>> # 6, Perform the max operation for the second time:
>>> #     second_input_x = Max(input_x[0][0], updates[1]) = [[2.2, 0.3, 3.6], [0.4, 0.5, -3.
↪2]]

```

(下页继续)

(续上页)

```
>>> output = op(input_x, indices, updates)
>>> print(output)
[[ 2.2  0.3  3.6]
 [ 0.4  0.5 -3.2]]
```

mindspore.ops.TensorScatterMin

class mindspore.ops.TensorScatterMin

根据指定的更新值 *updates* 和输入索引 *indices*，计算原值与更新值的较小值并更新原值，返回更新后的 Tensor。

更多参考详见 `mindspore.ops.tensor_scatter_min()`。

$$\text{output}[\text{indices}] = \min(\text{input\_x}, \text{update})$$

输入：

- **input_x** (Tensor) - 输入 Tensor。*input_x* 的维度必须不小于 `indices.shape[-1]`。
- **indices** (Tensor) - 输入 Tensor 的索引，数据类型为 int32 或 int64。其 rank 必须至少为 2。
- **updates** (Tensor) - 指定与 *input_x* 取最小值的 Tensor，其数据类型与 *input_x* 相同，并且 shape 应等于 `indices.shape[:-1] + input_x.shape[indices.shape[-1]:]`。

输出：

Tensor，shape 和数据类型与输入 *input_x* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]), mindspore.float32)
>>> indices = Tensor(np.array([[0, 0], [0, 0]]), mindspore.int32)
>>> updates = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> # Next, demonstrate the approximate operation process of this operator:
>>> # 1, indices[0] = [0, 0], indices[1] = [0, 0]
>>> # 2, And input_x[0, 0] = -0.1
>>> # 3, So input_x[indices] = [-0.1, -0.1]
>>> # 4, Satisfy the above formula: input_x[indices].shape=(2) == updates.shape=(2)
>>> op = ops.TensorScatterMin()
>>> # 5, Perform the min operation for the first time:
>>> #     first_input_x = Min(input_x[0][0], updates[0]) = [[-0.1, 0.3, 3.6], [0.4, 0.5, -3.
↪2]]
>>> # 6, Perform the min operation for the second time:
>>> #     second_input_x = Min(input_x[0][0], updates[1]) = [[-0.1, 0.3, 3.6], [0.4, 0.5, -
↪3.2]]
>>> output = op(input_x, indices, updates)
>>> print(output)
[[ -0.1  0.3  3.6]
 [ 0.4  0.5 -3.2]]
```

mindspore.ops.TensorScatterMul

class mindspore.ops.TensorScatterMul

根据指定的更新值 *updates* 和输入索引 *indices*，使用乘法运算更新输入 Tensor 的值。当同一索引有不同更新值时，更新的结果将是累积乘法的结果。此操作与 *mindspore.ops.ScatterNdMul* 类似，但更新后的结果是返回一个新的输出 Tensor，而不是直接更新 *input_x*。

$$\text{output}[\text{indices}] = \text{input\_x} \times \text{update}$$

更多参考相见 *mindspore.ops.tensor_scatter_mul()*。

输入：

- **input_x** (Tensor) - 输入 Tensor。*input_x* 的维度必须大于等于 *indices.shape[-1]*。
- **indices** (Tensor) - *input_x* 执行 scatter 操作的目标索引，数据类型为 int32 或 int64，rank 必须大于等于 2。
- **updates** (Tensor) - 指定与 *input_x* 相乘操作的 Tensor，其数据类型与 *input_x* 相同。并且 shape 应等于 *indices.shape[:-1] + input_x.shape[indices.shape[-1]:]*。

输出：

Tensor，shape 和数据类型与输入 *input_x* 相同。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]], mindspore.float32)
>>> indices = Tensor(np.array([[0, 0], [0, 0]]), mindspore.int32)
>>> updates = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> # Next, demonstrate the approximate operation process of this operator:
>>> # 1, indices[0] = [0, 0], indices[1] = [0, 0]
>>> # 2, And input_x[0, 0] = -0.1
>>> # 3, So input_x[indices] = [-0.1, -0.1]
>>> # 4, Satisfy the above formula: input_x[indices].shape=(2) == updates.shape=(2)
>>> op = ops.TensorScatterMul()
>>> # 5, Perform the multiply operation for the first time:
>>> #     first_input_x = input_x[0][0] * updates[0] = [[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]
>>> # 6, Perform the multiply operation for the second time:
>>> #     second_input_x = input_x[0][0] * updates[1] = [[-0.22, 0.3, 3.6], [0.4, 0.5, -3.
↪2]]
>>> output = op(input_x, indices, updates)
>>> print(output)
[[-0.22  0.3   3.6  ]
 [ 0.4   0.5  -3.2  ]]
```

mindspore.ops.TensorScatterSub

class mindspore.ops.TensorScatterSub

根据指定的更新值 *input_x* 和输入索引 *indices*, 进行减法运算更新输入 Tensor 的值。当同一索引有不同更新值时, 更新的结果将是累积减法的结果。此操作与 *mindspore.ops.ScatterNdSub* 类似, 只是更新后的结果是通过算子 *output* 返回, 而不是直接原地更新 *input*。

```
# 遍历所有索引
for i in range(indices.shape[0]):
    for j in range(indices.shape[1]):
        ...
        for k in range(indices.shape[-2]): # 最后一维是坐标维度
            # 获取当前索引组合
            index_tuple = (i, j, ..., k)
            # 获取目标位置
            target_index = indices[index_tuple]
            # 获取对应更新值
            update_value = updates[index_tuple]
            # 执行减法操作
            output[target_index] -= update_value
```

更多参考详见 *mindspore.ops.tensor_scatter_sub()*。

输入:

- **input_x** (Tensor) - 输入 Tensor。*input_x* 的维度必须大于等于 *indices.shape[-1]*。
- **indices** (Tensor) - 输入 Tensor 的索引, 数据类型为 int32 或 int64, rank 必须大于等于 2。
- **updates** (Tensor) - 指定与 *input_x* 相减操作的 Tensor, 其数据类型与 *input_x* 相同。并且 shape 应等于 *indices.shape[:-1] + input_x.shape[indices.shape[-1]:]*。

输出:

Tensor, shape 和数据类型与输入 *input_x* 相同。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[[-0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]], mindspore.float32))
>>> indices = Tensor(np.array([[0, 0], [0, 0]], mindspore.int32))
>>> updates = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> # Next, demonstrate the approximate operation process of this operator:
>>> # 1, indices[0] = [0, 0], indices[1] = [0, 0]
>>> # 2, And input_x[0, 0] = -0.1
>>> # 3, So input_x[indices] = [-0.1, -0.1]
>>> # 4, Satisfy the above formula: input_x[indices].shape=(2) == updates.shape=(2)
>>> op = ops.TensorScatterSub()
>>> # 5, Perform the subtract operation for the first time:
>>> #     first_input_x = input_x[0][0] - updates[0] = [[-1.1, 0.3, 3.6], [0.4, 0.5, -3.2]]
>>> # 6, Perform the subtract operation for the second time:
>>> #     second_input_x = input_x[0][0] - updates[1] = [[-3.3, 0.3, 3.6], [0.4, 0.5, -3.2]]
>>> output = op(input_x, indices, updates)
>>> print(output)
```

(下页继续)

(续上页)

```
[[-3.3000002  0.3      3.6      ]
 [ 0.4        0.5     -3.2     ]]
```

mindspore.ops.TensorScatterUpdate

class mindspore.ops.TensorScatterUpdate

根据指定的更新值 *update* 和输入索引 *indices*，通过更新操作更新输出 Tensor 的值。此操作几乎等同于使用 *mindspore.ops.ScatterNdUpdate*，但更新后的结果是返回一个新的输出 Tensor，而不是直接更新 *input_x*。

indices 的 rank 大于等于 2，最后一个轴表示每个索引向量的深度。对于每个索引向量，*update* 中必须有相应的值。如果每个索引 Tensor 的深度与 *input_x* 的 rank 匹配，则每个索引向量对应于 *input_x* 中的 Scalar，并且每次更新都会更新一个 Scalar。如果每个索引 Tensor 的深度小于 *input_x* 的 rank，则每个索引向量对应于 *input_x* 中的切片，并且每次更新都会更新一个切片。

更新的顺序是不确定的，这意味着如果 *indices* 中有多个索引向量对应于同一位置，则输出中该位置值是不确定的。

$$output[indices] = update$$

输入：

- **input_x** (Tensor) - TensorScatterUpdate 的输入，任意维度的 Tensor。其数据类型为数值型。*input_x* 的维度必须不小于 *indices.shape[-1]*。其 shape 为 $(N,*)$ ，其中 * 为任意数量的额外维度。
- **indices** (Tensor) - *input_x* 执行 scatter 操作的目标索引，数据类型为 int32 或 int64，rank 必须大于等于 2。
- **update** (Tensor) - 指定与 *input_x* 做更新操作的 Tensor，其数据类型与 *input_x* 相同。并且 *update.shape = indices.shape[: -1] + input_x.shape[indices.shape[-1] :]*。

输出：

Tensor，shape 和数据类型与输入 *input_x* 相同。

异常：

- **TypeError** - *indices* 的数据类型不满足 int32 或 int64。
- **ValueError** - *input_x* 的 rank 小于 *indices.shape* 的最后一维。
- **ValueError** - *input_x* 的值与输入 *indices* 不匹配。
- **RuntimeError** - *indices* 中的值超出了 *input_x* 的索引范围。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.array([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]]), mindspore.float32)
>>> indices = Tensor(np.array([[0, 0], [1, 1]]), mindspore.int32)
>>> update = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> op = ops.TensorScatterUpdate()
>>> output = op(input_x, indices, update)
>>> print(output)
[[ 1.   0.3  3.6]
 [ 0.4  2.2 -3.2]]
```

mindspore.ops.TensorShape

class mindspore.ops.TensorShape

返回输入 Tensor 的 Shape。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> input_x = Tensor(np.ones(shape=[3, 2, 1]), mindspore.float32)
>>> output = ops.TensorShape()(input_x)
>>> print(output)
[3 2 1]
```

mindspore.ops.Tile

class mindspore.ops.Tile

按照给定的次数复制输入 Tensor。

更多参考详见 `mindspore.ops.tile()`。

输入：

- **input** (Tensor) - 需要被复制的 Tensor，设其 shape 为 $(x_1, x_2, \dots, x_S)$ 。
- **dims** (tuple[int]) - 指定复制次数的参数，参数类型为 tuple，数据类型为整数，如 $(y_1, y_2, \dots, y_S)$ ，只支持常量值。

说明：在 Ascend 平台上，*dims* 参数的个数不大于 8，当前不支持超过 4 个维度同时做 repeat 的场景。

输出：

Tensor，具有与 *input* 相同的数据类型。假设 *dims* 的长度为 *d*，*input* 的维度为 *input.dim*，*input* 的 shape 为 $(x_1, x_2, \dots, x_S)$ 。

- 如果 $input.dim = d$ ，将其相应位置的 shape 相乘，输出的 shape 为 $(x_1 * y_1, x_2 * y_2, \dots, x_S * y_S)$ 。
- 如果 $input.dim < d$ ，在 *input* 的 shape 的前面填充 1，直到它们的长度一致。例如将 *input* 的 shape 设置为 $(1, \dots, x_1, x_2, \dots, x_S)$ ，然后可以将其相应位置的 shape 相乘，输出的 shape 为 $(1 * y_1, \dots, x_R * y_R, x_S * y_S)$ 。
- 如果 $input.dim > d$ ，在 *dims* 的前面填充 1，直到它们的长度一致。例如将 *dims* 设置为 $(1, \dots, y_1, y_2, \dots, y_S)$ ，然后可以将其相应位置的 shape 相乘，输出的 shape 为 $(x_1 * 1, \dots, x_R * y_R, x_S * y_S)$ 。

异常：

- **TypeError** - *dims* 不是 tuple 或者其元素并非全部是 int。
- **ValueError** - *dims* 的元素并非全部大于或等于 0。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> tile = ops.Tile()
>>> input = Tensor(np.array([[1, 2], [3, 4]]), mindspore.float32)
>>> dims = (2, 3)
>>> output = tile(input, dims)
>>> print(output)
[[1.  2.  1.  2.  1.  2.]
 [3.  4.  3.  4.  3.  4.]
 [1.  2.  1.  2.  1.  2.]
 [3.  4.  3.  4.  3.  4.]]
>>> dims = (2, 3, 2)
>>> output = tile(input, dims)
>>> print(output)
[[[1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]]
 [[1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]
 [1.  2.  1.  2.]
 [3.  4.  3.  4.]]]]

```

mindspore.ops.Trace

class mindspore.ops.Trace

```

prim = ops.Trace()
out = prim(input)

```

等价于

```
ops.trace(input)
```

更多详情请查看: [mindspore.ops.trace\(\)](#)。

mindspore.ops.Transpose

class mindspore.ops.Transpose

```

prim = ops.Transpose()
out = prim(input, input_perm)

```

等价于

```
ops.transpose(input, input_perm)
```

更多详情请查看: [mindspore.ops.transpose\(\)](#)。

mindspore.ops.Tril**class** mindspore.ops.Tril(*diagonal=0*)

返回单个或一批二维矩阵下三角形部分，其他位置的元素将被置零。矩阵的下三角形部分定义为对角线本身和对角线以下的元素。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **diagonal** (int, 可选) - 指定对角线位置，默认值：0，指定主对角线。

输入：

- **x** (Tensor) - 输入 Tensor。shape 为 $(*, M, N)$ ，其中 $*$ 为任意数量的额外维度。

输出：

Tensor，其数据类型和 shape 维度与输入 x 相同。

异常：

- **TypeError** - 如果 x 不是 Tensor。
- **TypeError** - 如果 *diagonal* 不是 int 类型。
- **ValueError** - 如果 x 的秩小于 2。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> x = Tensor(np.array([[ 1,  2,  3,  4],
...                      [ 5,  6,  7,  8],
...                      [10, 11, 12, 13],
...                      [14, 15, 16, 17]]))
>>> tril = ops.Tril()
>>> result = tril(x)
>>> print(result)
[[ 1  0  0  0]
 [ 5  6  0  0]
 [10 11 12  0]
 [14 15 16 17]]
>>> x = Tensor(np.array([[ 1,  2,  3,  4],
...                      [ 5,  6,  7,  8],
...                      [10, 11, 12, 13],
...                      [14, 15, 16, 17]]))
>>> tril = ops.Tril(diagonal=1)
>>> result = tril(x)
>>> print(result)
[[ 1  2  0  0]
 [ 5  6  7  0]
 [10 11 12 13]
 [14 15 16 17]]
>>> x = Tensor(np.array([[ 1,  2,  3,  4],
```

(下页继续)

(续上页)

```

...           [ 5,  6,  7,  8],
...           [10, 11, 12, 13],
...           [14, 15, 16, 17]])
>>> tril = ops.Tril(diagonal=-1)
>>> result = tril(x)
>>> print(result)
[[ 0  0  0  0]
 [ 5  0  0  0]
 [10 11  0  0]
 [14 15 16  0]]

```

mindspore.ops.TrilIndices

class mindspore.ops.TrilIndices (row, col, offset=0, dtype=mstype.int32)

计算二维矩阵的下三角元素的索引，并将它们作为一个 Tensor 返回。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.tril_indices()`。

参数：

- **row** (int) - 二维矩阵的行数。
- **col** (int) - 二维矩阵的列数。
- **offset** (int, 可选) - 对角线偏移量。默认值：0。
- **dtype** (`mindspore.dtype`, 可选) - 指定输出 Tensor 的数据类型，支持的数据类型为 `mstype.int32` 和 `mstype.int64`，默认值：`mstype.int32`。

输出：

- **y** (Tensor) - 矩阵的下三角元素的索引。数据类型由 `dtype` 指定，`shape` 为 `(2, tril_size)`，其中，`tril_size` 为下三角矩阵的元素总数。

支持平台：

Ascend GPU CPU

样例：

```

>>> from mindspore import ops
>>> from mindspore import dtype as mstype
>>> net = ops.TrilIndices(4, 3, -1, mstype.int64)
>>> output = net()
>>> print(output)
[[1 2 2 3 3 3]
 [0 0 1 0 1 2]]
>>> print(output.dtype)
Int64

```

mindspore.ops.Triu

class mindspore.ops.Triu (diagonal=0)

```
prim = ops.Triu(diagonal)
out = prim(input)
```

等价于

```
ops.triu(input, diagonal)
```

更多详情请查看: [mindspore.ops.triu\(\)](#)。

mindspore.ops.TriuIndices

class mindspore.ops.TriuIndices (row, col, offset=0, dtype=mstype.int32)

计算 $row * col$ 行列矩阵的上三角元素的索引, 并将它们作为一个 $2 \times N$ 的 Tensor 返回。

警告: 这是一个实验性 API, 后续可能修改或删除。

更多参考详见[mindspore.ops.triu_indices\(\)](#)。

参数:

- **row** (int) - 2-D 矩阵的行数。
- **col** (int) - 2-D 矩阵的列数。
- **offset** (int, 可选) - 对角线偏移量。默认值: 0。
- **dtype** ([mindspore.dtype](#), 可选) - 指定输出 Tensor 数据类型, 支持的数据类型为 `mstype.int32` 和 `mstype.int64`, 默认值: `mstype.int32`。

输出:

- **y** (Tensor) - 矩阵的上三角形部分的索引。数据类型由 `dtype` 指定, `shape` 为 $(2, tril\_size)$, 其中, `tril_size` 为上三角矩阵的元素总数。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import ops
>>> from mindspore import dtype as mstype
>>> net = ops.TriuIndices(5, 4, 2, mstype.int64)
>>> output = net()
>>> print(output)
[[0 0 1]
 [2 3 3]]
>>> print(output.dtype)
Int64
```

mindspore.ops.Unique

class mindspore.ops.Unique

返回输入 Tensor 的唯一元素以及其对应的每个值的索引。

输出包含 Tensor *y* 和 Tensor *idx*，格式形如 (*y*, *idx*)。Tensor *y* 和 Tensor *idx* 的 shape 在大多数情况下是不同的，因为 Tensor *y* 可能存在重复，并且 Tensor *idx* 的 shape 与输入保持一致。

要获得 *idx* 和 *y* 之间相同的 shape，请参考 `mindspore.ops.UniqueWithPad`。

输入：

- **input_x** (Tensor) - 输入 Tensor。shape 为 (*N*, *), 其中 * 表示，任意数量的附加维度。

输出：

Tuple，形如 (*y*, *idx*) 的 Tensor 对象，*y* 与 *input_x* 的数据类型相同，记录的是 *input_x* 中的唯一元素。*idx* 是一个 Tensor，记录的是输入 *input_x* 元素相对应的索引。

异常：

- **TypeError** - 如果 *input_x* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, nn
>>> input_x = Tensor(np.array([1, 2, 5, 2]), mindspore.int32)
>>> output = ops.Unique()(input_x)
>>> print(output)
(Tensor(shape=[3], dtype=Int32, value= [1, 2, 5]), Tensor(shape=[4], dtype=Int32, value= [0, 1, 2, 1]))
>>> y = output[0]
>>> print(y)
[1 2 5]
>>> idx = output[1]
>>> print(idx)
[0 1 2 1]
>>> # As can be seen from the above, y and idx shape
>>> # note that for GPU, this operator must be wrapped inside a model, and executed in graph mode.
>>> class UniqueNet(nn.Cell):
...     def __init__(self):
...         super(UniqueNet, self).__init__()
...         self.unique_op = ops.Unique()
...
...     def construct(self, x):
...         output, indices = self.unique_op(x)
...         return output, indices
...
>>> input_x = Tensor(np.array([1, 2, 5, 2]), mindspore.int32)
>>> net = UniqueNet()
>>> output = net(input_x)
>>> print(output)
(Tensor(shape=[3], dtype=Int32, value= [1, 2, 5]), Tensor(shape=[4], dtype=Int32, value= [0, 1, 2, 1]))
```

mindspore.ops.UniqueConsecutive

class mindspore.ops.UniqueConsecutive (*return_inverse=False, return_counts=False, dim=None*)

对输入 Tensor 中连续且重复的元素去重。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.unique_consecutive()`。

参数：

- **return_inverse** (bool, 可选) - 是否返回每个输入中元素映射到输出中位置的索引。默认值：False。
- **return_counts** (bool, 可选) - 是否返回每个去重元素在输入所在的连续序列的计数。默认值：False。
- **dim** (int, 可选) - 维度。如果为 None，则对输入进行展平操作。如果指定，必须是 int32 或 int64 类型。默认值：None。

输入：

- **x** (Tensor) - 输入 Tensor。

输出：

Tensor 或包含 Tensor 对象的元组 (*output*、*idx*、*counts*)。

- *output* 为去重后的输出，与 *x* 具有相同的数据类型。
- 如果 *return_inverse* 为 True，则返回 Tensor *idx*，*shape* 与 *input* 相同，表示每个输入中元素映射到输出中位置的索引。
- 如果 *return_counts* 为 True，则返回 Tensor *counts*，表示每个去重元素在输入中所在的连续序列的计数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> x = Tensor(np.array([1, 1, 2, 2, 3, 1, 1, 2]), mstype.int32)
>>> unique_consecutive = ops.UniqueConsecutive(True, True, None)
>>> output, idx, counts = unique_consecutive(x)
>>> print(output)
[1 2 3 1 2]
>>> print(idx)
[0 0 1 1 2 3 3 4]
>>> print(counts)
[2 2 1 2 1]
```

`mindspore.ops.UniqueWithPad`

`class mindspore.ops.UniqueWithPad`

`ops.UniqueWithPad` 从 2.4 版本开始已被弃用，并将在未来版本中被移除。请组合使用 `mindspore.ops.unique()` 和 `mindspore.ops.pad()` 实现同样的功能。

支持平台：

Deprecated

`mindspore.ops.UnsortedSegmentMax`

`class mindspore.ops.UnsortedSegmentMax`

沿分段计算输入 Tensor 的最大值。

更多参考详见 `mindspore.ops.unsorted_segment_max()`。

输入：

- **input_x** (Tensor) - shape: $(x_1, x_2, \dots, x_R)$ 。数据类型支持 float16、float32 或 int32。
- **segment_ids** (Tensor) - 用于指示每个元素所属段的标签，将 shape 设置为 $(x_1, x_2, \dots, x_N)$ ，其中 $0 < N \leq R$ 。数据类型支持 int32。
- **num_segments** (Union[int, Tensor]) - 分段数量 z ，可以为 int 或零维的 Tensor。

输出：

Tensor, shape 为: $(z, x_{N+1}, \dots, x_R)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> # case 1: Only have two num_segments, where is 0 and 1, and segment_ids=[0, 1, 1]
>>> # num_segments = 2 indicates that there are two types of segment_id,
>>> # the first number '0' in [0, 1, 1] indicates input_x[0],
>>> # the second number '1' in [0, 1, 1] indicates input_x[1],
>>> # the third number '1' in [0, 1, 1] indicates input_x[2],
>>> # input_x[0], which is [1, 2, 3] will not be compared to other segment_id.
>>> # Only the same segment_id will be compared.
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> import numpy as np
>>> input_x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [4, 2, 1]]).astype(np.float32))
>>> segment_ids = Tensor(np.array([0, 1, 1]).astype(np.int32))
>>> num_segments = 2
>>> unsorted_segment_max = ops.UnsortedSegmentMax()
>>> output = unsorted_segment_max(input_x, segment_ids, num_segments)
>>> print(output)
[[1. 2. 3.]
 [4. 5. 6.]]
>>>
>>> # case 2: The segment_ids=[0, 0, 1, 1].
>>> # [1, 2, 3] will compare with [4, 2, 0],
>>> # and [4, 5, 6] will compare with [4, 2, 1].
>>> input_x = Tensor(np.array([[1, 2, 3], [4, 2, 0], [4, 5, 6], [4, 2, 1]]).astype(np.
↪float32))
```

(下页继续)

(续上页)

```

>>> segment_ids = Tensor(np.array([0, 0, 1, 1]).astype(np.int32))
>>> num_segments = 2
>>> unsorted_segment_max = ops.UnsortedSegmentMax()
>>> output = unsorted_segment_max(input_x, segment_ids, num_segments)
>>> print(input_x.shape)
(4, 3)
>>> print(output)
[[4. 2. 3.]
 [4. 5. 6.]]
>>> # case 3: If the input_x have three dimensions even more, what will happen?
>>> # The shape of input_x is (2, 4, 3),
>>> # and the length of segment_ids should be the same as the first dimension of input_x.
>>> # Because the segment_ids are different, input_x[0] will not be compared to input_x[1].
>>> input_x = Tensor(np.array([[ [1, 2, 3], [4, 2, 0], [4, 5, 6], [4, 2, 1]],
...                             [ [1, 2, 3], [4, 2, 0], [4, 5, 6], [4, 2, 1]]]).astype(np.
↳ float32))
>>> segment_ids = Tensor(np.array([0, 1]).astype(np.int32))
>>> num_segments = 2
>>> unsorted_segment_max = ops.UnsortedSegmentMax()
>>> output = unsorted_segment_max(input_x, segment_ids, num_segments)
>>> print(input_x.shape)
(2, 4, 3)
>>> print(output)
[[ [1. 2. 3.]
   [4. 2. 0.]
   [4. 5. 6.]
   [4. 2. 1.]]
 [ [1. 2. 3.]
   [4. 2. 0.]
   [4. 5. 6.]
   [4. 2. 1.]]]
>>> # case 4: It has the same input with the 3rd case.
>>> # Because num_segments is equal to 2, there are two segment_ids, but currently only one_
↳ 0 is used.
>>> # the segment_id i is absent in the segment_ids, then output[i] will be filled with
>>> # the smallest possible value of the input_x's type.
>>> segment_ids = Tensor(np.array([0, 0]).astype(np.int32))
>>> output = unsorted_segment_max(input_x, segment_ids, num_segments)
>>> print(output)
[[ [ 1.0000000e+00  2.0000000e+00  3.0000000e+00]
   [ 4.0000000e+00  2.0000000e+00  0.0000000e+00]
   [ 4.0000000e+00  5.0000000e+00  6.0000000e+00]
   [ 4.0000000e+00  2.0000000e+00  1.0000000e+00]]
 [[-3.4028235e+38 -3.4028235e+38 -3.4028235e+38]
  [-3.4028235e+38 -3.4028235e+38 -3.4028235e+38]
  [-3.4028235e+38 -3.4028235e+38 -3.4028235e+38]
  [-3.4028235e+38 -3.4028235e+38 -3.4028235e+38]]]

```

mindspore.ops.UnsortedSegmentMin

class mindspore.ops.UnsortedSegmentMin

沿分段计算输入 Tensor 的最小值。

更多参考详见 `mindspore.ops.unsorted_segment_min()`。

输入：

- **input_x** (Tensor) - shape: $(x_1, x_2, \dots, x_R)$ 。数据类型支持 float16、float32 或 int32。
- **segment_ids** (Tensor) - 用于指示每个元素所属段的标签，将 shape 设置为 $(x_1, x_2, \dots, x_N)$ ，其中 $0 < N \leq R$ 。数据类型支持 int32。
- **num_segments** (Union[int, Tensor]) - 分段数量 z ，可以为 int 或零维的 Tensor。

输出：

Tensor, shape 为: $(z, x_{N+1}, \dots, x_R)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> import numpy as np
>>> input_x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [4, 2, 1]]).astype(np.float32))
>>> segment_ids = Tensor(np.array([0, 1, 1]).astype(np.int32))
>>> num_segments = 2
>>> unsorted_segment_min = ops.UnsortedSegmentMin()
>>> output = unsorted_segment_min(input_x, segment_ids, num_segments)
>>> print(output)
[[1. 2. 3.]
 [4. 2. 1.]]
```

mindspore.ops.UnsortedSegmentProd

class mindspore.ops.UnsortedSegmentProd

沿分段计算输入 Tensor 元素的乘积。

更多参考详见 `mindspore.ops.unsorted_segment_prod()`。

输入：

- **input_x** (Tensor) - shape: $(x_1, x_2, \dots, x_R)$ 。数据类型支持 float16、float32 或 int32。
- **segment_ids** (Tensor) - 用于指示每个元素所属段的标签，将 shape 设置为 $(x_1, x_2, \dots, x_N)$ ，其中 $0 < N \leq R$ 。数据类型支持 int32。
- **num_segments** (Union[int, Tensor]) - 分段数量 z ，可以为 int 或零维的 Tensor。

输出：

Tensor, shape 为: $(z, x_{N+1}, \dots, x_R)$ 。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> import numpy as np
>>> input_x = Tensor(np.array([[1, 2, 3], [4, 5, 6], [4, 2, 1]]).astype(np.float32))
>>> segment_ids = Tensor(np.array([0, 1, 0]).astype(np.int32))
>>> num_segments = 2
>>> unsorted_segment_prod = ops.UnsortedSegmentProd()
>>> output = unsorted_segment_prod(input_x, segment_ids, num_segments)
>>> print(output)
[[4. 4. 3.]
 [4. 5. 6.]]
```

mindspore.ops.UnsortedSegmentSum

class mindspore.ops.UnsortedSegmentSum

```
prim = ops.UnsortedSegmentSum()
out = prim(input_x, segment_ids, num_segments)
```

等价于

```
ops.unsorted_segment_sum(input_x, segment_ids, num_segments)
```

更多详情请查看: [mindspore.ops.unsorted_segment_sum\(\)](#)。

mindspore.ops.Unstack

class mindspore.ops.Unstack (axis=0, num=None)

根据指定轴对输入矩阵进行分解, 与 Stack 函数操作相反。若输入 Tensor 的 rank 为 R , 则输出 Tensor 的 rank 为 $(R-1)$ 。

更多参考详见[mindspore.ops.unstack\(\)](#)。

参数:

- **axis** (int, 可选) - 指定矩阵分解的轴。取值范围为 $[-R, R)$, 默认值: 0。
- **num** (Union[None, int], 可选) - 输出 Tensor 的个数。如果设置为 None, 则根据 input_x 和 axis 自动推导输出个数。默认值: None。

输入:

- **input_x** (Tensor) - 输入 Tensor, 其 shape 为 $(x_1, x_2, \dots, x_R)$, 其 rank 必须大于 0。

输出:

Tensor 对象组成的 tuple。每个 Tensor 对象的 shape 相同。给定一个 shape 为 $(x_1, x_2, \dots, x_R)$ 的 Tensor, 如果存在 $0 \leq \text{axis}$, 则输出 Tensor 的 shape 为 $(x_1, x_2, \dots, x_{\text{axis}}, x_{\text{axis}+2}, \dots, x_R)$ 。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> unstack = ops.Unstack()
```

(下页继续)

(续上页)

```
>>> input_x = Tensor(np.array([[1, 1, 1, 1], [2, 2, 2, 2]]))
>>> output = unstack(input_x)
>>> print(output)
(Tensor(shape=[4], dtype=Int64, value= [1, 1, 1, 1]), Tensor(shape=[4], dtype=Int64, value=
↳ [2, 2, 2, 2]))
```

20.4.4 类型转换

接口名	概述	支持平台
<code>mindspore.ops.ScalarCast</code>	<i>ops.ScalarCast</i> 从 2.3 版本开始已被弃用，并将在未来版本中被移除，建议使用 <i>int(x)</i> 或 <i>float(x)</i> 代替。	弃用
<code>mindspore.ops.ScalarToTensor</code>	将 <i>Scalar</i> 转换为指定数据类型的 <i>Tensor</i> 。	Ascend GPU CPU
<code>mindspore.ops.TupleToArray</code>	将 <i>tuple</i> 转换为 <i>Tensor</i> 。	Ascend GPU CPU

mindspore.ops.ScalarCast

class mindspore.ops.ScalarCast

ops.ScalarCast 从 2.3 版本开始已被弃用，并将在未来版本中被移除，建议使用 *int(x)* 或 *float(x)* 代替。

支持平台：

Deprecated

样例：

```
>>> import mindspore
>>> from mindspore import ops
>>> scalar_cast = ops.ScalarCast()
>>> output = scalar_cast(255.0, mindspore.int64)
>>> print(output)
255
```

mindspore.ops.ScalarToTensor

class mindspore.ops.ScalarToTensor

将 *Scalar* 转换为指定数据类型的 *Tensor*。

更多参考详见`mindspore.ops.scalar_to_tensor()`。

输入：

- **input_x** (Union[int, float]) - 输入是 *Scalar*。只能是常量值。
- **dtype** (mindspore.dtype) - 指定输出的数据类型。只能是常量值。默认值：mindspore.float32。

输出：

Tensor，零维 Tensor，其值和输入一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import ops
>>> op = ops.ScalarToTensor()
>>> data = 1
>>> output = op(data, mindspore.float32)
>>> print(output)
1.0
```

mindspore.ops.TupleToArray

class mindspore.ops.TupleToArray

将 tuple 转换为 Tensor。
更多参考详见mindspore.ops.tuple_to_array()。

输入：
• input_x (tuple) - 数值型组成的 tuple。其元素具有相同的类型。shape 为 (N,)。

输出：
Tensor。如果输入 tuple 包含 N 个数值型元素，则输出 Tensor 的 shape 为 (N,)。

支持平台：
Ascend GPU CPU

样例：

```
>>> from mindspore import ops
>>> input_x = (1,2,3)
>>> print(type(input_x))
<class 'tuple'>
>>> output = ops.TupleToArray()(input_x)
>>> print(type(output))
<class 'mindspore.common.tensor.Tensor'>
>>> print(output)
[1 2 3]
```

20.5 Parameter 操作算子

接口名	概述	支持平台
mindspore.ops.Assign	更多详情请查看： mindspore.ops.assign() 。	Ascend GPU CPU
mindspore.ops.AssignAdd	更多详情请查看： mindspore.ops.assign_add() 。	Ascend GPU CPU
mindspore.ops.AssignSub	更多详情请查看： mindspore.ops.assign_sub() 。	Ascend GPU CPU
mindspore.ops.ScatterAdd	根据指定更新值和输入索引通过加法运算更新输入数据的值。	Ascend GPU CPU
mindspore.ops.ScatterDiv	通过除法操作更新输入张量的值。	Ascend GPU CPU
mindspore.ops.ScatterMax	通过最大操作更新输入张量的值。	Ascend GPU CPU
mindspore.ops.ScatterMin	通过最小操作更新输入张量的值。	Ascend GPU CPU

下页继续

表 19 – 续上页

<code>mindspore.ops.ScatterMul</code>	根据指定更新值和输入索引通过乘法运算更新输入数据的值。	Ascend GPU CPU
<code>mindspore.ops.ScatterNdAdd</code>	使用给定值通过加法运算和输入索引更新 Tensor 值。	Ascend GPU CPU
<code>mindspore.ops.ScatterNdSub</code>	使用给定值通过减法运算和输入索引更新 Tensor 值。	Ascend GPU CPU
<code>mindspore.ops.ScatterNdUpdate</code>	使用给定值以及输入索引更新输入数据的值。	Ascend GPU CPU
<code>mindspore.ops.ScatterNonAliasingAdd</code>	ScatterNonAliasingAdd 接口已废弃，请使用接口 <code>mindspore.ops.TensorScatterAdd</code> 替换，废弃版本 2.1。	弃用
<code>mindspore.ops.ScatterSub</code>	使用给定更新值通过减法操作和输入索引来更新 Tensor 值。	Ascend GPU CPU
<code>mindspore.ops.ScatterUpdate</code>	使用给定的更新值和输入索引更新输入 Tensor 的值。	Ascend GPU CPU

20.5.1 mindspore.ops.Assign

class mindspore.ops.Assign

```
prim = ops.Assign()
out = prim(variable, value)
```

等价于

```
ops.assign(variable, value)
```

更多详情请查看：`mindspore.ops.assign()`。

20.5.2 mindspore.ops.AssignAdd

class mindspore.ops.AssignAdd

```
prim = ops.AssignAdd()
out = prim(variable, value)
```

等价于

```
ops.assign_add(variable, value)
```

更多详情请查看：`mindspore.ops.assign_add()`。

20.5.3 mindspore.ops.AssignSub

class mindspore.ops.AssignSub

```
prim = ops.AssignSub()
out = prim(variable, value)
```

等价于

```
ops.assign_sub(variable, value)
```

更多详情请查看: `mindspore.ops.assign_sub()`。

20.5.4 mindspore.ops.ScatterAdd

class mindspore.ops.ScatterAdd(*use_locking=False*)

根据指定更新值和输入索引通过加法运算更新输入数据的值。

对于 `indices.shape` 的每个 $i, \dots, j$:

$$\text{input\_x}[\text{indices}[i, \dots, j], :] += \text{updates}[i, \dots, j, :]$$

输入的 `input_x` 和 `updates` 遵循隐式类型转换规则, 以确保数据类型一致。如果数据类型不同, 则低精度数据类型将转换为高精度的数据类型。当参数的数据类型需要转换时, 则会抛出 `RuntimeError` 异常。

说明: 这是一个运行即更新的算子。因此, `input_x` 在运算完成后即更新。

参数:

- **use_locking** (bool, 可选) - 是否启用锁保护。如果为 `True`, 则 `input_x` 将受到锁的保护。否则计算结果是未定义的。默认值: `False`。

输入:

- **input_x** (Union[Parameter, Tensor]) - ScatterAdd 的输入, 数据类型为 `Parameter` 或 `Tensor`。
- **indices** (Tensor) - 指定相加操作的索引, 数据类型为 `mindspore.int32` 或者 `mindspore.int64`。
- **updates** (Tensor) - 指定与 `input_x` 相加操作的 `Tensor`, 数据类型与 `input_x` 相同, `shape` 为 `indices.shape + x.shape[1:]`。

输出:

`Tensor`, 更新后的 `input_x`, `shape` 和数据类型与 `input_x` 相同。

异常:

- **TypeError** - `use_locking` 不是 `bool`。
- **TypeError** - `indices` 不是 `int32` 或者 `int64`。
- **ValueError** - `updates` 的 `shape` 不等于 `indices.shape + x.shape[1:]`。
- **RuntimeError** - 当 `input_x` 和 `updates` 类型不一致, 需要进行类型转换时, 如果 `updates` 不支持转成 `input_x` 需要的数据类型, 即报该错误。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), mindspore.
↪ float32), name="x")
```

(下页继续)

(续上页)

```

>>> indices = Tensor(np.array([[0, 1], [1, 1]]), mindspore.int32)
>>> updates = Tensor(np.ones([2, 2, 3]), mindspore.float32)
>>> scatter_add = ops.ScatterAdd()
>>> output = scatter_add(input_x, indices, updates)
>>> print(output)
[[ 1.  1.  1.]
 [ 3.  3.  3.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), mindspore.
↳ float32), name="x")
>>> # for indices = [[0, 1], [1, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [0.0, 0.0, 0.0] + [1.0, 1.0, 1.0] = [1.0, 1.0, 1.0]
>>> # input_x[1] = [0.0, 0.0, 0.0] + [3.0, 3.0, 3.0] = [3.0, 3.0, 3.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [3.0, 3.0, 3.0] + [7.0, 7.0, 7.0] = [10.0, 10.0, 10.0]
>>> # input_x[1] = [10.0, 10.0, 10.0] + [9.0, 9.0, 9.0] = [19.0, 19.0, 19.0]
>>> indices = Tensor(np.array([[0, 1], [1, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mindspore.float32)
>>> scatter_add = ops.ScatterAdd()
>>> output = scatter_add(input_x, indices, updates)
>>> print(output)
[[ 1.  1.  1.]
 [19. 19. 19.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), mindspore.
↳ float32), name="x")
>>> # for indices = [[1, 0], [1, 1]]
>>> # step 1: [1, 0]
>>> # input_x[0] = [0.0, 0.0, 0.0] + [3.0, 3.0, 3.0] = [3.0, 3.0, 3.0]
>>> # input_x[1] = [0.0, 0.0, 0.0] + [1.0, 1.0, 1.0] = [1.0, 1.0, 1.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [1.0, 1.0, 1.0] + [7.0, 7.0, 7.0] = [8.0, 8.0, 8.0]
>>> # input_x[1] = [8.0, 8.0, 8.0] + [9.0, 9.0, 9.0] = [17.0, 17.0, 17.0]
>>> indices = Tensor(np.array([[1, 0], [1, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mindspore.float32)
>>> scatter_add = ops.ScatterAdd()
>>> output = scatter_add(input_x, indices, updates)
>>> print(output)
[[ 3.  3.  3.]
 [17. 17. 17.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), mindspore.
↳ float32), name="x")
>>> # for indices = [[0, 1], [0, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [0.0, 0.0, 0.0] + [1.0, 1.0, 1.0] = [1.0, 1.0, 1.0]
>>> # input_x[1] = [0.0, 0.0, 0.0] + [3.0, 3.0, 3.0] = [3.0, 3.0, 3.0]
>>> # step 2: [0, 1]
>>> # input_x[0] = [1.0, 1.0, 1.0] + [7.0, 7.0, 7.0] = [8.0, 8.0, 8.0]
>>> # input_x[1] = [3.0, 3.0, 3.0] + [9.0, 9.0, 9.0] = [12.0, 12.0, 12.0]

```

(下页继续)

(续上页)

```
>>> indices = Tensor(np.array([[0, 1], [0, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mindspore.float32)
>>> scatter_add = ops.ScatterAdd()
>>> output = scatter_add(input_x, indices, updates)
>>> print(output)
[[ 8.  8.  8.]
 [12. 12. 12.]
```

20.5.5 mindspore.ops.ScatterDiv

class mindspore.ops.ScatterDiv(*use_locking=False*)

通过除法操作更新输入张量的值。

根据指定更新值和输入索引通过除法操作更新输入数据的值。该操作在更新完成后输出 *input_x*，这样方便使用更新后的值。

对于 *indices.shape* 的每个 *i, ..., j*：

$$\text{input\_x}[\text{indices}[i, \dots, j], :] \div \text{updates}[i, \dots, j, :]$$

输入的 *input_x* 和 *updates* 遵循隐式类型转换规则，以确保数据类型一致。如果数据类型不同，则低精度数据类型将转换为高精度的数据类型。当 *updates* 不支持转成 *input_x* 需要的数据类型时，则会抛出 `RuntimeError` 异常。

参数：

- **use_locking** (bool) - 是否启用锁保护。默认值：False。

输入：

- **input_x** (Union[Parameter, Tensor]) - ScatterDiv 的输入，任意维度的 Parameter 或 Tensor。shape: (*N*, *), 其中 * 表示任意数量的附加维度。
- **indices** (Tensor) - 指定相除操作的索引，数据类型必须为 mindspore.int32 或者 mindspore.int64。
- **updates** (Tensor) - 指定与 *input_x* 相除的 Tensor，数据类型与 *input_x* 相同，shape 为 *indices.shape* + *input_x.shape[1:]*。

输出：

Tensor，更新后的 *input_x*，shape 和类型与 *input_x* 相同。

异常：

- **TypeError** - *use_locking* 不是 bool。
- **TypeError** - *indices* 不是 int32 或者 int64。
- **ValueError** - *updates* 的 shape 不等于 *indices.shape* + *input_x.shape[1:]*。
- **RuntimeError** - 当 *input_x* 和 *updates* 类型不一致，需要进行类型转换时，如果 *updates* 不支持转成 *input_x* 需要的数据类型，就会报错。
- **RuntimeError** - 在 Ascend 平台上，输入的 *input_x*，*indices* 和 *updates* 的数据维度大于八维。

支持平台：

Ascend GPU CPU

样例：

```

>>> import numpy as np
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([[6.0, 6.0, 6.0], [2.0, 2.0, 2.0]]), mstype.float32),
    ↪ name="x")
>>> indices = Tensor(np.array([0, 1]), mstype.int32)
>>> updates = Tensor(np.array([[2.0, 2.0, 2.0], [2.0, 2.0, 2.0]]), mstype.float32)
>>> scatter_div = ops.ScatterDiv()
>>> output = scatter_div(input_x, indices, updates)
>>> print(output)
[[3. 3. 3.]
 [1. 1. 1.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
    ↪ initialized.
>>> input_x = Parameter(Tensor(np.array([[105.0, 105.0, 105.0],
    ...                               [315.0, 315.0, 315.0]]), mstype.float32), name="x")
>>> # for indices = [[0, 1], [1, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [105.0, 105.0, 105.0] / [1.0, 1.0, 1.0] = [105.0, 105.0, 105.0]
>>> # input_x[1] = [315.0, 315.0, 315.0] / [3.0, 3.0, 3.0] = [105.0, 105.0, 105.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [105.0, 105.0, 105.0] / [5.0, 5.0, 5.0] = [21.0, 21.0, 21.0]
>>> # input_x[1] = [21.0, 21.0, 21.0] / [7.0, 7.0, 7.0] = [3.0, 3.0, 3.0]
>>> indices = Tensor(np.array([[0, 1], [1, 1]]), mstype.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
    ...                       [[5.0, 5.0, 5.0], [7.0, 7.0, 7.0]]), mstype.float32)
>>> scatter_div = ops.ScatterDiv()
>>> output = scatter_div(input_x, indices, updates)
>>> print(output)
[[105. 105. 105.]
 [ 3.  3.  3.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
    ↪ initialized.
>>> input_x = Parameter(Tensor(np.array([[105.0, 105.0, 105.0],
    ...                               [315.0, 315.0, 315.0]]), mstype.float32), name="x")
>>> # for indices = [[1, 0], [1, 1]]
>>> # step 1: [1, 0]
>>> # input_x[0] = [105.0, 105.0, 105.0] / [3.0, 3.0, 3.0] = [35.0, 35.0, 35.0]
>>> # input_x[1] = [315.0, 315.0, 315.0] / [1.0, 1.0, 1.0] = [315.0, 315.0, 315.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [315.0, 315.0, 315.0] / [5.0, 5.0, 5.0] = [63.0 63.0 63.0]
>>> # input_x[1] = [63.0 63.0 63.0] / [7.0, 7.0, 7.0] = [9.0, 9.0, 9.0]
>>> indices = Tensor(np.array([[1, 0], [1, 1]]), mstype.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
    ...                       [[5.0, 5.0, 5.0], [7.0, 7.0, 7.0]]), mstype.float32)
>>> scatter_div = ops.ScatterDiv()
>>> output = scatter_div(input_x, indices, updates)
>>> print(output)
[[35. 35. 35.]
 [ 9.  9.  9.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
    ↪ initialized.
>>> input_x = Parameter(Tensor(np.array([[105.0, 105.0, 105.0],
    ...                               [315.0, 315.0, 315.0]]), mstype.float32), name="x")
>>> # for indices = [[0, 1], [0, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [105.0, 105.0, 105.0] / [1.0, 1.0, 1.0] = [105.0, 105.0, 105.0]

```

(下页继续)

(续上页)

```

>>> # input_x[1] = [315.0, 315.0, 315.0] / [3.0, 3.0, 3.0] = [105.0, 105.0, 105.0]
>>> # step 2: [0, 1]
>>> # input_x[0] = [105.0, 105.0, 105.0] / [5.0, 5.0, 5.0] = [21.0, 21.0, 21.0]
>>> # input_x[1] = [105.0, 105.0, 105.0] / [7.0, 7.0, 7.0] = [15.0, 15.0, 15.0]
>>> indices = Tensor(np.array([[0, 1], [0, 1]]), mstype.int32)
>>> updates = Tensor(np.array([[[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[5.0, 5.0, 5.0], [7.0, 7.0, 7.0]]]), mstype.float32)
>>> scatter_div = ops.ScatterDiv()
>>> output = scatter_div(input_x, indices, updates)
>>> print(output)
[[21. 21. 21.]
 [15. 15. 15.]]

```

20.5.6 mindspore.ops.ScatterMax

class mindspore.ops.ScatterMax (use_locking=False)

通过最大操作更新输入张量的值。

根据指定更新值和输入索引通过最大值操作更新输入数据的值。该操作在更新完成后输出 *input_x*，这样方便使用更新后的值。

对于 *indices.shape* 的每个 *i, ..., j*：

$$\text{input\_x}[\text{indices}[i, \dots, j], :] = \max(\text{input\_x}[\text{indices}[i, \dots, j], :], \text{updates}[i, \dots, j, :])$$

输入的 *input_x* 和 *updates* 遵循隐式类型转换规则，以确保数据类型一致。如果数据类型不同，则低精度数据类型将转换为高精度的数据类型。当 *updates* 不支持转成 *input_x* 需要的数据类型时，则会抛出 **RuntimeError** 异常。

参数：

- **use_locking** (bool) - 是否启用锁保护。默认值：False。

输入：

- **input_x** (Union[Parameter, Tensor]) - ScatterMax 的输入，任意维度的 Parameter 或 Tensor。shape: (N,*), 其中 * 表示任意数量的附加维度。
- **indices** (Tensor) - 指定最大值操作的索引，数据类型必须为 mindspore.int32 或者 mindspore.int64。
- **updates** (Tensor) - 指定与 *input_x* 取最大值操作的 Tensor，数据类型与 *input_x* 相同，shape 为 *indices.shape* + *x.shape[1:]*。

输出：

Tensor，更新后的 *input_x*，shape 和类型与 *input_x* 相同。

异常：

- **TypeError** - *use_locking* 不是 bool。
- **TypeError** - *indices* 不是 int32 或者 int64。
- **ValueError** - *updates* 的 shape 不等于 *indices.shape* + *x.shape[1:]*。
- **RuntimeError** - 当 *input_x* 和 *updates* 类型不一致，需要进行类型转换时，如果 *updates* 不支持转成 *input_x* 需要的数据类型，就会报错。
- **RuntimeError** - 在 Ascend 平台上，输入的 *input_x*，*indices* 和 *updates* 的数据维度大于八维。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([[1.0, 2.0, 3.0], [4.0, 5.0, 6.0]]), mindspore.
    ↪float32),
...                       name="input_x")
>>> indices = Tensor(np.array([[0, 0], [1, 1]]), mindspore.int32)
>>> updates = Tensor(np.ones([2, 2, 3]) * 88, mindspore.float32)
>>> scatter_max = ops.ScatterMax()
>>> output = scatter_max(input_x, indices, updates)
>>> print(output)
[[88. 88. 88.]
 [88. 88. 88.]
```

20.5.7 mindspore.ops.ScatterMin

class mindspore.ops.ScatterMin (use_locking=False)

通过最小操作更新输入张量的值。

根据指定更新值和输入索引通过最小值操作更新输入数据的值。该操作在更新完成后输出 *input_x*，这样方便使用更新后的值。

对于 *indices.shape* 的每个 *i, ..., j* :

$$\text{input\_x}[\text{indices}[i, \dots, j], :] = \min(\text{input\_x}[\text{indices}[i, \dots, j], :], \text{updates}[i, \dots, j, :])$$

输入的 *input_x* 和 *updates* 遵循隐式类型转换规则，以确保数据类型一致。如果数据类型不同，则低精度数据类型将转换为高精度的数据类型。当 *updates* 不支持转成 *input_x* 需要的数据类型时，则会抛出 **RuntimeError** 异常。

参数:

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值: `False`。

输入:

- **input_x** (Union[Parameter, Tensor]) - ScatterMin 的输入，任意维度的 Parameter 或 Tensor。shape: $(N, *)$ ，其中 $*$ 表示任意数量的附加维度。
- **indices** (Tensor) - 指定最小值操作的索引，数据类型必须为 `mindspore.int32` 或者 `mindspore.int64`。
- **updates** (Tensor) - 指定与 *input_x* 取最小值操作的 Tensor，数据类型与 *input_x* 相同，shape 为 *indices.shape* + *x.shape[1:]*。

输出:

Tensor，更新后的 *input_x*，shape 和类型与 *input_x* 相同。

异常:

- **TypeError** - *use_locking* 不是 bool。
- **TypeError** - *indices* 不是 `int32` 或者 `int64`。
- **ValueError** - *updates* 的 shape 不等于 *indices.shape* + *x.shape[1:]*。
- **RuntimeError** - 当 *input_x* 和 *updates* 类型不一致，需要进行类型转换时，如果 *updates* 不支持转成 *input_x* 需要的数据类型，就会报错。

- **RuntimeError** - 在 Ascend 平台上，输入的 *input_x*，*indices* 和 *updates* 的数据维度大于八维。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([[0.0, 1.0, 2.0], [0.0, 0.0, 0.0]]), mindspore.
↳float32),
...                      name="input_x")
>>> indices = Tensor(np.array([[0, 0], [1, 1]]), mindspore.int32)
>>> update = Tensor(np.ones([2, 2, 3]), mindspore.float32)
>>> scatter_min = ops.ScatterMin()
>>> output = scatter_min(input_x, indices, update)
>>> print(output)
[[0. 1. 1.]
 [0. 0. 0.]]
```

20.5.8 mindspore.ops.ScatterMul

class mindspore.ops.ScatterMul (use_locking=False)

根据指定更新值和输入索引通过乘法运算更新输入数据的值。

对于 *indices.shape* 的每个 *i, ..., j*：

$$\text{input\_x}[\text{indices}[i, \dots, j], :] \ast= \text{updates}[i, \dots, j, :]$$

输入的 *input_x* 和 *updates* 遵循隐式类型转换规则，以确保数据类型一致。如果数据类型不同，则低精度数据类型将转换为高精度的数据类型。当参数的数据类型需要转换时，则会抛出 **RuntimeError** 异常。

参数：

- **use_locking** (bool) - 是否启用锁保护。默认值：False。

输入：

- **input_x** (Union[Parameter, Tensor]) - ScatterMul 的输入，任意维度的 Parameter 或 Tensor。shape: (N,*), 其中 * 表示任意数量的附加维度。
- **indices** (Tensor) - 指定相乘操作的索引，数据类型必须为 mindspore.int32。
- **updates** (Tensor) - 指定与 *input_x* 相乘的 Tensor，数据类型与 *input_x* 相同，shape 为 *indices.shape* + *x.shape[1:]*。

输出：

Tensor，更新后的 *input_x*，shape 和类型与 *input_x* 相同。

异常：

- **TypeError** - *use_locking* 不是 bool。
- **TypeError** - *indices* 不是 int32。
- **ValueError** - *updates* 的 shape 不等于 *indices.shape* + *x.shape[1:]*。
- **RuntimeError** - 当 *input_x* 和 *updates* 类型不一致，需要进行类型转换时，如果 *updates* 不支持转成 *input_x* 需要的数据类型，就会报错。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([[1.0, 1.0, 1.0], [2.0, 2.0, 2.0]]), mstype.float32),
↳ name="x")
>>> indices = Tensor(np.array([0, 1]), mstype.int32)
>>> updates = Tensor(np.array([[2.0, 2.0, 2.0], [2.0, 2.0, 2.0]]), mstype.float32)
>>> scatter_mul = ops.ScatterMul()
>>> output = scatter_mul(input_x, indices, updates)
>>> print(output)
[[2. 2. 2.]
 [4. 4. 4.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[1.0, 1.0, 1.0], [2.0, 2.0, 2.0]]), mstype.float32),
↳ name="x")
>>> # for indices = [[0, 1], [1, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [1.0, 1.0, 1.0] = [1.0, 1.0, 1.0]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [3.0, 3.0, 3.0] = [6.0, 6.0, 6.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [6.0, 6.0, 6.0] * [7.0, 7.0, 7.0] = [42.0, 42.0, 42.0]
>>> # input_x[1] = [42.0, 42.0, 42.0] * [9.0, 9.0, 9.0] = [378.0, 378.0, 378.0]
>>> indices = Tensor(np.array([[0, 1], [1, 1]]), mstype.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mstype.float32)
>>> scatter_mul = ops.ScatterMul()
>>> output = scatter_mul(input_x, indices, updates)
>>> print(output)
[[ 1.   1.   1.]
 [378. 378. 378.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[1.0, 1.0, 1.0], [2.0, 2.0, 2.0]]), mstype.float32),
↳ name="x")
>>> # for indices = [[1, 0], [1, 1]]
>>> # step 1: [1, 0]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [3.0, 3.0, 3.0] = [3.0, 3.0, 3.0]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [1.0, 1.0, 1.0] = [2.0, 2.0, 2.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [7.0, 7.0, 7.0] = [14.0, 14.0, 14.0]
>>> # input_x[1] = [14.0, 14.0, 14.0] * [9.0, 9.0, 9.0] = [126.0, 126.0, 126.0]
>>> indices = Tensor(np.array([[1, 0], [1, 1]]), mstype.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mstype.float32)
>>> scatter_mul = ops.ScatterMul()
>>> output = scatter_mul(input_x, indices, updates)
>>> print(output)
[[ 3.   3.   3.]
 [126. 126. 126.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[1.0, 1.0, 1.0], [2.0, 2.0, 2.0]]), mstype.float32),
↳ name="x")
```

(下页继续)

(续上页)

```
>>> # for indices = [[0, 1], [0, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [1.0, 1.0, 1.0] = [1.0, 1.0, 1.0]
>>> # input_x[1] = [2.0, 2.0, 2.0] * [3.0, 3.0, 3.0] = [6.0, 6.0, 6.0]
>>> # step 2: [0, 1]
>>> # input_x[0] = [1.0, 1.0, 1.0] * [7.0, 7.0, 7.0] = [7.0, 7.0, 7.0]
>>> # input_x[1] = [6.0, 6.0, 6.0] * [9.0, 9.0, 9.0] = [54.0, 54.0, 54.0]
>>> indices = Tensor(np.array([[0, 1], [0, 1]]), mstype.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mstype.float32)
>>> scatter_mul = ops.ScatterMul()
>>> output = scatter_mul(input_x, indices, updates)
>>> print(output)
[[ 7.  7.  7.]
 [54. 54. 54.]]
```

20.5.9 mindspore.ops.ScatterNdAdd

class mindspore.ops.ScatterNdAdd (use_locking=False)

使用给定值通过加法运算和输入索引更新 Tensor 值。在更新完成后输出 *input_x*。这有利于更加方便地使用更新后的值。

更多参考详见 `mindspore.ops.scatter_nd_add()`。

参数：

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值：False。

输入：

- **input_x** (Union[Parameter, Tensor]) - 输入参数，数据类型是 Parameter 或 Tensor。其 shape 为 $(N, *)$ ，其中 * 为任意数量的额外维度。
- **indices** (Tensor) - 指定加法操作的索引，数据类型为 `mindspore.int32`。索引的 rank 必须至少为 2，并且 `indices.shape[-1] <= len(shape)`。
- **updates** (Tensor) - 指定与 *input_x* 相加操作的 Tensor，数据类型与 *input_x* 相同，shape 为 `indices.shape[:-1] + x.shape[indices.shape[-1]:]`。

输出：

Tensor，更新后的 *input_x*，shape 和数据类型与 *input_x* 相同。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8]), mindspore.float32), name=
↳ "x")
>>> indices = Tensor(np.array([[2], [4], [1], [7]]), mindspore.int32)
>>> updates = Tensor(np.array([6, 7, 8, 9]), mindspore.float32)
>>> use_locking = False
>>> scatter_nd_add = ops.ScatterNdAdd(use_locking)
```

(下页继续)

(续上页)

```

>>> output = scatter_nd_add(input_x, indices, updates)
>>> print(output)
[ 1. 10.  9.  4. 12.  6.  7. 17.]
>>> input_x = Parameter(Tensor(np.zeros((4, 4, 4)), mindspore.int32))
>>> indices = Tensor(np.array([[0], [2]]), mindspore.int32)
>>> updates = Tensor(np.array([[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                             [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]]),
mindspore.int32)
>>> use_locking = False
>>> scatter_nd_add = ops.ScatterNdAdd(use_locking)
>>> output = scatter_nd_add(input_x, indices, updates)
>>> print(output)
[[[1 1 1 1]
  [2 2 2 2]
  [3 3 3 3]
  [4 4 4 4]]
 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]
 [[5 5 5 5]
  [6 6 6 6]
  [7 7 7 7]
  [8 8 8 8]]
 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]

```

20.5.10 mindspore.ops.ScatterNdSub

class mindspore.ops.ScatterNdSub (use_locking=False)

使用给定值通过减法运算和输入索引更新 Tensor 值。在更新完成后输出 *input_x*，这有利于更加方便地使用更新后的值。

更多参考详见 *mindspore.ops.scatter_nd_sub()*。

参数：

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值：False。

输入：

- **input_x** (Union[Parameter, Tensor]) - 输入参数，数据类型为 Parameter 或 Tensor，其 shape 为 $(N, *)$ ，其中 * 为任意数量的额外维度。
- **indices** (Tensor) - 指定减法操作的索引，数据类型为 mindspore.int32。索引的 rank 必须至少为 2，并且 $indices.shape[-1] \leq len(shape)$ 。
- **updates** (Tensor) - 指定与 *input_x* 进行减法操作的 Tensor，数据类型与 *input_x* 相同，shape 为 $indices.shape[:-1] + x.shape[indices.shape[-1]:]$ 。

输出：

Tensor，更新后的 *input_x*，shape 和数据类型与 *input_x* 相同。

支持平台：

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([1, 2, 3, 4, 5, 6, 7, 8]), mindspore.float32), name=
↳ "x")
>>> indices = Tensor(np.array([[2], [4], [1], [7]]), mindspore.int32)
>>> updates = Tensor(np.array([6, 7, 8, 9]), mindspore.float32)
>>> use_locking = False
>>> scatter_nd_sub = ops.ScatterNdSub(use_locking)
>>> output = scatter_nd_sub(input_x, indices, updates)
>>> print(output)
[ 1. -6. -3.  4. -2.  6.  7. -1.]
>>> input_x = Parameter(Tensor(np.zeros((4, 4, 4)), mindspore.int32))
>>> indices = Tensor(np.array([[0], [2]]), mindspore.int32)
>>> updates = Tensor(np.array([[[[1, 1, 1, 1], [2, 2, 2, 2], [3, 3, 3, 3], [4, 4, 4, 4]],
...                               [[5, 5, 5, 5], [6, 6, 6, 6], [7, 7, 7, 7], [8, 8, 8, 8]]]),
↳ mindspore.int32)
>>> use_locking = False
>>> scatter_nd_sub = ops.ScatterNdSub(use_locking)
>>> output = scatter_nd_sub(input_x, indices, updates)
>>> print(output)
[[[-1 -1 -1 -1]
  [-2 -2 -2 -2]
  [-3 -3 -3 -3]
  [-4 -4 -4 -4]]
 [[ 0  0  0  0]
  [ 0  0  0  0]
  [ 0  0  0  0]
  [ 0  0  0  0]]
 [[-5 -5 -5 -5]
  [-6 -6 -6 -6]
  [-7 -7 -7 -7]
  [-8 -8 -8 -8]]
 [[ 0  0  0  0]
  [ 0  0  0  0]
  [ 0  0  0  0]
  [ 0  0  0  0]]]
```

20.5.11 mindspore.ops.ScatterNdUpdate

class mindspore.ops.ScatterNdUpdate (use_locking=True)

使用给定值以及输入索引更新输入数据的值。

input_x 的 rank 为 P ，而 *indices* 的 rank 为 Q ， $Q \geq 2$ 。

indices 的 shape 为 $(i_0, i_1, \dots, i_{Q-2}, N)$ ， $N \leq P$ 。

indices 的最后一个维度（长度为 N ）表示沿着 *input_x* 的 N 个维度进行切片。

updates 表示 rank 为 $Q-1+P-N$ 的 Tensor，shape 为 $(i_0, i_1, \dots, i_{Q-2}, x\_shape_N, \dots, x\_shape_{P-1})$ 。

输入的 *input_x* 和 *updates* 遵循隐式类型转换规则，以确保数据类型一致。如果数据类型不同，则低优先级数据类型将转换为相对最高优先级的数据类型。如果无法进行数据类型转换，则会抛出 RuntimeError 异常。

参数：

- **use_locking** (bool, 可选) - 是否启用锁保护。默认值：True。

输入：

- **input_x** (Union[Parameter, Tensor]) - ScatterNdUpdate 的输入，任意维度的 Parameter 或 Tensor。其 shape 为 $(N, *)$ ，其中 $*$ 为任意数量的额外维度。
- **indices** (Tensor) - 指定更新操作的索引，数据类型为 int32 或者 int64。
- **updates** (Tensor) - 指定与 *input_x* 更新操作的 Tensor，类型与输入相同。shape 为 $indices.shape[:-1] + x.shape[indices.shape[-1]:]$ 。

输出：

Tensor，shape 和数据类型与输入 *input_x* 相同。

异常：

- **TypeError** - *use_locking* 不是 bool。
- **TypeError** - *indices* 不是 int32 或者 int64。
- **RuntimeError** - 当 *input_x* 和 *updates* 类型不一致，需要进行类型转换时，如果 *updates* 不支持转成 *input_x* 需要的数据类型，就会报错。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> np_x = np.array([[ -0.1, 0.3, 3.6], [0.4, 0.5, -3.2]])
>>> input_x = mindspore.Parameter(Tensor(np_x, mindspore.float32), name="x")
>>> indices = Tensor(np.array([[0, 0], [1, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([1.0, 2.2]), mindspore.float32)
>>> op = ops.ScatterNdUpdate()
>>> output = op(input_x, indices, updates)
>>> print(output)
[[1.  0.3  3.6]
 [0.4  2.2 -3.2]]
```

20.5.12 mindspore.ops.ScatterNonAliasingAdd

class mindspore.ops.ScatterNonAliasingAdd

ScatterNonAliasingAdd 接口已废弃，请使用接口 *mindspore.ops.TensorScatterAdd* 替换，废弃版本 2.1。

支持平台：

Deprecated

20.5.13 mindspore.ops.ScatterSub

class mindspore.ops.ScatterSub (use_locking=False)

使用给定更新值通过减法操作和输入索引来更新 Tensor 值。此操作在更新完成后输出数据，这有利于更加方便地使用更新后的值。

对于每个在 *indices.shape* 中的 $i, \dots, j$:

$$\text{input\_x}[\text{indices}[i, \dots, j], :] -= \text{updates}[i, \dots, j, :]$$

输入的 *input_x* 和 *updates* 遵循隐式类型转换规则，以确保数据类型一致。如果它们具有不同的数据类型，则优先级低的数据类型将转换为优先级相对最高的数据类型。当需要转换 Parameter 的数据类型时，会抛出 RuntimeError 异常。

参数：

- **use_locking** (bool, 可选) - 表示是否使用锁来保护。默认值：False。

输入：

- **input_x** (Union[Parameter, Tensor]) - ScatterSub 的输入，数据类型为 Parameter。其 shape 为 $(N, *)$ ，其中 * 为任意数量的额外维度。
- **indices** (Tensor) - 指定相减操作的索引，其数据类型必须为 mindspore.int32 或 mindspore.int64。
- **updates** (Tensor) - 指定与 *input_x* 相减的 Tensor，其数据类型与 *input_x* 的相同，shape 为 $\text{indices\_shape} + \text{x\_shape}[1:]$ 。

输出：

Tensor，表示更新后的 *input_x*，其 shape 和数据类型与 *input_x* 的相同。

异常：

- **TypeError** - *use_locking* 不是 bool。
- **TypeError** - *indices* 不是 int32。
- **ValueError** - *updates* 的 shape 不是 $\text{indices\_shape} + \text{x\_shape}[1:]$ 。
- **RuntimeError** - 当 *input_x* 和 *updates* 类型不一致，需要进行类型转换时，如果 *updates* 不支持转成参数 *input_x* 需要的数据类型，就会报错。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops, Parameter
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [1.0, 1.0, 1.0]]), mindspore.
↪float32), name="x")
>>> indices = Tensor(np.array([[0, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [2.0, 2.0, 2.0]]), mindspore.float32)
>>> scatter_sub = ops.ScatterSub()
>>> output = scatter_sub(input_x, indices, updates)
>>> print(output)
[[-1. -1. -1.]
 [-1. -1. -1.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↪initialized.
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), mindspore.
↪float32), name="x")
```

(下页继续)

(续上页)

```

>>> # for indices = [[0, 1], [1, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [0.0, 0.0, 0.0] - [1.0, 1.0, 1.0] = [-1.0, -1.0, -1.0]
>>> # input_x[1] = [0.0, 0.0, 0.0] - [3.0, 3.0, 3.0] = [-3.0, -3.0, -3.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [-3.0, -3.0, -3.0] - [7.0, 7.0, 7.0] = [-10.0, -10.0, -10.0]
>>> # input_x[1] = [-10.0, -10.0, -10.0] - [9.0, 9.0, 9.0] = [-19.0, -19.0, -19.0]
>>> indices = Tensor(np.array([[0, 1], [1, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mindspore.float32)
>>> scatter_sub = ops.ScatterSub()
>>> output = scatter_sub(input_x, indices, updates)
>>> print(output)
[[ -1.  -1.  -1.]
 [-19. -19. -19.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), mindspore.
↳ float32), name="x")
>>> # for indices = [[1, 0], [1, 1]]
>>> # step 1: [1, 0]
>>> # input_x[0] = [0.0, 0.0, 0.0] - [3.0, 3.0, 3.0] = [-3.0, -3.0, -3.0]
>>> # input_x[1] = [0.0, 0.0, 0.0] - [1.0, 1.0, 1.0] = [-1.0, -1.0, -1.0]
>>> # step 2: [1, 1]
>>> # input_x[1] = [-1.0, -1.0, -1.0] - [7.0, 7.0, 7.0] = [-8.0, -8.0, -8.0]
>>> # input_x[1] = [-8.0, -8.0, -8.0] - [9.0, 9.0, 9.0] = [-17.0, -17.0, -17.0]
>>> indices = Tensor(np.array([[1, 0], [1, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mindspore.float32)
>>> scatter_sub = ops.ScatterSub()
>>> output = scatter_sub(input_x, indices, updates)
>>> print(output)
[[ -3.  -3.  -3.]
 [-17. -17. -17.]]
>>> # for input_x will be updated after the operation is completed. input_x need to be re-
↳ initialized.
>>> input_x = Parameter(Tensor(np.array([[0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]), mindspore.
↳ float32), name="x")
>>> # for indices = [[0, 1], [0, 1]]
>>> # step 1: [0, 1]
>>> # input_x[0] = [0.0, 0.0, 0.0] - [1.0, 1.0, 1.0] = [-1.0, -1.0, -1.0]
>>> # input_x[1] = [0.0, 0.0, 0.0] - [3.0, 3.0, 3.0] = [-3.0, -3.0, -3.0]
>>> # step 2: [0, 1]
>>> # input_x[0] = [-1.0, -1.0, -1.0] - [7.0, 7.0, 7.0] = [-8.0, -8.0, -8.0]
>>> # input_x[1] = [-3.0, -3.0, -3.0] - [9.0, 9.0, 9.0] = [-12.0, -12.0, -12.0]
>>> indices = Tensor(np.array([[0, 1], [0, 1]]), mindspore.int32)
>>> updates = Tensor(np.array([[1.0, 1.0, 1.0], [3.0, 3.0, 3.0]],
...                             [[7.0, 7.0, 7.0], [9.0, 9.0, 9.0]]), mindspore.float32)
>>> scatter_sub = ops.ScatterSub()
>>> output = scatter_sub(input_x, indices, updates)
>>> print(output)
[[ -8.  -8.  -8.]
 [-12. -12. -12.]]

```

20.5.14 mindspore.ops.ScatterUpdate

class mindspore.ops.ScatterUpdate (use_locking=True)

使用给定的更新值和输入索引更新输入 Tensor 的值。

使用给定的值来更新张量值，以及输入指数。

若 *indices* 的 shape 为 (i, ..., j)，则

$$\text{input\_x}[\text{indices}[i, \dots, j], :] = \text{updates}[i, \dots, j, :]$$

输入的 *input_x* 和 *updates* 遵循隐式类型转换规则，以确保数据类型一致。如果它们具有不同的数据类型，则低精度数据类型将转换为高精度数据类型。当需要转换 Parameter 的数据类型时，会抛出 RuntimeError 异常。

参数：

- **use_locking** (bool) - 表示是否使用锁来保护。默认值：True。

输入：

- **input_x** (Union[Parameter, Tensor]) - ScatterUpdate 的输入，shape：零维或 (N, *)，其中 * 表示任意数量的附加维度。
- **indices** (Tensor) - 指定更新操作的索引。数据类型为 int32。如果索引中存在重复项，则更新的顺序无法得知。
- **updates** (Tensor) - 指定与 *input_x* 更新操作的 Tensor，其数据类型与 *input_x* 相同，shape 为 *indices.shape* + *input_x.shape[1:]*。

输出：

Tensor，shape 和数据类型与输入 *input_x* 相同。

异常：

- **TypeError** - *use_locking* 不是 bool。
- **TypeError** - *indices* 不是 int32。
- **ValueError** - *updates* 的 shape 不等于 *indices.shape* + *input_x.shape[1:]*。
- **RuntimeError** - 当 *input_x* 和 *updates* 类型不一致，需要进行类型转换时，如果 *updates* 不支持转成 *input_x* 需要的数据类型，就会报错。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> np_x = np.array([[0.1, 0.3, 3.6], [0.4, 0.5, -3.2]])
>>> input_x = mindspore.Parameter(Tensor(np_x, mindspore.float32), name="x")
>>> indices = Tensor(np.array([0, 1]), mindspore.int32)
>>> np_updates = np.array([[2.0, 1.2, 1.0], [3.0, 1.2, 1.0]])
>>> updates = Tensor(np_updates, mindspore.float32)
>>> op = ops.ScatterUpdate()
>>> output = op(input_x, indices, updates)
>>> print(output)
[[2. 1.2 1.]
 [3. 1.2 1.]
```

20.6 数据操作算子

接口名	概述	支持平台
<code>mindspore.ops.GetNext</code>	返回数据集队列中的下一个元素。	Ascend GPU

20.6.1 mindspore.ops.GetNext

class `mindspore.ops.GetNext` (*types, shapes, output_num, shared_name*)
返回数据集队列中的下一个元素。

说明: `GetNext` 操作需要与 `network` 一起使用, 且依赖'dataset' 接口, 例如: `mindspore.dataset.MnistDataset`。不能单独操作。详见`mindspore.connect _network_with_dataset` 的源码。

参数:

- **types** (list[`mindspore.dtype`]) - 输出的数据类型。
- **shapes** (list[tuple[int]]) - 输出数据的 `shape` 大小。
- **output_num** (int) - 输出编号、*types* 和 *shapes* 的长度。
- **shared_name** (str) - 待获取数据的队列名称。

输入:

没有输入。

输出:

tuple[`Tensor`], `Dataset` 的输出。Shape 和类型参见 *shapes*、*types*。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> from mindspore import ops
>>> from mindspore import dataset as ds
>>> from mindspore import dtype as mstype
>>> data_path = "/path/to/MNIST_Data/train/"
>>> train_dataset = ds.MnistDataset(data_path, num_samples=10)
>>> dataset_helper = mindspore.DatasetHelper(train_dataset, dataset_sink_mode=True)
>>> dataset = dataset_helper.iter_dataset
>>> dataset_types, dataset_shapes = dataset_helper.types_shapes()
>>> queue_name = dataset.__transfer_dataset__.queue_name
>>> get_next = ops.GetNext(dataset_types, dataset_shapes, len(dataset_types), queue_name)
>>> data, label = get_next()
>>> relu = ops.ReLU()
>>> result = relu(data.astype(mstype.float32))
>>> print(result.shape)
(28, 28, 1)
```

20.7 通信算子

在分布式训练中进行数据传输涉及通信操作，详情请参考 [分布式集合通信原语](#)。

注意，以下列表中的接口需要先配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

接口名	概述	支持平台
<code>mindspore.ops.AllGather</code>	在指定的通信组中汇聚 <code>Tensor</code> ，返回汇聚后的张量。	Ascend GPU
<code>mindspore.ops.AllReduce</code>	使用指定方式对通信组内的所有设备的 <code>Tensor</code> 数据进行规约操作，所有设备都得到相同的结果，返回规约操作后的张量。	Ascend GPU CPU
<code>mindspore.ops.AlltoAll</code>	<code>AlltoAll</code> 是一个集合通信函数。	Ascend
<code>mindspore.ops.AlltoAllV</code>	相对 <code>AlltoAll</code> 来说， <code>AlltoAllV</code> 算子支持不等分的切分和聚合。	Ascend
<code>mindspore.ops.Barrier</code>	同步通信域内的多个进程。	Ascend
<code>mindspore.ops.Broadcast</code>	对输入数据整组广播。	Ascend GPU
<code>mindspore.ops.CollectiveGather</code>	对通信组的输入张量进行聚合。	Ascend
<code>mindspore.ops.CollectiveScatter</code>	将输入数据均匀散射到通信域的卡上。	Ascend
<code>mindspore.ops.NeighborExchangeV2</code>	<code>NeighborExchangeV2</code> 是一个集合通讯操作。	Ascend
<code>mindspore.ops.Receive</code>	接收来自 <code>src_rank</code> 线程的张量。	Ascend GPU
<code>mindspore.ops.ReduceOp</code>	规约张量的操作选项。	Ascend GPU
<code>mindspore.ops.ReduceScatter</code>	规约并且分发指定通信组中的张量，返回分发后的张量。	Ascend GPU
<code>mindspore.ops.Reduce</code>	规约指定通信组中的 <code>Tensor</code> ，并将规约结果发送到目标为 <code>dest_rank</code> 的进程中，返回发送到目标进程的 <code>Tensor</code> 。	Ascend
<code>mindspore.ops.Send</code>	发送张量到指定线程。	Ascend GPU

20.7.1 mindspore.ops.AllGather

class `mindspore.ops.AllGather` (*group=GlobalComm.WORLD_COMM_GROUP*)

在指定的通信组中汇聚 `Tensor`，返回汇聚后的张量。

说明:

- 集合中所有进程的 `Tensor` 必须具有相同的 `shape` 和格式。

参数:

- group** (str, 可选) - 工作的通信组，默认值: `GlobalComm.WORLD_COMM_GROUP`（即 Ascend 平台为 `"hccl_world_group"`，GPU 平台为 `"nccl_world_group"`）。

输入:

- input_x** (`Tensor`) - `AllGather` 的输入，`shape` 为 $(x_1, x_2, \dots, x_R)$ 的 `Tensor`。

输出:

`Tensor`，如果组中的 `device` 数量为 `N`，则输出的 `shape` 为 $(N * x_1, x_2, \dots, x_R)$ 。

异常:

- **TypeError** - *group* 不是 str。
- **ValueError** - 调用进程的 rank id 大于本通信组的 rank 大小。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。

支持平台:

Ascend GPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import ops
>>> import mindspore.nn as nn
>>> from mindspore.communication import init
>>> from mindspore import Tensor
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.allgather = ops.AllGather()
...
...     def construct(self, x):
...         return self.allgather(x)
...
>>> input_x = Tensor(np.ones([2, 8]).astype(np.float32))
>>> net = Net()
>>> output = net(input_x)
>>> print(output)
[[1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 1. 1. 1.]]
```

教程样例:

- 分布式集合通信原语 - AllGather

(续上页)

```
>>> input_ = Tensor(np.ones([2, 8]).astype(np.float32))
>>> net = Net()
>>> output = net(input_)
>>> print(output)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]]
```

教程样例：

- 分布式集合通信原语 - AllReduce

20.7.3 mindspore.ops.AlltoAll

class mindspore.ops.**AlltoAll** (*split_count, split_dim, concat_dim, group=GlobalComm.WORLD_COMM_GROUP*)
AlltoAll 是一个集合通信函数。

AlltoAll 将输入数据在特定的维度切分成特定的块数 (blocks)，并按顺序发送给其他 rank。一般有两个阶段：

- 分发阶段：在每个进程上，操作数沿着 *split_dim* 拆分为 *split_count* 个块 (blocks)，且分发到指定的 rank 上，例如，第 *i* 块被发送到第 *i* 个 rank 上。
- 聚合阶段：每个 rank 沿着 *concat_dimension* 拼接接收到的数据。

说明：聚合阶段，所有进程中的 Tensor 必须具有相同的 shape 和格式。

要求全连接配网方式，每台设备具有相同的 vlan id，ip 和 mask 在同一子网，请查看 [详细信息](#)。

参数：

- **split_count** (int) - 在每个进程上，将块 (blocks) 拆分为 *split_count* 个。
- **split_dim** (int) - 在每个进程上，沿着 *split_dim* 维度进行拆分。
- **concat_dim** (int) - 在每个进程上，沿着 *concat_dimension* 拼接接收到的块 (blocks)。
- **group** (str, 可选) - AlltoAll 的通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

输入：

- **input_x** (Tensor) - shape 为 $(x_1, x_2, \dots, x_R)$ 。

输出：

Tensor，设输入的 shape 是 $(x_1, x_2, \dots, x_R)$ ，则输出的 shape 为 $(y_1, y_2, \dots, y_R)$ ，其中：

- $y_{split\_dim} = x_{split\_dim} / split\_count$
- $y_{concat\_dim} = x_{concat\_dim} * split\_count$
- $y_{other} = x_{other}$ 。

异常：

- **TypeError** - 如果 *group* 不是字符串。

支持平台：

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 8 卡环境下运行。

```
>>> import os
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication import init
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> import numpy as np
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.alltoall = ops.AlltoAll(split_count = 8, split_dim = -2, concat_dim = -1)
...
...     def construct(self, x):
...         out = self.alltoall(x)
...         return out
...
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> net = Net()
>>> rank_id = int(os.getenv("RANK_ID"))
>>> input_x = Tensor(np.ones([1, 1, 8, 1]) * rank_id, dtype = ms.float32)
>>> output = net(input_x)
>>> print(output)
[[[[0. 1. 2. 3. 4. 5. 6. 7.]]]]
```

教程样例:

- [分布式集合通信原语 - AlltoAll](#)

20.7.4 mindspore.ops.AlltoAllV

class mindspore.ops.AlltoAllV(*group=GlobalComm.WORLD_COMM_GROUP, block_size=1*)

相对 AlltoAll 来说, AlltoAllV 算子支持不等分的切分和聚合。

说明: 只支持一维的输入, 使用该接口前需要将输入数据展开成一维。

参数:

- **group** (str, 可选) - AlltoAll 的通信域。默认值: `GlobalComm.WORLD_COMM_GROUP`, Ascend 平台表示为 `"hccl_world_group"`。
- **block_size** (int, 可选) - 通过 `send_numel_list` 和 `recv_numel_list` 切分和聚合数据量的基本单位。默认值: 1。

输入:

- **input_x** (Tensor) - 一维待分发的张量, shape 为 (x_1) 。

- **send_numel_list** (Union[tuple[int], list[int], Tensor]) - 分发给每张卡的数据量。实际分发数据量为 ($send\_numel\_list * block\_size * input\_x.dtype$)。
- **recv_numel_list** (Union[tuple[int], list[int], Tensor]) - 从每张卡聚合的数据量。实际聚合数据量为 ($send\_numel\_list * block\_size * input\_x.dtype$)。

输出:

Tensor, 从每张卡上聚合的一维数据结果。如果结果为空, 则返回空张量, 且张量数值无意义。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> from mindspore import ops
>>> import mindspore.nn as nn
>>> from mindspore.communication import init, get_rank
>>> from mindspore import Tensor
>>>
>>> init()
>>> rank = get_rank()
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.all_to_all = ops.AlltoAllV()
...
...     def construct(self, x, send_numel_list, recv_numel_list):
...         return self.all_to_all(x, send_numel_list, recv_numel_list)
>>> send_numel_list = []
>>> recv_numel_list = []
>>> if rank == 0:
...     send_tensor = Tensor([0, 1, 2])
...     send_numel_list = [1, 2]
...     recv_numel_list = [1, 2]
>>> elif rank == 1:
...     send_tensor = Tensor([3, 4, 5])
...     send_numel_list = [2, 1]
...     recv_numel_list = [2, 1]
>>> net = Net()
>>> output = net(send_tensor, send_numel_list, recv_numel_list)
>>> print(output)
rank 0:
[0. 3. 4]
rank 1:
[1. 2. 5]
```

教程样例:

- [分布式集合通信原语 - AlltoAllV](#)

20.7.5 mindspore.ops.Barrier

class mindspore.ops.Barrier (group=GlobalComm.WORLD_COMM_GROUP)

同步通信域内的多个进程。进程调用到该算子后进入阻塞状态，直到通信域内所有进程调用到该算子，进程被唤醒并继续执行。

参数：

- **group** (str, 可选) - 表示通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

异常：

- **TypeError** - group 不为 str。
- **RuntimeError** - 后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 4 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore.communication import init
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> # Launch 4 processes.
>>> init()
>>> class BarrierNet(nn.Cell):
...     def __init__(self):
...         super(BarrierNet, self).__init__()
...         self.barrier = ops.Barrier()
...
...     def construct(self):
...         self.barrier()
>>> net = BarrierNet()
>>> net()
```

教程样例：

- 分布式集合通信原语 - Barrier

20.7.6 mindspore.ops.Broadcast

class mindspore.ops.**Broadcast** (*root_rank*, *group*=*GlobalComm.WORLD_COMM_GROUP*)
对输入数据整组广播。

说明：集合中的所有进程的 Tensor 的 shape 和数据格式相同。

参数：

- **root_rank** (int) - 表示发送源的进程编号。只有该进程会广播张量。
- **group** (str, 可选) - 表示通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

输入：

- **input_x** (tuple[Tensor]) - Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。

输出：

tuple[Tensor], Tensor 的 shape 与输入相同，即 $(x_1, x_2, \dots, x_R)$ 。内容取决于 *root_rank* device 的数据。

异常：

- **TypeError** - *root_rank* 不是 int 或 *group* 不是 str。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在多卡环境下运行。

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication import init
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> import numpy as np
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.broadcast = ops.Broadcast(1)
...
...     def construct(self, x):
...         return self.broadcast((x,))
...
>>> input_x = Tensor(np.ones([2, 4]).astype(np.int32))
>>> net = Net()
>>> output = net(input_x)
>>> print(output)
```

(下页继续)

(续上页)

```
(Tensor(shape=[2,4], dtype=Int32, value=
[[1, 1, 1, 1],
[1, 1, 1, 1]]),)
```

教程样例：

- 分布式集合通信原语 - Broadcast

20.7.7 mindspore.ops.CollectiveGather

class mindspore.ops.CollectiveGather (*dest_rank, group=GlobalComm.WORLD_COMM_GROUP*)

对通信组的输入张量进行聚合。操作会将每张卡的输入 Tensor 的第 0 维度进行聚合，发送到对应卡上。

说明：只有目标为 `dest_rank` 的进程（全局的进程编号）才会收到聚合操作后的输出。其他进程只得到一个形状为 [1] 的张量，且该张量没有数学意义。

参数：

- **dest_rank** (int) - 表示发送目标的进程编号。只有该进程会接收汇聚张量。
- **group** (str, 可选) - 表示通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

输入：

- **input_x** (Tensor) - 输入待聚合的 Tensor，Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。

输出：

Tensor，Tensor 第 0 维等于各输入数据第 0 维的累加，其他 shape 维度相同即 $(\sum x_1, x_2, \dots, x_R)$ 。

异常：

- **TypeError** - *group* 不为 str。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。
- **ValueError** - 调用进程的 rank id 大于本通信组的 rank 大小。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore as ms
>>> import mindspore.nn as nn
>>> from mindspore.communication import init
>>> from mindspore import Tensor
```

(下页继续)

(续上页)

```

>>> from mindspore import ops
>>> # Launch 2 processes.
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> class CollectiveGatherNet(nn.Cell):
...     def __init__(self):
...         super(CollectiveGatherNet, self).__init__()
...         self.collective_gather = ops.CollectiveGather(dest_rank=0)
...
...     def construct(self, x):
...         return self.collective_gather(x)
...
>>> input = Tensor(np.arange(4).reshape([2, 2]).astype(np.float32))
>>> net = CollectiveGatherNet()
>>> output = net(input)
>>> print(output)
Process with rank 0: [[0. 1.],
[2. 3.],
[0. 1.],
[2. 3.]]
Process with rank 1: [0.]

```

教程样例:

- 分布式集合通信原语 - [CollectiveGather](#)

20.7.8 mindspore.ops.CollectiveScatter

class mindspore.ops.CollectiveScatter (src_rank=0, group=GlobalComm.WORLD_COMM_GROUP)

将输入数据均匀散射到通信域的卡上。

说明: 该接口只支持 Tensor 输入, 且只支持均匀切分。只有源为 src_rank 的进程 (全局的进程编号) 才会将输入张量作为散射源。

参数:

- **src_rank** (int, 可选) - 表示发送源的进程编号。只有该进程会发送散射源 Tensor。
- **group** (str, 可选) - 表示通信域。默认值: GlobalComm.WORLD_COMM_GROUP。

输入:

- **input_x** (Tensor) - 输入待散射的 Tensor。Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。

输出:

Tensor, Tensor 第 0 维等于散射源输入第 0 维除以 src_rank, 其他 shape 维度相同即 $(x_1/src\_rank, x_2, \dots, x_R)$ 。

异常:

- **TypeError** - group 不为 str。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。
- **ValueError** - 调用进程的 rank id 大于本通信组的 rank 大小。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> from mindspore.communication.management import init, get_rank
>>> from mindspore import ops
>>> # Launch 2 processes.
>>> init()
>>> class CollectiveScatterNet(nn.Cell):
...     def __init__(self):
...         super(CollectiveScatterNet, self).__init__()
...         self.collective_scatter = ops.CollectiveScatter(src_rank=0)
...
...     def construct(self, x):
...         return self.collective_scatter(x)
>>>
>>> input = Tensor(np.arange(8).reshape([4, 2]).astype(np.float32))
>>> net = CollectiveScatterNet()
>>> output = net(input)
>>> print(output)
Process with rank 0: [[0. 1.],
[2. 3.]]
Process with rank 1: [[4. 5.],
[6. 7.]]
```

教程样例：

- [分布式集合通信原语 - CollectiveScatter](#)

20.7.9 mindspore.ops.NeighborExchangeV2

class mindspore.ops.NeighborExchangeV2(*send_rank_ids, recv_rank_ids, send_lens, recv_lens, data_format, group=GlobalComm.WORLD_COMM_GROUP*)

NeighborExchangeV2 是一个集合通讯操作。

将数据从本地 rank 发送到 *send_rank_ids* 中指定的 rank，同时从 *recv_rank_ids* 接收数据。请参考下方教程样例了解具体的数据是如何在相邻设备间交换的。

说明：

- 要求全连接配网，每台设备具有相同的 `vlan id`，`ip` 和 `mask` 在同一子网，请查看 [分布式集合通信原语注意事项](#)。
- 需要用户确保接收数据的长度 *recv_lens* 与发送数据的长度 *send_lens* 一致。

参数:

- **send_rank_ids** (list(int)) - 指定发送数据的 rank。8 个 rank_id 分别代表 8 个方向上的数据要向哪个 rank 发送, 如果某个方向上不发送数据, 则设为-1。
- **recv_rank_ids** (list(int)) - 指定接收数据的 rank。8 个 rank_id 分别代表 8 个方向上的数据要从哪个 rank 接收, 如果某个方向上不接收数据, 则设为-1。
- **send_lens** (list(int)) - 指定 *send_rank_ids* 发送数据的长度, 4 个数字分别代表 [send_top, send_bottom, send_left, send_right]4 个方向上的长度。
- **recv_lens** (list(int)) - 指定 *recv_rank_ids* 接收数据的长度, 4 个数字分别代表 [recv_top, recv_bottom, recv_left, recv_right]4 个方向上的长度。
- **data_format** (str) - 数据格式, 现在只支持 NCHW。
- **group** (str, 可选) - 工作的通信组。默认值: GlobalComm.WORLD_COMM_GROUP (即 Ascend 平台为 "hccl_world_group", GPU 平台为 "nccl_world_group")。

输入:

- **input_x** (Tensor) - 交换前的输入 Tensor, 其 shape 为 (N, C, H, W) 。

输出:

数据交换后的输出 Tensor, 如果输入的 shape 是 (N, C, H, W) , 则输出 shape 为 $(N, C, H+recv\_top+recv\_bottom, W+recv\_left+recv\_right)$ 。

异常:

- **TypeError** - 如果 *group* 不是一个 string 或者 *send_rank_ids*、*recv_rank_ids*、*send_lens* 和 *recv_lens* 中任意一个不是一个 list。
- **ValueError** - 如果 *send_rank_ids* 或者 *recv_rank_ids* 存在小于-1 的值或者存在重复值。
- **ValueError** - 如果 *send_lens* 或者 *recv_lens* 存在小于零的值。
- **ValueError** - 如果 *data_format* 不是"NCHW"。

支持平台:

Ascend

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import os
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> import numpy as np
>>>
>>> class Net0(nn.Cell):
...     def __init__(self):
...         super(Net0, self).__init__()
...         self.neighbor_exchangev2 = ops.NeighborExchangeV2(send_rank_ids=[-1, -1, -1, -1, -1, -1, -1, -1],
...         recv_rank_ids=[-1, -1, -1, -1, -1, -1, -1, -1],
```

(下页继续)

(续上页)

```

...
...
↪1, -1, -1, -1],
...
↪format="NCHW")
...
...
def construct(self, x):
...
    out = self.neighbor_exchangev2(x)
...
    return out
>>> class Net1(nn.Cell):
...
    def __init__(self):
...
        super(Net1, self).__init__()
...
        self.neighbor_exchangev2 = ops.NeighborExchangeV2(send_rank_ids=[0, -1, -1, -1, -
↪1, -1, -1, -1],
...
...
...
        send_lens=[1, 0, 0, 0],
        recv_rank_ids=[0, -1, -1, -1, -
↪1, -1, -1, -1],
...
...
        recv_lens=[1, 0, 0, 0], data_
↪format="NCHW")
...
...
def construct(self, x):
...
    out = self.neighbor_exchangev2(x)
...
    return out
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> rank_id = int(os.getenv("RANK_ID"))
>>> if (rank_id % 2 == 0):
...
    input_x = ms.Tensor(np.ones([1, 1, 2, 2]), dtype = ms.float32)
...
    net = Net0()
...
    output = net(input_x)
...
    print(output)
...
else:
...
    input_x = ms.Tensor(np.ones([1, 1, 2, 2]) * 2, dtype = ms.float32)
...
    net = Net1()
...
    output = net(input_x)
...
    print(output)
rank 0:
[[[1. 1.], [1. 1.], [2. 2.]]]
rank 1:
[[[1. 1.], [2. 2.], [2. 2.]]]

```

教程样例:

- 分布式集合通信原语 - NeighborExchangeV2

20.7.10 mindspore.ops.Receive

class mindspore.ops.Receive (*sr_tag, src_rank, shape, dtype, group=GlobalComm.WORLD_COMM_GROUP, group_back=GlobalComm.WORLD_COMM_GROUP*)

接收来自 *src_rank* 线程的张量。

说明: Send 和 Receive 算子需组合使用, 且需要有同一个 *sr_tag*。

参数:

- **sr_tag** (int) - 用于区分发送、接收消息的标签。该算子将接收具有相同 *sr_tag* 标签的 Send 算子发送的张量。
- **src_rank** (int) - 表示发送源的进程编号。只会接收来自源进程的张量。
- **shape** (list[int]) - 表示发送源的张量形状。
- **dtype** (Type) - 表示发送源的张量类型。所支持的类型有: int8、int16、int32、float16、float32。
- **group** (str, 可选) - 表示通信域。默认值: GlobalComm.WORLD_COMM_GROUP。
- **group_back** (str, 可选) - 表示计算反向传播时的通信域。默认值: GlobalComm.WORLD_COMM_GROUP。

输出:

- Tensor - Tensor 的 shape 与 Send 算子所发送 Tensor 的 shape 相同。

异常:

- **TypeError** - *src_rank* 不是 int 或 *group* 不是 str。
- **RuntimeError** - 如果目标设备无效, 或者后端无效, 或者分布式初始化失败。
- **ValueError** - 如果该线程的 rank id 大于通信组的 rank size。

支持平台:

Ascend GPU

样例:

说明: 运行以下样例之前, 需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备, 推荐使用 `msrun` 启动方式, 无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import os
>>> import numpy as np
>>> import mindspore.ops as ops
>>> import mindspore.nn as nn
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore import Tensor
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE, jit_level="O2")
>>> init()
>>>
>>> class SendNet(nn.Cell):
...     def __init__(self):
```

(下页继续)

(续上页)

```

...     super(SendNet, self).__init__()
...     self.depend = ops.Depend()
...     self.send = ops.Send(sr_tag=0, dest_rank=1, group="hccl_world_group")
...
...     def construct(self, x):
...         out = self.depend(x, self.send(x))
...         return out
>>>
>>> class ReceiveNet(nn.Cell):
...     def __init__(self):
...         super(ReceiveNet, self).__init__()
...         self.recv = ops.Receive(sr_tag=0, src_rank=0, shape=[2, 8], dtype=ms.float32,
...                                 group="hccl_world_group")
...
...     def construct(self):
...         out = self.recv()
...         return out
>>>
>>> if __name__ == "__main__":
...     rank_id = os.environ["RANK_ID"]
...     rank_size = os.environ["RANK_SIZE"]
...     if rank_id == "0":
...         input_ = Tensor(np.ones([2, 8]).astype(np.float32))
...         send_net = SendNet()
...         output = send_net(input_)
...     else:
...         recv_net = ReceiveNet()
...         output = recv_net()
...     print(output.asnumpy())

```

教程样例：

- 分布式集合通信原语 - Receive

20.7.11 mindspore.ops.ReduceOp

class mindspore.ops.ReduceOp

规约张量的操作选项。这是枚举类型，而不是运算符。

主要调用方法如下：

- SUM: ReduceOp.SUM.
- MAX: ReduceOp.MAX.
- MIN: ReduceOp.MIN.
- PROD: ReduceOp.PROD.

有四种操作选项，“SUM”、“MAX”、“MIN”和“PROD”。

- SUM: 求和。
- MAX: 求最大值。
- MIN: 求最小值。
- PROD: 求乘积。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import numpy as np
>>> import mindspore
>>> from mindspore.communication import init
>>> from mindspore import Tensor, ops, nn
>>> from mindspore.ops import ReduceOp
>>>
>>> init()
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.allreduce_sum = ops.AllReduce(ReduceOp.SUM)
...
...     def construct(self, x):
...         return self.allreduce_sum(x)
...
>>> input_ = Tensor(np.ones([2, 8]).astype(np.float32))
>>> net = Net()
>>> output = net(input_)
>>> print(output)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]]
```

20.7.12 mindspore.ops.ReduceScatter

class mindspore.ops.ReduceScatter (*op=ReduceOp.SUM, group=GlobalComm.WORLD_COMM_GROUP*)

规约并且分发指定通信组中的张量，返回分发后的张量。

说明：在集合的所有过程中，Tensor 必须具有相同的 shape 和格式。

参数：

- **op** (str, 可选) - 指定用于元素的规约操作，如 SUM 和 MAX。默认值：ReduceOp.SUM。
- **group** (str, 可选) - 要处理的通信组。默认值：GlobalComm.WORLD_COMM_group。

输入：

- **input_x** (Tensor) - 输入 Tensor，假设其形状为 $(N, *)$ ，其中 $*$ 为任意数量的额外维度。N 必须能够被 rank_size 整除，rank_size 为当前通讯组里面的计算卡数量。

输出：

Tensor，数据类型与 *input_x* 一致，shape 为 $(N/rank\_size, *)$ 。

异常：

- **TypeError** - 如果 *op* 和 *group* 不是字符串。
- **ValueError** - 如果输入的第一个维度不能被 rank size 整除。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.communication import init
>>> from mindspore.ops import ReduceOp
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> import numpy as np
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> init()
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.reducescatter = ops.ReduceScatter(ReduceOp.SUM)
...
...     def construct(self, x):
...         return self.reducescatter(x)
...
>>> input_ = Tensor(np.ones([8, 8]).astype(np.float32))
>>> net = Net()
>>> output = net(input_)
>>> print(output)
[[2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
 [2. 2. 2. 2. 2. 2. 2. 2.]
```

教程样例：

- 分布式集合通信原语 - `ReduceScatter`

20.7.13 mindspore.ops.Reduce

class mindspore.ops.Reduce (dest_rank, op=ReduceOp.SUM, group=GlobalComm.WORLD_COMM_GROUP)

规约指定通信组中的 Tensor，并将规约结果发送到目标为 dest_rank 的进程中，返回发送到目标进程的 Tensor。

说明：只有目标为 dest_rank 的进程（通信组的本地进程编号）才会收到规约操作后的输出。当前支持 Pynative 和 Graph 模式。但 Graph 模式只支持图编译等级为 O0 的场景。其他进程只得到一个形状为 [1] 的 Tensor，且该 Tensor 没有数学意义。

参数：

- **dest_rank** (int) - 指定接收输出的目标进程编号（通信组的本地进程编号），只有该进程会接收规约操作后的输出结果。
- **op** (str, 可选) - 规约的具体操作。如 "sum"、"prod"、"max"、和 "min"。默认值：ReduceOp.SUM。
- **group** (str, 可选) - 工作的通信组。默认值：GlobalComm.WORLD_COMM_GROUP（即 Ascend 平台为 "hccl_world_group"，GPU 平台为 "nccl_world_group"）。

输入：

- **input_x** (Tensor) - Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。

输出：

Tensor，返回规约操作后，目标进程的 Tensor。数据类型与输入的 tensor 一致，shape 为 $(x_1, x_2, \dots, x_R)$ 。

异常：

- **TypeError** - 首个输入的数据类型不为 Tensor，op 和 group 不是字符串。
- **RuntimeError** - 目标设备无效，或者后端无效，或者分布式初始化失败。

支持平台：

Ascend

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 msrcun 启动方式，无第三方以及配置文件依赖。详见 [msrcun 启动](#)。

该样例需要在 4 卡环境下运行。

```
>>> from mindspore import ops
>>> import mindspore.nn as nn
>>> from mindspore.communication import init
>>> from mindspore import Tensor
>>> import numpy as np
>>> # Launch 4 processes.
>>> init()
>>> class ReduceNet(nn.Cell):
...     def __init__(self):
...         super(ReduceNet, self).__init__()
...         self.reduce = ops.Reduce(dest_rank=1)
...
...     def construct(self, x):
```

(下页继续)

(续上页)

```

...         out = self.reduce(x)
...         return out
>>> input = Tensor(np.ones([2, 8]).astype(np.float32))
>>> net = ReduceNet()
>>> output = net(input)
>>> print(output)
Process with rank 1: [[4. 4. 4. 4. 4. 4. 4. 4.]
[4. 4. 4. 4. 4. 4. 4. 4.]],
Other proesses: [0.].

```

教程样例：

- 分布式集合通信原语 - Reduce

20.7.14 mindspore.ops.Send

class mindspore.ops.Send(*sr_tag*, *dest_rank*, *group*=GlobalComm.WORLD_COMM_GROUP,
group_back=GlobalComm.WORLD_COMM_GROUP)

发送张量到指定线程。

说明：Send 和 Receive 算子需组合使用，且需要有同一个 *sr_tag*。

参数：

- **sr_tag** (int) - 用于区分发送、接收消息的标签。该算子发送的消息将被拥有相同 *sr_tag* 的 Receive 算子接收。
- **dest_rank** (int) - 表示发送目标的进程编号。只有目标进程会收到张量。
- **group** (str, 可选) - 表示通信域。默认值：GlobalComm.WORLD_COMM_GROUP。
- **group_back** (str, 可选) - 表示计算反向传播时的通信域。默认值：GlobalComm.WORLD_COMM_GROUP。

输入：

- **input_x** (Tensor) - 输入待发送的 Tensor，Tensor 的 shape 为 $(x_1, x_2, \dots, x_R)$ 。

异常：

- **TypeError** - *dest_rank* 不是 int 或 *group* 不是 str。
- **RuntimeError** - 如果目标设备无效，或者后端无效，或者分布式初始化失败。
- **ValueError** - 如果该线程的 rank id 大于通信组的 rank size。

支持平台：

Ascend GPU

样例：

说明：运行以下样例之前，需要配置好通信环境变量。

针对 Ascend/GPU/CPU 设备，推荐使用 `msrun` 启动方式，无第三方以及配置文件依赖。详见 [msrun 启动](#)。

该样例需要在 2 卡环境下运行。

```

>>> import os
>>> import numpy as np
>>> import mindspore.ops as ops
>>> import mindspore.nn as nn
>>> import mindspore as ms
>>> from mindspore.communication import init
>>> from mindspore import Tensor
>>>
>>> ms.set_context(mode=ms.GRAPH_MODE, jit_level="O2")
>>> init()
>>>
>>> class SendNet(nn.Cell):
...     def __init__(self):
...         super(SendNet, self).__init__()
...         self.depend = ops.Depend()
...         self.send = ops.Send(sr_tag=0, dest_rank=1, group="hccl_world_group")
...
...     def construct(self, x):
...         out = self.depend(x, self.send(x))
...         return out
>>>
>>> class ReceiveNet(nn.Cell):
...     def __init__(self):
...         super(ReceiveNet, self).__init__()
...         self.recv = ops.Receive(sr_tag=0, src_rank=0, shape=[2, 8], dtype=ms.float32,
...                               group="hccl_world_group")
...
...     def construct(self):
...         out = self.recv()
...         return out
>>>
>>> if __name__ == "__main__":
...     rank_id = os.environ["RANK_ID"]
...     rank_size = os.environ["RANK_SIZE"]
...     if rank_id == "0":
...         input_ = Tensor(np.ones([2, 8]).astype(np.float32))
...         send_net = SendNet()
...         output = send_net(input_)
...     else:
...         recv_net = ReceiveNet()
...         output = recv_net()
...         print(output.asnumpy())

```

教程样例：

- 分布式集合通信原语 - Send

20.8 调试算子

接口名	概述	支持平台
<code>mindspore.ops.HistogramSummary</code>	计算 Tensor 的直方图并保存到 Summary 文件。	Ascend GPU CPU
<code>mindspore.ops.ImageSummary</code>	将图像保存到 Summary 文件。	Ascend GPU CPU
<code>mindspore.ops.ScalarSummary</code>	将标量数据保存到 Summary 文件。	Ascend GPU CPU
<code>mindspore.ops.TensorSummary</code>	将 Tensor 保存到 Summary 文件。	Ascend GPU CPU
<code>mindspore.ops.TensorDump</code>	将 Tensor 保存为 numpy 格式的 np y 文件。	Ascend
<code>mindspore.ops.Print</code>	将输入数据进行打印输出。	Ascend GPU CPU
<code>mindspore.ops.NPUAllocFloatStatus</code>	分配一个标志来存储溢出状态。	Ascend
<code>mindspore.ops.NPUClearFloatStatus</code>	清除存储溢出状态的标识。	Ascend
<code>mindspore.ops.NPUGetFloatStatus</code>	在 执 行 <code>mindspore.ops.NPUAllocFloatStatus</code> 后, <code>NPUGetFloatStatus</code> 获取最新溢出状态并更新标志。	Ascend

20.8.1 mindspore.ops.HistogramSummary

class mindspore.ops.HistogramSummary

计算 **Tensor** 的直方图并保存到 **Summary** 文件。必须和 **SummaryRecord** 或 **SummaryCollector** 一起使用, **Summary** 文件的保存路径由 **SummaryRecord** 或 **SummaryCollector** 指定。

在 Ascend 平台上的 **Graph** 模式下, 可以通过设置环境变量 `MS_DUMP_SLICE_SIZE` 和 `MS_DUMP_WAIT_TIME` 解决该算子在调用比较密集的场景下执行失败的问题。

输入:

- **name** (str) - 输入变量的名称。
- **value** (Tensor) - **Tensor** 的值, **Tensor** 的 rank 必须大于 0。

异常:

- **TypeError** - 如果 `name` 不是 str。
- **TypeError** - 如果 `value` 不是 **Tensor**。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore import Tensor, set_context
>>>
>>>
>>> class SummaryDemo(nn.Cell):
...     def __init__(self):
...         super(SummaryDemo, self).__init__()
...         self.summary = ops.HistogramSummary()
...         self.add = ops.Add()
...
...     def construct(self, x, y):
```

(下页继续)

(续上页)

```

...     x = self.add(x, y)
...     name = "x"
...     self.summary(name, x)
...     return x
>>> set_context(mode=mindspore.GRAPH_MODE)
>>> summary = SummaryDemo()(Tensor([1, 2]), Tensor([3, 4]))
>>> print(summary)
[4 6]

```

20.8.2 mindspore.ops.ImageSummary

class mindspore.ops.ImageSummary

将图像保存到 Summary 文件。该算子必须和 SummaryRecord 或 SummaryCollector 一起使用，Summary 文件的保存路径由 SummaryRecord 或 SummaryCollector 指定。

在 Ascend 平台上的 Graph 模式下，可以通过设置环境变量 `MS_DUMP_SLICE_SIZE` 和 `MS_DUMP_WAIT_TIME` 解决该算子在调用比较密集的场景下执行失败的问题。

输入：

- **name** (str) - 输入变量的名称，不能是空字符串。
- **value** (Tensor) - 图像数据的值，Tensor 的 rank 必须为 4。

异常：

- **TypeError** - 如果 *name* 不是 str。
- **TypeError** - 如果 *value* 不是 Tensor。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore.nn as nn
>>> from mindspore import ops
>>>
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.summary = ops.ImageSummary()
...
...     def construct(self, x):
...         name = "image"
...         self.summary(name, x)
...         return x
...

```

20.8.3 mindspore.ops.ScalarSummary

class mindspore.ops.ScalarSummary

将标量数据保存到 Summary 文件。必须和 `mindspore.SummaryRecord` 或 `mindspore.SummaryCollector` 一起使用，Summary 文件的保存路径由 SummaryRecord 或 SummaryCollector 指定。

在 Ascend 平台上的 Graph 模式下，可以通过设置环境变量 `MS_DUMP_SLICE_SIZE` 和 `MS_DUMP_WAIT_TIME` 解决在密集调用场景下，该算子执行失败的问题。

输入：

- **name** (str) - 输入标量的名称，不能是空字符串。
- **value** (Tensor) - 标量数据的值，维度必须为 0 或者 1。

异常：

- **TypeError** - 如果 `name` 不是 str。
- **TypeError** - 如果 `value` 不是 Tensor。
- **ValueError** - 如果 `value` 的维度大于 1。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore import Tensor, set_context
>>>
>>>
>>> class SummaryDemo(nn.Cell):
...     def __init__(self,):
...         super(SummaryDemo, self).__init__()
...         self.summary = ops.ScalarSummary()
...         self.add = ops.Add()
...
...     def construct(self, x, y):
...         name = "x"
...         self.summary(name, x)
...         x = self.add(x, y)
...         return x
>>> set_context(mode=mindspore.GRAPH_MODE)
>>> summary = SummaryDemo()(Tensor(3), Tensor(4))
>>> print(summary)
7
```

20.8.4 mindspore.ops.TensorSummary

class mindspore.ops.TensorSummary

将 Tensor 保存到 Summary 文件。必须和 SummaryRecord 或 SummaryCollector 一起使用，Summary 文件的保存路径由 SummaryRecord 或 SummaryCollector 指定。

在 Ascend 平台上的 Graph 模式下，可以通过设置环境变量 `MS_DUMP_SLICE_SIZE` 和 `MS_DUMP_WAIT_TIME` 解决该算子在调用比较密集的场景下执行失败的问题。

输入：

- **name** (str) - 输入变量的名称。
- **value** (Tensor) - Tensor 的值，Tensor 的维度必须大于 0。

异常：

- **TypeError** - 如果 *name* 不是 str。
- **TypeError** - 如果 *value* 不是 Tensor。
- **ValueError** - 如果 *value* 的维度等于 0。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore import Tensor, set_context
>>>
>>>
>>> class SummaryDemo(nn.Cell):
...     def __init__(self,):
...         super(SummaryDemo, self).__init__()
...         self.summary = ops.TensorSummary()
...         self.add = ops.Add()
...
...     def construct(self, x, y):
...         x = self.add(x, y)
...         name = "x"
...         self.summary(name, x)
...         return x
>>> set_context(mode=mindspore.GRAPH_MODE)
>>> summary = SummaryDemo()(Tensor([[1]]), Tensor([[2]]))
>>> print(summary)
[[3]]
```

20.8.5 mindspore.ops.TensorDump

class mindspore.ops.**TensorDump** (*input_output='out'*)
将 Tensor 保存为 numpy 格式的 npy 文件。

警告：参数 `input_output` 将不再支持参数值为'all'。

说明：在 Ascend 平台上的 Graph 模式下，可以通过设置环境变量 `MS_DUMP_SLICE_SIZE` 和 `MS_DUMP_WAIT_TIME` 解决在输出大 Tensor 或输出 Tensor 比较密集的场景下算子执行失败的问题。

参数：

- **input_output** (str, 可选) - 控制 TensorDump 行为模式的参数，可选的值为 ['out', 'in'] 中的一个，默认值：out。

对于算子 A → 重排算子 → 算子 B 这样的情况，由于重排算子的插入，导致算子 A 的输出不再等价于算子 B 的输入。

假设一种情况是算子 A 的计算结果既作为算子 B 的输入，也作为 TensorDump 算子的输入。则在该情况下，通过设置 `input_output` 参数可以实现不同的保存数据的需求：

- 如果 `input_output` 参数设置为 'out'，保存的数据仅包含算子 A 的输出分片。
- 如果 `input_output` 参数设置为 'in'，保存的数据将仅包含算子 B 的输入分片。

当 `input_output` 参数被配置为 'in' 时，生成的输入分片所对应的 npy 文件命名格式为：file-Name_dumpMode_dtype_id.npy。

当 `input_output` 参数被配置为 'out' 时，生成的输出分片所对应的 npy 文件命名格式为：fileName_dtype_id.npy。

- **fileName**：参数 `file` 的值（若该参数传入时是一个使用者指定的路径，则 `fileName` 的值为路径的最后一级）。
- **dumpMode**：`input_output` 参数的值。
- **dtype**：原始的数据类型。bfloat16 类型数据保存在 .npy 文件中会被转换成 float32 类型。
- **id**：一个自增的 ID。

输入：

- **file** (str) - 要保存的文件路径。
- **input_x** (Tensor) - 任意维度的 Tensor。

异常：

- **TypeError** - 如果 `file` 不是 str。
- **TypeError** - 如果 `input_x` 不是 Tensor。

支持平台：

Ascend

样例：

```

>>> import numpy as np
>>> import mindspore as ms
>>> import time
>>> from mindspore import nn, Tensor, ops
>>> ms.set_context(mode=ms.GRAPH_MODE)
>>> ms.set_device(device_target="Ascend")
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.dump = ops.TensorDump()
...
...     def construct(self, x):
...         x += 1.
...         self.dump('add', x)
...         x /= 2.
...         self.dump('div', x)
...         x *= 5.
...         self.dump('mul', x)
...         return x
...
>>> x = np.array([[1, 2, 3, 4], [5, 6, 7, 8]]).astype(np.float32)
>>> input_x = Tensor(x)
>>> net = Net()
>>> out = net(input_x)
>>> time.sleep(0.5)
>>> add = np.load('add_float32_0.npy')
>>> print(add)
[[2. 3. 4. 5.]
 [6. 7. 8. 9.]]

```

20.8.6 mindspore.ops.Print

class mindspore.ops.Print

将输入数据进行打印输出。

更多参考详见 `mindspore.ops.print_()`。

输入：

- **input_x** (Union[Tensor, bool, int, float, str]) - Print 的输入。支持多个输入，用‘,’分隔。

输出：

Tensor，数据类型和 shape 与 `input_x` 相同。

支持平台：

Ascend GPU CPU

样例：

```

>>> import numpy as np
>>> from mindspore import Tensor, nn, ops
>>> class PrintDemo(nn.Cell):
...     def __init__(self):
...         super(PrintDemo, self).__init__()
...         self.print = ops.Print()
...

```

(下页继续)

(续上页)

```

...     def construct(self, x, y):
...         self.print('Print Tensor x and Tensor y:', x, y)
...         return x
...
>>> x = Tensor(np.ones([2, 1]).astype(np.int32))
>>> y = Tensor(np.ones([2, 2]).astype(np.int32))
>>> net = PrintDemo()
>>> result = net(x, y)
Print Tensor x and Tensor y:
Tensor(shape=[2, 1], dtype=Int32, value=
[[1],
 [1]])
Tensor(shape=[2, 2], dtype=Int32, value=
[[1, 1],
 [1, 1]])

```

20.8.7 mindspore.ops.NPUAllocFloatStatus

class mindspore.ops.NPUAllocFloatStatus

分配一个标志来存储溢出状态。

标志是一个 Tensor，其 shape 为 (8,)，数据类型为 *mindspore.dtype.float32*。

说明：请参考 *mindspore.ops.NPUGetFloatStatus* 的样例。

输出：

Tensor, shape 为 (8,)。

支持平台：

Ascend

样例：

```

>>> from mindspore import ops
>>> alloc_status = ops.NPUAllocFloatStatus()
>>> output = alloc_status()
>>> print(output)
[0. 0. 0. 0. 0. 0. 0. 0.]

```

20.8.8 mindspore.ops.NPUClearFloatStatus

class mindspore.ops.NPUClearFloatStatus

清除存储溢出状态的标识。

说明：该标志位于 *Ascend* 设备上的寄存器中。在调用 *NPUClearFloatStatus* 后，它将被重置，不能再次重用。此外，使用有严格的使用顺序要求，即在使用 *mindspore.ops.NPUGetFloatStatus* 算子之前，需要确保 *NPUClearFloatStatus* 和需执行的计算已执行。我们使用 *mindspore.ops.Depend* 确保执行顺序。

请参考 *mindspore.ops.NPUGetFloatStatus* 的样例。

输入:

- **x** (Tensor) - `mindspore.ops.NPUAllocFloatStatus` 的输出 Tensor。数据类型必须为 float16 或 float32。

输出:

Tensor, shape 与 *x* 相同。Tensor 中的所有元素都将为零。

支持平台:

Ascend

样例:

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor
>>> class Net(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.alloc_status = ops.NPUAllocFloatStatus()
...         self.get_status = ops.NPUGetFloatStatus()
...         self.clear_status = ops.NPUClearFloatStatus()
...         self.sub = ops.Sub()
...         self.neg = ops.Neg()
...
...     def construct(self, x):
...         init = self.alloc_status()
...         clear_status = self.clear_status(init)
...         x = ops.depend(x, clear_status)
...         res = self.sub(x, self.neg(x))
...         init = ops.depend(init, res)
...         get_status = self.get_status(init)
...         res = ops.depend(res, get_status)
...         return res
>>>
>>> value = 5
>>> data = np.full((2, 3), value, dtype=np.float16)
>>> x = Tensor(data, dtype=mstype.float16)
>>> net = Net()
>>> res = net(x)
>>> print(res)
[[10. 10. 10.]
 [10. 10. 10.]]
```

20.8.9 mindspore.ops.NPUGetFloatStatus

class mindspore.ops.NPUGetFloatStatus

在执行 `mindspore.ops.NPUAllocFloatStatus` 后, `NPUGetFloatStatus` 获取最新溢出状态并更新标志。

说明: 标志是一个 Tensor, 其 shape 为 (8,), 数据类型为 `mindspore.dtype.float32`。如果标志的和等于 0, 则没有发生溢出。如果标志之和大于 0, 则发生溢出。此外, 算子使用有严格的顺序要求, 即在使用 `NPUGetFloatStatus` 算子之前, 需要确保 `mindspore.ops.NPUClearFloatStatus` 和需执行的计算已执行。使用 `mindspore.ops.Depend` 确保执行顺序正确。

输入：

- **x (Tensor)** - `mindspore.ops.NPUAllocFloatStatus` 的输出 Tensor。数据类型必须为 float16 或 float32。
(N,*), 其中 * 表示任意附加维度, 其 rank 应小于 8。

输出：

Tensor, shape 与 *x* 相同。

异常：

- **TypeError** - 如果 *x* 不是 Tensor。
- **TypeError** - 如果 *x* 的数据类型既不是 float16 也不是 float32。

支持平台：

Ascend

样例：

```
>>> import numpy as np
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor
>>> class Net(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.alloc_status = ops.NPUAllocFloatStatus()
...         self.get_status = ops.NPUGetFloatStatus()
...         self.clear_status = ops.NPUClearFloatStatus()
...         self.sub = ops.Sub()
...         self.neg = ops.Neg()
...
...     def construct(self, x):
...         init = self.alloc_status()
...         clear_status = self.clear_status(init)
...         x = ops.depend(x, clear_status)
...         res = self.sub(x, self.neg(x))
...         init = ops.depend(init, res)
...         get_status = self.get_status(init)
...         res = ops.depend(res, get_status)
...         return res
>>>
>>> value = 5
>>> data = np.full((2, 3), value, dtype=np.float16)
>>> x = Tensor(data, dtype=mstype.float16)
>>> net = Net()
>>> res = net(x)
>>> print(res)
[[10. 10. 10.]
 [10. 10. 10.]]
```


20.9 稀疏算子

接口名	概述	支持平台
<code>mindspore.ops.SparseTensorDenseMatmul</code>	稀疏矩阵 A 乘以稠密矩阵 B 。	GPU CPU
<code>mindspore.ops.SparseToDense</code>	将稀疏 Tensor 转换为密集 Tensor。	CPU

20.9.1 mindspore.ops.SparseTensorDenseMatmul

`class mindspore.ops.SparseTensorDenseMatmul (adjoint_st=False, adjoint_dt=False)`

稀疏矩阵 A 乘以稠密矩阵 B 。稀疏矩阵和稠密矩阵的秩必须等于 2。

参数：

- **adjoint_st** (bool) - 如果为 True，则在乘法之前转置稀疏矩阵 A 。默认值：False。
- **adjoint_dt** (bool) - 如果为 True，则在乘法之前转置稠密矩阵 B 。默认值：False。

输入：

- **indices** (Tensor) - 二维 Tensor，表示元素在稀疏 Tensor 中的位置。支持 int32、int64，每个元素值都应该都是非负的。shape 是 $(n, 2)$ 。
- **values** (Tensor) - 一维 Tensor，表示 *indices* 位置上对应的值。支持 float16、float32、float64、int32、int64、complex64、complex128。shape 是 $(n,)$ 。
- **sparse_shape** (tuple(int) 或 Tensor) - 指定稀疏 Tensor 的 shape，由两个正整数组成，表示稀疏 Tensor 的 shape 为 (N, C) 。
- **dense** (Tensor) - 二维 Tensor，数据类型与 *values* 相同。
 - 如果 *adjoint_st* 为 False，*adjoint_dt* 为 False，则 shape 必须为 (C, M) 。
 - 如果 *adjoint_st* 为 False，*adjoint_dt* 为 True，则 shape 必须为 (M, C) 。
 - 如果 *adjoint_st* 为 True，*adjoint_dt* 为 False，则 shape 必须为 (N, M) 。
 - 如果 *adjoint_st* 为 True，*adjoint_dt* 为 True，则 shape 必须为 (M, N) 。

输出：

Tensor，数据类型与 *values* 相同。如果 *adjoint_st* 为 False，则 shape 为 (N, M) 。如果 *adjoint_st* 为 True，则 shape 为 (C, M) 。

异常：

- **TypeError** - 如果 *adjoint_st* 或 *adjoint_dt* 的数据类型不是 bool，或者 *indices*、*values*、*dense* 的数据类型不符合参数描述中支持的数据类型。
- **ValueError** - 如果 *sparse_shape*、*indices*、*values* 和 *dense* 的 shape 不满足参数描述中的约束条件。

支持平台：

GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor
>>> from mindspore.ops import operations as ops
>>> from mindspore import dtype as mstype
```

(下页继续)

(续上页)

```
>>> indices = Tensor([[0, 1], [1, 2]], dtype=mindspore.int32)
>>> values = Tensor([1, 2], dtype=mindspore.float32)
>>> sparse_shape = (3, 4)
>>> dense = Tensor([[1, 1], [2, 2], [3, 3], [4, 4]], dtype=mindspore.float32)
>>> sparse_dense_matmul = ops.SparseTensorDenseMatmul()
>>> out = sparse_dense_matmul(indices, values, sparse_shape, dense)
>>> print(out)
[[2. 2.]
 [6. 6.]
 [0. 0.]]
```

20.9.2 mindspore.ops.SparseToDense

class mindspore.ops.SparseToDense

将稀疏 Tensor 转换为密集 Tensor。

输入：

- **indices** (Tensor) - 二维 Tensor，表示元素在稀疏 Tensor 中的位置。支持 int32、int64，每个元素值都应该是非负的。shape 是 $(n, 2)$ 。
- **values** (Tensor) - 一维 Tensor，表示 *indices* 位置上对应的值。shape 为 $(n,)$ 。
- **sparse_shape** (tuple(int)) - 指定稀疏 Tensor 的 shape，由两个正整数组成，表示稀疏 Tensor 的 shape 为 (N, C) 。

输出：

Tensor，计算后的 Tensor。数据类型与 *values* 相同，shape 由 *sparse_shape* 所指定。

异常：

- **TypeError** - 如果 *indices* 的数据类型既不是 int32 也不是 int64。
- **ValueError** - 如果 *sparse_shape*、*indices* 和 *values* 的 shape 不符合参数中所描述支持的数据类型。

支持平台：

CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> indices = Tensor([[0, 1], [1, 2]])
>>> values = Tensor([1, 2], dtype=mindspore.float32)
>>> sparse_shape = (3, 4)
>>> sparse_to_dense = ops.SparseToDense()
>>> out = sparse_to_dense(indices, values, sparse_shape)
>>> print(out)
[[0. 1. 0. 0.]
 [0. 0. 2. 0.]
 [0. 0. 0. 0.]]
```

20.10 框架算子

接口名	概述	支持平台
<code>mindspore.ops.Depend</code>	用来处理操作间的依赖关系。	Ascend GPU CPU
<code>mindspore.ops.ForiLoop</code>	在指定范围内执行循环操作。	Ascend GPU CPU
<code>mindspore.ops.GradOperation</code>	一个高阶函数，为输入函数生成梯度函数。	Ascend GPU CPU
<code>mindspore.ops.HookBackward</code>	用来导出中间变量中的梯度。	Ascend GPU CPU
<code>mindspore.ops.HyperMap</code>	对输入序列做集合运算。	Ascend GPU CPU
<code>mindspore.ops.InsertGradientOf</code>	为图节点附加回调函数，将在梯度计算时被调用。	Ascend GPU CPU
<code>mindspore.ops.Morph</code>	<i>Morph</i> 算子用于对用户自定义函数 <i>fn</i> 进行封装，允许其被当做自定义算子使用。	
<code>mindspore.ops.Map</code>	<i>Map</i> 将对输入序列应用设置的函数操作。	Ascend GPU CPU
<code>mindspore.ops.MultitypeFuncGraph</code>	<i>MultitypeFuncGraph</i> 是一个用于生成重载函数的类，使用不同类型作为输入。	Ascend GPU CPU
<code>mindspore.ops.Partial</code>	生成偏函数的实例。	Ascend GPU CPU
<code>mindspore.ops.Scan</code>	将一个函数循环作用于一个数组，且对当前元素的处理依赖上一个元素的执行结果。	Ascend GPU CPU
<code>mindspore.ops.WhileLoop</code>	在编译阶段不进行循环展开的循环算子。	Ascend GPU CPU

20.10.1 mindspore.ops.Depend

class mindspore.ops.Depend

用来处理操作间的依赖关系。

在大多数情况下，如果操作有作用在 IO 或内存上的副作用，它们将按照用户的指令依序执行。在某些情况下，如果两个操作 A 和 B 没有顺序上的依赖性，而 A 必须在 B 之前执行，我们建议使用 **Depend** 来指定它们的执行顺序。使用方法如下：

```
a = A(x)          --->      a = A(x)
b = B(y)          --->      y = Depend(y, a)
                   --->      b = B(y)
```

输入：

- **value** (Tensor) - 应被 **Depend** 操作符返回的 Tensor。
- **expr** (Expression) - 应被执行的无输出的表达式。

输出：

Tensor，作为 *value* 传入的变量。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> from mindspore import Tensor
```

(下页继续)

(续上页)

```

>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.softmax = ops.Softmax()
...         self.depend = ops.Depend()
...
...     def construct(self, x, y):
...         mul = x * y
...         y = self.depend(y, mul)
...         ret = self.softmax(y)
...         return ret
...
>>> x = Tensor(np.ones([4, 5]), dtype=mindspore.float32)
>>> y = Tensor(np.ones([4, 5]), dtype=mindspore.float32)
>>> net = Net()
>>> output = net(x, y)
>>> print(output)
[[0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]
 [0.2 0.2 0.2 0.2 0.2]]

```

20.10.2 mindspore.ops.ForilLoop

class mindspore.ops.ForilLoop

在指定范围内执行循环操作。ForilLoop 算子的执行逻辑可以近似表示为如下代码:

```

def ForilLoop(lower, upper, loop_func, init_val):
    for i in range(lower, upper):
        init_val = loop_func(i, init_val)
    return init_val

```

当前 ForilLoop 算子存在以下语法限制:

- 暂不支持 `loop_func` 为副作用函数, 例如对 `Parameter`、全局变量的修改等操作。
- 暂不支持 `loop_func` 的返回值与初始值 `init_val` 的类型或形状不同。
- 暂不支持负数或自定义增量。

警告: 这是一个实验性 API, 后续可能修改或删除。

输入:

- **lower** (Union[int, Tensor]) - 循环的起始索引值。
- **upper** (Union[int, Tensor]) - 循环的结束索引值。
- **loop_func** (Function) - 循环体函数。
- **init_val** (Union[Tensor, number, str, bool, list, tuple, dict]) - 循环的初始值。
- **unroll** (bool, 可选) - 是否在编译阶段展开, 只在循环次数确定的情况下有效。默认值: `True`。

输出:

Union[Tensor, number, str, bool, list, tuple, dict], 循环的最终结果, 和 `init_val` 的类型和形状相同。

异常：

- **TypeError** - *lower* 不是一个整数或者 Tensor。
- **TypeError** - *upper* 不是一个整数或者 Tensor。
- **TypeError** - *loop_func* 不是一个函数。
- **ValueError** - *loop_func* 不能接受索引值和 *init_val* 作为参数，或者输出值和 *init_val* 的类型或形状不同。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import ops
>>> def cumsum(index, res):
...     return index + res
...
>>> result_init = 0
>>> fori_loop = ops.ForiLoop()
>>> result = fori_loop(0, 4, cumsum, result_init)
>>> print(result)
6
```

20.10.3 mindspore.ops.GradOperation

class mindspore.ops.GradOperation (*get_all=False, get_by_list=False, sens_param=False*)

一个高阶函数，为输入函数生成梯度函数。

由 *GradOperation* 高阶函数生成的梯度函数可以通过构造参数自定义。

例如构建一个以 *x* 和 *y* 为输入的函数 *net = Net()*，并带有一个参数 *z*，详见样例中的 *Net*。

- 用于对输入求导：
 1. 返回第一个输入的梯度（见样例中的 *GradNetWrtX*）：
 - 1) 构建一个带有默认参数的 *GradOperation* 高阶函数：*grad_op = GradOperation()*。
 - 2) 将 *net* 作为参数调用 *grad_op*，得到梯度函数：*gradient_function = grad_op(net)*。
 - 3) 用 *net* 的输入作为参数调用梯度函数，得到第一个输入的梯度：*grad_op(net)(x, y)*。
 2. 返回所有输入的梯度（见样例中的 *GradNetWrtXY*）：
 - 1) 构造一个带有 *get_all=True* 参数的 *GradOperation* 高阶函数，表示获得在样例中 *Net()* 中的 *x* 和 *y* 所有输入的梯度：*grad_op = GradOperation(get_all=True)*。
 - 2) 将 *net* 作为参数调用 *grad_op*，得到梯度函数：*gradient_function = grad_op(net)*。
 - 3) 用 *net* 的输入作为参数调用梯度函数，得到所有输入的梯度：*gradient_function(x, y)*。
- 用于对参数求导：

返回给定参数的梯度（见样例中的 *GradNetWithWrtParams*）：

 1. 构造一个带有 *get_by_list=True* 参数的 *GradOperation* 高阶函数：*grad_op = GradOperation(get_by_list=True)*。
 2. 当构建 *GradOperation* 高阶函数时，创建一个 *ParameterTuple* 和 *net* 作为参数输入，*ParameterTuple* 作为参数过滤器决定返回哪个梯度：*params = ParameterTuple(net.trainable_params())*。
 3. 将 *net* 和 *params* 作为参数输入 *grad_op*，得到梯度函数：*gradient_function = grad_op(net, params)*。

4. 用 *net* 的输入作为参数调用梯度函数，得到关于给定参数的梯度：*gradient_function(x, y)*。
- 用于同时对输入和参数求导：
以 *((dx, dy), (dz))* 的格式返回关于所有输入和给定参数的梯度（见样例中的 *GradNetWrtInputsAndParams*）：
 1. 构建一个带有 *get_all=True* 和 *get_by_list=True* 参数的 *GradOperation* 高阶函数：*grad_op = GradOperation(get_all=True, get_by_list=True)*。
 2. 当构建 *GradOperation* 高阶函数时，创建一个 *ParameterTuple* 和 *net* 作为参数输入：*params = ParameterTuple(net.trainable_params())*。
 3. 将 *net* 和 *params* 作为参数输入 *grad_op*，得到梯度函数：*gradient_function = grad_op(net, params)*。
 4. 用 *net* 的输入作为参数调用梯度函数，得到关于所有输入和给定参数的梯度：*gradient_function(x, y)*。
 - 我们可以设置 *sens_param* 等于 *True* 来配置灵敏度（关于输出的梯度），向梯度函数传递一个额外的灵敏度输入值。这个输入值必须与 *net* 的输出具有相同的形状和类型（见样例中的 *GradNetWrtXYWithSensParam*）。
 1. 构建一个带有 *get_all=True* 和 *sens_param=True* 参数的 *GradOperation* 高阶函数：*grad_op = GradOperation(get_all=True, sens_param=True)*。
 2. 当 *sens_param=True*，定义 *grad_wrt_output*（关于输出的梯度）：*grad_wrt_output = Tensor(np.ones([2, 2]).astype(np.float32))*。
 3. 用 *net* 作为参数输入 *grad_op*，得到梯度函数：*gradient_function = grad_op(net)*。
 4. 用 *net* 的输入和 *sens_param* 作为参数调用梯度函数，得到关于所有输入的梯度：*gradient_function(x, y, grad_wrt_output)*。

说明：对于上面产生的梯度函数，其返回值会因返回梯度的数量不同而出现差异：

- 如果仅有一个梯度结果，返回单独值；
- 如果有多个梯度结果，返回 *tuple*；
- 如果没有任何梯度结果，返回空 *tuple*。

参数：

- **get_all** (bool) - 计算梯度，如果等于 *False*，获得第一个输入的梯度，如果等于 *True*，获得所有输入的梯度。默认值：*False*。
- **get_by_list** (bool) - 如果 *get_all* 和 *get_by_list* 都等于 *False*，则得到第一个输入的梯度。如果等于 *True*，获得所有 *Parameter* 自由变量的梯度。如果 *get_all* 和 *get_by_list* 都等于 *True*，则同时得到输入和 *Parameter* 自由变量的梯度，输出形式为（“所有输入的梯度”，“所有 *Parameter* 自由变量的梯度”）。默认值：*False*。
- **sens_param** (bool) - 是否在输入中配置灵敏度（关于输出的梯度）。如果 *sens_param* 等于 *False*，自动添加一个 *ones_like(output)* 灵敏度。如果 *sens_param* 等于 *True*，灵敏度（输出的梯度），必须通过位置参数或键值对参数来传递，如果是通过键值对参数传递 *value*，那么 *key* 必须为 *sens*。默认值：*False*。

返回：

将一个函数作为参数，并返回梯度函数的高阶函数。

异常：

- **TypeError** - 如果 *get_all*、*get_by_list* 或者 *sens_param* 不是 *bool*。

支持平台：

Ascend GPU CPU

样例：

```

>>> import mindspore
>>> import numpy as np
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops, nn, Parameter
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.matmul = ops.MatMul()
...         self.z = Parameter(Tensor(np.array([1.0], np.float32)), name='z')
...     def construct(self, x, y):
...         x = x * self.z
...         out = self.matmul(x, y)
...         return out
...
>>> class GradNetWrtX(nn.Cell):
...     def __init__(self, net):
...         super(GradNetWrtX, self).__init__()
...         self.net = net
...         self.grad_op = ops.GradOperation()
...     def construct(self, x, y):
...         gradient_function = self.grad_op(self.net)
...         return gradient_function(x, y)
...
>>> x = Tensor([[0.5, 0.6, 0.4], [1.2, 1.3, 1.1]], dtype=mstype.float32)
>>> y = Tensor([[0.01, 0.3, 1.1], [0.1, 0.2, 1.3], [2.1, 1.2, 3.3]], dtype=mstype.float32)
>>> output = GradNetWrtX(Net())(x, y)
>>> print(output)
[[1.4100001 1.5999999 6.6      ]
 [1.4100001 1.5999999 6.6      ]]
>>>
>>> class GradNetWrtXY(nn.Cell):
...     def __init__(self, net):
...         super(GradNetWrtXY, self).__init__()
...         self.net = net
...         self.grad_op = ops.GradOperation(get_all=True)
...     def construct(self, x, y):
...         gradient_function = self.grad_op(self.net)
...         return gradient_function(x, y)
...
>>>
>>> x = Tensor([[0.8, 0.6, 0.2], [1.8, 1.3, 1.1]], dtype=mstype.float32)
>>> y = Tensor([[0.1, 3.3, 1.1], [1.1, 0.2, 1.4], [1.1, 2.2, 0.3]], dtype=mstype.float32)
>>> output = GradNetWrtXY(Net())(x, y)
>>> print(output)
(Tensor(shape=[2, 3], dtype=Float32, value=
[[ 4.50000000e+00,  2.70000005e+00,  3.60000014e+00],
 [ 4.50000000e+00,  2.70000005e+00,  3.60000014e+00]]), Tensor(shape=[3, 3], dtype=Float32,
↪value=
[[ 2.59999990e+00,  2.59999990e+00,  2.59999990e+00],
 [ 1.89999998e+00,  1.89999998e+00,  1.89999998e+00],
 [ 1.30000007e+00,  1.30000007e+00,  1.30000007e+00]]))
>>>
>>> class GradNetWrtXYWithSensParam(nn.Cell):
...     def __init__(self, net):
...         super(GradNetWrtXYWithSensParam, self).__init__()
...         self.net = net
...         self.grad_op = ops.GradOperation(get_all=True, sens_param=True)
...         self.grad_wrt_output = Tensor([[0.1, 0.6, 0.2], [0.8, 1.3, 1.1]], dtype=mstype.
↪float32)

```

(下页继续)

(续上页)

```

...     def construct(self, x, y):
...         gradient_function = self.grad_op(self.net)
...         return gradient_function(x, y, self.grad_wrt_output)
>>>
>>> x = Tensor([[0.8, 0.6, 0.2], [1.8, 1.3, 1.1]], dtype=mstype.float32)
>>> y = Tensor([[0.11, 3.3, 1.1], [1.1, 0.2, 1.4], [1.1, 2.2, 0.3]], dtype=mstype.float32)
>>> output = GradNetWrtXYWithSensParam(Net())(x, y)
>>> print(output)
(Tensor(shape=[2, 3], dtype=Float32, value=
[[ 2.21099997e+00,  5.09999990e-01,  1.49000001e+00],
 [ 5.58800030e+00,  2.68000007e+00,  4.07000017e+00]]), Tensor(shape=[3, 3], dtype=Float32,
↪value=
[[ 1.51999998e+00,  2.81999993e+00,  2.14000010e+00],
 [ 1.09999990e+00,  2.04999995e+00,  1.54999995e+00],
 [ 9.00000036e-01,  1.54999995e+00,  1.25000000e+00]]))
>>>
>>> class GradNetWithWrtParams(nn.Cell):
...     def __init__(self, net):
...         super(GradNetWithWrtParams, self).__init__()
...         self.net = net
...         self.params = ParameterTuple(net.trainable_params())
...         self.grad_op = ops.GradOperation(get_by_list=True)
...     def construct(self, x, y):
...         gradient_function = self.grad_op(self.net, self.params)
...         return gradient_function(x, y)
>>>
>>> x = Tensor([[0.8, 0.6, 0.2], [1.8, 1.3, 1.1]], dtype=mstype.float32)
>>> y = Tensor([[0.11, 3.3, 1.1], [1.1, 0.2, 1.4], [1.1, 2.2, 0.3]], dtype=mstype.float32)
>>> output = GradNetWithWrtParams(Net())(x, y)
>>> print(output)
(Tensor(shape=[1], dtype=Float32, value= [ 2.15359993e+01]),)
>>>
>>> class GradNetWrtInputsAndParams(nn.Cell):
...     def __init__(self, net):
...         super(GradNetWrtInputsAndParams, self).__init__()
...         self.net = net
...         self.params = ParameterTuple(net.trainable_params())
...         self.grad_op = ops.GradOperation(get_all=True, get_by_list=True)
...     def construct(self, x, y):
...         gradient_function = self.grad_op(self.net, self.params)
...         return gradient_function(x, y)
>>>
>>> x = Tensor([[0.1, 0.6, 1.2], [0.5, 1.3, 0.1]], dtype=mstype.float32)
>>> y = Tensor([[0.12, 2.3, 1.1], [1.3, 0.2, 2.4], [0.1, 2.2, 0.3]], dtype=mstype.float32)
>>> output = GradNetWrtInputsAndParams(Net())(x, y)
>>> print(output)
((Tensor(shape=[2, 3], dtype=Float32, value=
[[ 3.51999998e+00,  3.90000010e+00,  2.59999990e+00],
 [ 3.51999998e+00,  3.90000010e+00,  2.59999990e+00]]), Tensor(shape=[3, 3], dtype=Float32,
↪value=
[[ 6.00000024e-01,  6.00000024e-01,  6.00000024e-01],
 [ 1.89999998e+00,  1.89999998e+00,  1.89999998e+00],
 [ 1.30000007e+00,  1.30000007e+00,  1.30000007e+00]]), (Tensor(shape=[1], dtype=Float32,
↪value=
[ 1.29020004e+01]),))

```


20.10.4 mindspore.ops.HookBackward

class mindspore.ops.HookBackward(hook_fn, cell_id="")

用来导出中间变量中的梯度。请注意，此函数仅在 PyNative 模式下支持。

说明：钩子函数必须定义为 *hook_fn(grad) -> new gradient or None*，其中'grad' 是传递给 Primitive 的梯度。可以通过返回新的梯度并传递到下一个 Primitive 来修改'grad'。钩子函数和 InsertGradientOf 的回调的区别在于，钩子函数是在 python 环境中执行的，而回调将被解析并添加到图中。

参数：

- **hook_fn** (Function) - Python 函数。钩子函数。
- **cell_id** (str, 可选) - 用于标识钩子注册的函数是否实际注册在指定的 cell 对象上。例如，*mindspore.nn.Conv2d* 是一个 cell 对象。默认值：""，此情况下系统将自动注册 cell_id 的值。此参数目前不支持自定义。

输入：

- **input** (Tensor) - 需要导出的变量的梯度。

输出：

- **output** (Tensor) - 直接返回 *input*。*HookBackward* 不影响前向结果。

异常：

- **TypeError** - 如果 *input* 不是 Tensor。
- **TypeError** - 如果 *hook_fn* 不是 Python 的函数。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore as ms
>>> from mindspore import ops
>>> from mindspore import Tensor
>>> from mindspore.ops import GradOperation
>>> ms.set_context(mode=ms.PYNATIVE_MODE)
>>> def hook_fn(grad):
...     print(grad)
...
>>> hook = ops.HookBackward(hook_fn)
>>> def hook_test(x, y):
...     z = x * y
...     z = hook(z)
...     z = z * y
...     return z
...
>>> grad_all = GradOperation(get_all=True)
>>> def backward(x, y):
...     return grad_all(hook_test)(x, y)
...
>>> output = backward(Tensor(1, ms.float32), Tensor(2, ms.float32))
(Tensor(shape=[], dtype=Float32, value= 2),)
```

(下页继续)

(续上页)

```
>>> print(output)
(Tensor(shape=[], dtype=Float32, value= 4), Tensor(shape=[], dtype=Float32, value= 4))
```

20.10.5 mindspore.ops.HyperMap

class mindspore.ops.HyperMap(*ops=None, reverse=False*)

对输入序列做集合运算。

对序列的每个元素或嵌套序列进行运算。与 `mindspore.ops.Map` 不同, `HyperMap` 能够用于嵌套结构。

参数:

- **ops** (Union[MultitypeFuncGraph, None], 可选) - *ops* 是指定运算操作。如果 *ops* 为 None, 则运算应该作为 `HyperMap` 实例的第一个入参。默认值为 None。
- **reverse** (bool, 可选) - 在某些场景下, 需要逆向以提高计算的并行性能, 一般情况下, 用户可以忽略。 *reverse* 用于决定是否逆向执行运算, 仅在图模式下支持。默认值为 False。

输入:

- **args** (Tuple[sequence]) -
 - 如果 *ops* 不是 None, 则所有入参都应该具有相同长度的序列, 并且序列的每一行都是运算的输入。
 - 如果 *ops* 是 None, 则第一个入参是运算, 其余都是输入。

说明: 输入数量等于 *ops* 的输入数量。

输出:

序列或嵌套序列, 执行 `operation(args[0][i], args[1][i])` 之后输出的序列, 其中 *operation* 为 *ops* 指定的一个函数。

异常:

- **TypeError** - 如果 *ops* 既不是 `mindspore.ops.MultitypeFuncGraph` 也不是 None。
- **TypeError** - 如果 *args* 不是一个 tuple。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> nest_tensor_list = ((Tensor(1, mstype.float32), Tensor(2, mstype.float32)),
...                     (Tensor(3, mstype.float32), Tensor(4, mstype.float32)))
>>> # square all the tensor in the nested list
>>>
>>> square = ops.MultitypeFuncGraph('square')
>>> @square.register("Tensor")
... def square_tensor(x):
...     return ops.square(x)
>>>
>>> common_map = ops.HyperMap()
>>> output = common_map(square, nest_tensor_list)
```

(下页继续)

(续上页)

```
>>> print(output)
((Tensor(shape=[], dtype=Float32, value= 1), Tensor(shape=[], dtype=Float32, value= 4)),
 (Tensor(shape=[], dtype=Float32, value= 9), Tensor(shape=[], dtype=Float32, value= 16)))
>>> square_map = ops.HyperMap(square, False)
>>> output = square_map(nest_tensor_list)
>>> print(output)
((Tensor(shape=[], dtype=Float32, value= 1), Tensor(shape=[], dtype=Float32, value= 4)),
 (Tensor(shape=[], dtype=Float32, value= 9), Tensor(shape=[], dtype=Float32, value= 16)))
```

20.10.6 mindspore.ops.InsertGradientOf

class mindspore.ops.InsertGradientOf(*f*)

为图节点附加回调函数，将在梯度计算时被调用。

参数：

- **f** (Function) - MindSpore Function。回调函数。

输入：

- **input_x** (Any) - 需要附加回调函数的图节点。

输出：

Tensor，直接返回输入 *input_x*。该算子不影响前向计算的结果。

异常：

- **TypeError** - *f* 不是 MindSpore Function。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, ops, jit
>>> a = Tensor(np.array([1.0]).astype(np.float32))
>>> b = Tensor(np.array([0.2]).astype(np.float32))
>>> def clip_gradient(dx):
...     ret = dx
...     if ret > a:
...         ret = a
...
...     if ret < b:
...         ret = b
...
...     return ret
...
>>> clip = ops.InsertGradientOf(clip_gradient)
>>> grad_all = ops.GradOperation(get_all=True)
>>> def InsertGradientOfClipDemo():
...     def clip_test(x, y):
...         x = clip(x)
...         y = clip(y)
...         c = x * y
...         return c
```

(下页继续)

```

...
... @jit
... def f(x, y):
...     return clip_test(x, y)
...
... def fd(x, y):
...     return grad_all(clip_test)(x, y)
...
... print("forward: ", f(Tensor(np.array([1.1]).astype(np.float32)),
...     Tensor(np.array([0.1]).astype(np.float32))))
... print("clip_gradient:", fd(Tensor(np.array([1.1]).astype(np.float32)),
...     Tensor(np.array([0.1]).astype(np.float32))))
>>> InsertGradientOfClipDemo()
forward: [0.11000001]
clip_gradient: (Tensor(shape=[1], dtype=Float32, value= [ 2.00000003e-01]),
    Tensor(shape=[1], dtype=Float32, value= [ 1.00000000e+00]))

```

20.10.7 mindspore.ops.Morph

class mindspore.ops.**Morph** (*fn, infer_shape, infer_dtype*)

Morph 算子用于对用户自定义函数 *fn* 进行封装，允许其被当做自定义算子使用。

Morph 算子的主要适用于静态图的分布式自动并行场景，通过在自定义函数 *fn* 中使用集合通信算子，实现自定义的并行计算逻辑，尤其适用于 *fn* 内存在动态 *Shape* 的场景。

Morph 算子作用于输入时，实际上是其内封装的自定义函数 *fn* 作用于输入。

Morph 算子与 *mindspore.ops.Custom()* 的主要区别在于，前者会在自动微分前被展开替换为用户自定义 *fn*，故无需实现反向函数。

说明：

- 本算子只支持图模式。
- *fn* 必须满足图模式语法约束。
- 用户无需实现自定义反向函数。
- 用户自定义函数不支持 *vararg*、*kwarg*、*kwonlyargs* 和自由变量。

参数：

- **fn** (Function) - MindSpore Function，用户自定义函数。
- **infer_shape** (Function) - Mindspore Function，用户自定义 *infer_shape* 函数。
- **infer_dtype** (Function) - Mindspore Function，用户自定义 *infer_dtype* 函数。

输入：

用户自定义 *fn* 的输入。

输出：

用户自定义 *fn* 的输出。

异常：

- **RuntimeError** - 如果算子在非图模式下被使用。

样例:

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import context, nn, ops, Tensor, Parameter
>>>
>>> np_weight0 = np.array([1.0, 2.0, 3.0])
>>> np_weight1 = np.array([4.0, 5.0, 6.0])
>>> np_input_x = np.array([7.0, 8.0, 9.0])
>>>
>>> def infer_dtype(args):
...     return args
>>>
>>> def infer_shape(args):
...     return args
>>>
>>> def mul_by(*args):
...     def inner(x):
...         return args[0] * x
...     return inner
>>>
>>> NUMBER_100 = 100
>>> class MorphNet(nn.Cell):
...     def __init__(self):
...         super(MorphNet, self).__init__()
...         self.weight0 = Parameter(Tensor(np_weight0, ms.float32), name="weight0")
...         self.weight1 = Parameter(Tensor(np_weight1, ms.float32), name="weight1")
...         self.mul_by_100 = ops.Morph(mul_by(NUMBER_100), infer_shape, infer_dtype)
...     def construct(self, x):
...         a = x * self.weight0
...         b = self.mul_by_100(a)
...         out = b * self.weight1
...         return out
>>>
>>> context.set_context(mode=context.GRAPH_MODE)
>>> input_x = Tensor(np_input_x, ms.float32)
>>> net = MorphNet()
>>> grad_op = ops.GradOperation(get_all=True, get_by_list=True)
>>> grad_net = grad_op(net, net.trainable_params())
>>> bwd_out = grad_net(input_x)
>>> x_grad = bwd_out[0][0].asnumpy()
>>> weight0_grad = bwd_out[1][0].asnumpy()
>>> weight1_grad = bwd_out[1][1].asnumpy()
>>> print("x_grad", x_grad)
>>> print("weight0_grad", weight0_grad)
>>> print("weight1_grad", weight1_grad)
x_grad [ 400. 1000. 1800.]
weight0_grad [2800. 4000. 5400.]
weight1_grad [ 700. 1600. 2700.]
```

20.10.8 mindspore.ops.Map

class mindspore.ops.Map (ops=None, reverse=False)

Map 将对输入序列应用设置的函数操作。

此操作将应用到输入序列的每个元素。

参数：

- **ops** (Union[MultitypeFuncGraph, None], 可选) - ops 是要应用的操作。如果 ops 为 None，则操作应放在实例的第一个输入中。默认值：None。
- **reverse** (bool, 可选) - 在某些场景中，优化器需要反转以提高并行性能，一般用户可忽略。reverse 表示是否为反向应用操作的标志。仅支持图模式。默认值：False。

输入：

- **args** (Tuple[sequence]) - 如果 ops 不是 None，则所有输入的序列和序列的每一行都应该是相同长度。例如，如果 args 的长度为 2，那么每个序列 (args[0][i], args[1][i]) 长度的 i 将作为操作的输入。如果 ops 为 None，则第一个输入是操作，另一个输入是待操作的序列。

输出：

序列，进行 ops 函数操作后的输出序列。例如 ops(args[0][i], args[1][i])。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import dtype as mstype
>>> from mindspore import Tensor, ops
>>> from mindspore.ops import MultitypeFuncGraph, Map
>>> tensor_list = (Tensor(1, mstype.float32), Tensor(2, mstype.float32), Tensor(3, mstype.
↳ float32))
>>> # square all the tensor in the list
>>>
>>> square = MultitypeFuncGraph('square')
>>> @square.register("Tensor")
... def square_tensor(x):
...     return ops.square(x)
>>>
>>> common_map = Map()
>>> output = common_map(square, tensor_list)
>>> print(output)
(Tensor(shape=[], dtype=Float32, value= 1), Tensor(shape=[], dtype=Float32, value= 4),
Tensor(shape=[], dtype=Float32, value= 9))
>>> square_map = Map(square, False)
>>> output = square_map(tensor_list)
>>> print(output)
(Tensor(shape=[], dtype=Float32, value= 1), Tensor(shape=[], dtype=Float32, value= 4),
Tensor(shape=[], dtype=Float32, value= 9))
```

20.10.9 mindspore.ops.MultitypeFuncGraph

class mindspore.ops.MultitypeFuncGraph (name, read_value=False)

MultitypeFuncGraph 是一个用于生成重载函数的类，使用不同类型作为输入。使用 *name* 初始化一个 MultitypeFuncGraph 对象，然后用带有输入类型的 *register* 注册器装饰注册类型。这样该函数可以使用不同的类型作为输入调用，一般与 *HyperMap*、*Map* 结合使用。

参数：

- **name** (str) - 操作名。
- **read_value** (bool, 可选) - 如果注册函数不需要对输入的值进行更改，即所有输入均为按值传递，则将 *read_value* 设置为 True。默认值：False。

异常：

- **ValueError** - 找不到给定参数类型所匹配的函数。

支持平台：

Ascend GPU CPU

样例：

```
>>> # `add` is a metagraph object which will add two objects according to
>>> # input type using ".register" decorator.
>>> from mindspore import Tensor
>>> from mindspore import dtype as mstype
>>> from mindspore import ops
>>>
>>> tensor_add = ops.Add()
>>> add = ops.MultitypeFuncGraph('add')
>>> @add.register("Number", "Number")
... def add_scala(x, y):
...     return x + y
>>> @add.register("Tensor", "Tensor")
... def add_tensor(x, y):
...     return tensor_add(x, y)
>>> output = add(1, 2)
>>> print(output)
3
>>> output = add(Tensor([0.1, 0.6, 1.2], dtype=mstype.float32), Tensor([0.1, 0.6, 1.2],
↪dtype=mstype.float32))
>>> print(output)
[0.2 1.2 2.4]
```

register (*type_names)

根据给出的字符串内容注册不同输入类型的函数。

参数：

- **type_names** (Union[str, *mindspore.dtype*]) - 输入类型的名称或者一个类型列表。

返回：

装饰器，一个根据 *type_names* 指定输入类型的注册函数的装饰器。

20.10.10 mindspore.ops.Partial

class mindspore.ops.Partial

生成偏函数的实例。通过给一般函数的部分参数提供初始值来衍生出有特定功能的新函数。

输入：

- **args** (Union[FunctionType, Tensor]) - 需传入的函数及其对应的参数。

输出：

FunctionType，偏函数及其对应的参数。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor
>>> from mindspore import ops
>>> def show_input(x, y, z):
...     return x, y, z
>>> partial = ops.Partial()
>>> partial_show_input = partial(show_input, Tensor(1))
>>> output1 = partial_show_input(Tensor(2), Tensor(3))
>>> print(output1)
(Tensor(shape=[], dtype=Int64, value= 1), Tensor(shape=[], dtype=Int64, value= 2),
↳Tensor(shape=[], dtype=Int64, value= 3))
>>> output2 = partial_show_input(Tensor(3), Tensor(4))
>>> print(output2)
(Tensor(shape=[], dtype=Int64, value= 1), Tensor(shape=[], dtype=Int64, value= 3),
↳Tensor(shape=[], dtype=Int64, value= 4))
```

20.10.11 mindspore.ops.Scan

class mindspore.ops.Scan

将一个函数循环作用于一个数组，且对当前元素的处理依赖上一个元素的执行结果。Scan 算子的执行逻辑可以近似表示为如下代码：

```
def Scan(loop_func, init, xs, length=None):
    if xs is None:
        xs = [None] * length
    carry = init
    ys = []
    for x in xs:
        carry, y = loop_func(carry, x)
        ys.append(y)
    return carry, ys
```

当前 Scan 算子存在以下语法限制：

- 暂不支持 *loop_func* 为副作用函数，如：对 Parameter、全局变量的修改等操作。
- 暂不支持 *loop_func* 的返回值的第一个元素与初始值 *init* 的类型或形状不同。

警告：这是一个实验性 API，后续可能修改或删除。

输入：

- **loop_func** (Function) - 循环体函数。
- **init** (Union[Tensor, number, str, bool, list, tuple, dict]) - 循环的初始值。
- **xs** (Union[tuple, list, None]) - 用于执行循环扫描的数组。
- **length** (Union[int, None], 可选) - 数组 xs 的长度，默认值：None。
- **unroll** (bool, 可选) - 是否在编译阶段进行循环展开，默认值：True。

输出：

Tuple(Union[Tensor, number, str, bool, list, tuple, dict], list)，由两个元素组成的 tuple，第一个元素为循环的最终结果，和 *init* 参数保持一样的类型和形状；第二个元素是一个列表，包含每次循环的执行结果。

异常：

- **TypeError** - *loop_func* 不是一个函数。
- **TypeError** - *xs* 不是一个 tuple、一个 list 或者 None。
- **TypeError** - *length* 不是一个整数或者 None。
- **TypeError** - *unroll* 不是一个布尔值。
- **ValueError** - *loop_func* 不能接受 *init* 以及 *xs* 的元素作为参数。
- **ValueError** - *loop_func* 的返回值不是一个包含两个元素的 tuple，或者 tuple 的第一个元素与 *init* 的类型或形状不同。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import ops
>>> def cumsum(res, el):
...     res = res + el
...     return res, res
...
>>> a = [1, 2, 3, 4]
>>> result_init = 0
>>> scan_op = ops.Scan()
>>> result = scan_op(cumsum, result_init, a)
>>> print(result == (10, [1, 3, 6, 10]))
True
```

20.10.12 mindspore.ops.WhileLoop

class mindspore.ops.WhileLoop

在编译阶段不进行循环展开的循环算子。WhileLoop 算子的执行逻辑可以近似表示为如下代码:

```
def WhileLoop(cond_func, loop_func, init_val):
    while cond_func(init_val):
        init_val = loop_func(init_val)
    return init_val
```

当前 WhileLoop 算子存在以下语法限制:

- 暂不支持 *loop_func* 为副作用函数, 如: 对 Parameter、全局变量的修改等操作。
- 暂不支持 *loop_func* 的返回值与初始值 *init_val* 的类型或形状不同。

警告: 这是一个实验性 API, 后续可能修改或删除。

输入:

- **cond_func** (Function) - 循环的条件函数。
- **loop_func** (Function) - 循环体函数, 接受一个参数, 并且返回值与输入参数的类型相同。
- **init_val** (Union[Tensor, number, str, bool, list, tuple, dict]) - 循环的初始值。

输出:

Union[Tensor, number, str, bool, list, tuple, dict], 循环的最终结果, 和 *init_val* 的类型和形状相同。

异常:

- **TypeError** - *cond_func* 不是一个函数。
- **TypeError** - *loop_func* 不是一个函数。
- **ValueError** - *loop_func* 不能接受 *init_val* 作为参数或者返回值和 *init_val* 的类型或形状不同。

支持平台:

Ascend GPU CPU

样例:

```
>>> from mindspore import ops
>>> def loop_while_fun(init_val):
...     val = init_val
...     val = val + 1
...     return val
...
>>> init_state = 10
>>> while_loop = ops.WhileLoop()
>>> result = while_loop(lambda x : x < 100, loop_while_fun, init_state)
>>> print(result)
100
```

20.11 算子信息注册

<code>mindspore.ops.AiCPURegOp</code>	AiCPU 算子信息注册类。
<code>mindspore.ops.CustomRegOp</code>	用于为 <code>mindspore.ops.Custom</code> 的 <code>func</code> 参数生成算子注册信息的类。
<code>mindspore.ops.DataType</code>	Ascend 算子的 <code>dtype</code> 和 <code>format</code> 的多种组合。
<code>mindspore.ops.TBERegOp</code>	注册 TBE 算子信息的类。
<code>mindspore.ops.get_vm_impl_fn</code>	通过 <code>Primitive</code> 对象或 <code>Primitive</code> 名称，获取虚拟实现函数。

20.11.1 mindspore.ops.AiCPURegOp

class mindspore.ops.AiCPURegOp(*op_name*)
AiCPU 算子信息注册类。

参数：

- **op_name** (str) - 算子名称。

样例：

```
>>> from mindspore.ops import AiCPURegOp, DataType
>>> stack_op_info = AiCPURegOp("Stack") \
...     .fusion_type("OPAQUE") \
...     .attr("axis", "int") \
...     .input(0, "x", "dynamic") \
...     .output(0, "y", "required") \
...     .dtype_format(DataType.I8_Default, DataType.I8_Default) \
...     .dtype_format(DataType.I16_Default, DataType.I16_Default) \
...     .dtype_format(DataType.I32_Default, DataType.I32_Default) \
...     .dtype_format(DataType.I64_Default, DataType.I64_Default) \
...     .dtype_format(DataType.U8_Default, DataType.U8_Default) \
...     .dtype_format(DataType.U16_Default, DataType.U16_Default) \
...     .dtype_format(DataType.U32_Default, DataType.U32_Default) \
...     .dtype_format(DataType.U64_Default, DataType.U64_Default) \
...     .dtype_format(DataType.F16_Default, DataType.F16_Default) \
...     .dtype_format(DataType.F32_Default, DataType.F32_Default) \
...     .dtype_format(DataType.F64_Default, DataType.F64_Default) \
...     .dtype_format(DataType.BOOL_Default, DataType.BOOL_Default) \
...     .get_op_info()
>>>
```

20.11.2 mindspore.ops.CustomRegOp

class mindspore.ops.CustomRegOp(*op_name='Custom'*)
用于为`mindspore.ops.Custom`的 `func` 参数生成算子注册信息的类。注册信息主要指定了 `func` 的输入和输出 Tensor 所支持的数据类型和数据格式、属性以及 `target` 信息。

参数：

- **op_name** (str, 可选) - 表示 kernel 名称，算子会把这个记录在算子节点的 `reg_op_name` 属性中。此外，算子会自动生成唯一名称作为识别注册信息的标识。默认值："Custom"。

样例：

```
>>> from mindspore.ops import CustomRegOp, DataType
>>> custom_op_ascend_info = CustomRegOp() \
...     .input(0, "x", "dynamic") \
...     .output(0, "y") \
...     .dtype_format(DataType.F16_Default, DataType.F16_Default) \
...     .dtype_format(DataType.F32_Default, DataType.F32_Default) \
...     .target("Ascend") \
...     .get_op_info()
```

attr (*name=None, param_type=None, value_type=None, default_value=None, **kwargs*)

指定 `mindspore.ops.Custom` 的 `func` 参数的属性信息。每次调用该函数都会产生一个属性信息，也就是说，如果 `func` 有两个属性，那么这个函数应该被连续调用两次。属性信息将生成为一个字典：{"name": *name*, "param_type": *param_type*, "value_type": *value_type*, "default_value": *default_value*}。

参数：

- **name** (str) - 表示属性的名称。如果该值为 `None`，键"index" 将不会出现在属性信息字典中。默认值：`None`。
- **param_type** (str) - 表示属性的参数类型，可以是 ["required", "optional"] 之一。如果该值为 `None`，键"param_type" 将不会出现在属性信息字典中。默认值：`None`。
 - "required": 表示必须为此属性提供值。可通过两种方式实现：通过在注册信息中设置默认值；在调用自定义算子时提供输入值。
 - "optional": 表示不强制为此属性提供值。
- **value_type** (str) - 表示属性的值的类型，可以是 ["int", "str", "bool", "float", "listInt", "listStr", "listBool", "listFloat"] 之一。如果该值为 `None`，键"value_type" 将不会出现在属性信息字典中。默认值：`None`。
 - "int": Python int 类型的字符串表示。
 - "str": Python str 类型的字符串表示。
 - "bool": Python bool 类型的字符串表示。
 - "float": Python float 类型的字符串表示。
 - "listInt": Python list of int 类型的字符串表示。
 - "listStr": Python list of str 类型的字符串表示。
 - "listBool": Python list of bool 类型的字符串表示。
 - "listFloat": Python list of float 类型的字符串表示。
- **default_value** (str) - 表示属性的默认值。`default_value` 和 `value_type` 配合使用。如果属性实际的默认值为 1.0，那么 `value_type` 是"float"，`default_value` 是"1.0"。如果属性实际的默认值是 [1, 2, 3]，那么 `value_type` 是"listInt"，`default_value` 是"1,2,3"，其中数值通过',' 分割。如果该值为 `None`，键"default_value" 将不会出现在属性信息字典中。目前用于"akg"、"aicpu" 和"tbe" 类型的自定义算子。默认值：`None`。
- ****kwargs** (dict) - 表示属性的其他信息，用于扩展。

异常：

- **TypeError** - *name* 既不是 str 也不是 `None`。
- **TypeError** - *param_type* 既不是 str 也不是 `None`。
- **TypeError** - *value_type* 既不是 str 也不是 `None`。
- **TypeError** - *default_value* 既不是 str 也不是 `None`。

dtype_format (*args)

指定`mindspore.ops.Custom`的 `func` 参数的每个输入 Tensor 和输出 Tensor 所支持的数据类型和数据格式。如上述样例，该函数应在 `input` 和 `output` 函数之后被调用。

参数：

- ***args** (tuple) - 表示（数据类型、格式）组合的列表，`args` 的长度应该等于输入 Tensor 和输出 Tensor 数目的总和。`args` 中的每一项也是一个 tuple，`tuple[0]` 和 `tuple[1]` 都是 str 类型，分别指定了一个 Tensor 的数据类型和数据格式。`mindspore.ops.DataType` 提供了很多预定义的（数据类型、格式）组合，例如 `DataType.F16_Default` 表示数据类型是 float16，数据格式是默认格式。

异常：

- **ValueError** - `args` 的长度不等于输入 Tensor 和输出 Tensor 数目的总和。

get_op_info()

将生成的注册信息以字典类型返回。如上述样例，`CustomRegOp` 实例最后调用该函数。

input (index=None, name=None, param_type='required', **kwargs)

指定`mindspore.ops.Custom`的 `func` 参数的输入 Tensor 信息。每次调用该函数都会产生一个输入 Tensor 信息，也就是说，如果 `func` 有两个输入 Tensor，那么该函数应该被连续调用两次。输入 Tensor 信息将生成一个字典：`{"index": index, "name": name, "param_type": param_type}`。

参数：

- **index** (int) - 表示输入的索引，从 0 开始。0 表示第一个输入 Tensor，1 表示第二个输入 Tensor，以此类推。如果该值为 None，键“index”将不会出现在输入 Tensor 信息字典中。默认值：None。
- **name** (str) - 表示第 `index` 个输入的名称。如果该值为 None，键“name”将不会出现在输入 Tensor 信息字典中。默认值：None。
- **param_type** (str) - 表示第 `index` 个输入的参数类型，可以是 `["required", "dynamic", "optional"]` 之一。如果该值为 None，键“param_type”将不会出现在输入 Tensor 信息字典中。默认值：“required”。
 - “required”：表示第 `index` 个输入存在并且只能是单个 Tensor。
 - “dynamic”：表示第 `index` 个输入存在且 Tensor 个数可能为多个，比如 AddN 算子的输入属于这种情况。
 - “optional”：表示第 `index` 个输入存在且为单个 Tensor，或者也可能不存在。
- ****kwargs** (dict) - 表示输入的其他信息，用于扩展。

异常：

- **TypeError** - `index` 既不是 int 也不是 None。
- **TypeError** - `name` 既不是 str 也不是 None。
- **TypeError** - `param_type` 既不是 str 也不是 None。

output (index=None, name=None, param_type='required', **kwargs)

指定`mindspore.ops.Custom`的 `func` 参数的输出 Tensor 信息。每次调用该函数都会产生一个输出 Tensor 信息，也就是说，如果 `func` 有两个输出 Tensor，那么该函数应该被连续调用两次。输出 Tensor 信息将生成一个字典：`{"index": index, "name": name, "param_type": param_type}`。

参数：

- **index** (int) - 表示输出的索引，从 0 开始。0 表示第一个输出 Tensor，1 表示第二个输出 Tensor，以此类推。如果该值为 None，键“index”将不会出现在输出 Tensor 信息字典中。默认值：None。
- **name** (str) - 表示第 `index` 个输出的名称。如果该值为 None，键“name”将不会出现在输出 Tensor 信息字典中。默认值：None。

- **param_type** (str) - 表示第 *index* 个输出的参数类型, 可以是 ["required", "dynamic", "optional"] 之一。如果该值为 None, 键“param_type”将不会出现在输出 Tensor 信息字典中。默认值: "required"。
- "required": 表示第 *index* 个输出存在并且只能是单个 Tensor。
- "dynamic": 表示第 *index* 个输出存在且 Tensor 个数可能为多个。
- "optional": 表示第 *index* 个输出存在且为单个 Tensor, 或者也可能不存在。
- ****kwargs** (dict) - 表示输出的其他信息, 用于扩展。

异常:

- **TypeError** - *index* 既不是 int 也不是 None。
- **TypeError** - *name* 既不是 str 也不是 None。
- **TypeError** - *param_type* 既不是 str 也不是 None。

target (*target=None*)

指定当前注册信息所对应的 target。

参数:

- **target** (str) - 表示当前注册信息所对应的 target, 可以是 ["Ascend", "GPU", "CPU"] 之一。对于同一个 *mindspore.ops.Custom* 的 *func* 参数, 其在不同的 target 上可能支持不同的数据类型和数据格式, 使用此参数指定注册信息用于哪个 target。如果该值为 None, 它将在 *mindspore.ops.Custom* 内部被自动推断。默认值: None。

异常:

- **TypeError** - *target* 既不是 str 也不是 None。

20.11.3 mindspore.ops.DataType

class mindspore.ops.DataType

Ascend 算子的 dtype 和 format 的多种组合。

当前支持:

```
None_None = ("", "")
None_Default = ("", "DefaultFormat")
BOOL_None = ("bool", "")
BOOL_Default = ("bool", "DefaultFormat")
BOOL_5HD = ("bool", "NC1HWC0")
BOOL_FracZ = ("bool", "FRACTAL_Z")
BOOL_FracNZ = ("bool", "FRACTAL_NZ")
BOOL_C1HWNC0C0 = ("bool", "C1HWNC0C0")
BOOL_NCHW = ("bool", "NCHW")
BOOL_NHWC = ("bool", "NHWC")
BOOL_HWCN = ("bool", "HWCN")
BOOL_NDHWC = ("bool", "NDHWC")
BOOL_ChannelLast = ("bool", "ChannelLast")

I8_None = ("int8", "")
I8_Default = ("int8", "DefaultFormat")
I8_5HD = ("int8", "NC1HWC0")
I8_FracZ = ("int8", "FRACTAL_Z")
I8_FracNZ = ("int8", "FRACTAL_NZ")
I8_C1HWNC0C0 = ("int8", "C1HWNC0C0")
```

(下页继续)

(续上页)

```

I8_NCHW = ("int8", "NCHW")
I8_NHWC = ("int8", "NHWC")
I8_HWCN = ("int8", "HWCN")
I8_NDHWC = ("int8", "NDHWC")
I8_ChannelLast = ("int8", "ChannelLast")

U8_None = ("uint8", "")
U8_Default = ("uint8", "DefaultFormat")
U8_5HD = ("uint8", "NC1HWC0")
U8_FracZ = ("uint8", "FRACTAL_Z")
U8_FracNZ = ("uint8", "FRACTAL_NZ")
U8_C1HWNC0C0 = ("uint8", "C1HWNC0C0")
U8_NCHW = ("uint8", "NCHW")
U8_NHWC = ("uint8", "NHWC")
U8_HWCN = ("uint8", "HWCN")
U8_NDHWC = ("uint8", "NDHWC")
U8_ChannelLast = ("uint8", "ChannelLast")

I16_None = ("int16", "")
I16_Default = ("int16", "DefaultFormat")
I16_5HD = ("int16", "NC1HWC0")
I16_FracZ = ("int16", "FRACTAL_Z")
I16_FracNZ = ("int16", "FRACTAL_NZ")
I16_C1HWNC0C0 = ("int16", "C1HWNC0C0")
I16_NCHW = ("int16", "NCHW")
I16_NHWC = ("int16", "NHWC")
I16_HWCN = ("int16", "HWCN")
I16_NDHWC = ("int16", "NDHWC")
I16_ChannelLast = ("int16", "ChannelLast")

U16_None = ("uint16", "")
U16_Default = ("uint16", "DefaultFormat")
U16_5HD = ("uint16", "NC1HWC0")
U16_FracZ = ("uint16", "FRACTAL_Z")
U16_FracNZ = ("uint16", "FRACTAL_NZ")
U16_C1HWNC0C0 = ("uint16", "C1HWNC0C0")
U16_NCHW = ("uint16", "NCHW")
U16_NHWC = ("uint16", "NHWC")
U16_HWCN = ("uint16", "HWCN")
U16_NDHWC = ("uint16", "NDHWC")
U16_ChannelLast = ("uint16", "ChannelLast")

I32_None = ("int32", "")
I32_Default = ("int32", "DefaultFormat")
I32_5HD = ("int32", "NC1HWC0")
I32_FracZ = ("int32", "FRACTAL_Z")
I32_FracNZ = ("int32", "FRACTAL_NZ")
I32_C1HWNC0C0 = ("int32", "C1HWNC0C0")
I32_NCHW = ("int32", "NCHW")
I32_NHWC = ("int32", "NHWC")
I32_HWCN = ("int32", "HWCN")
I32_NDHWC = ("int32", "NDHWC")
I32_ChannelLast = ("int32", "ChannelLast")

U32_None = ("uint32", "")
U32_Default = ("uint32", "DefaultFormat")

```

(下页继续)

(续上页)

```

U32_5HD = ("uint32", "NC1HWC0")
U32_FracZ = ("uint32", "FRACTAL_Z")
U32_FracNZ = ("uint32", "FRACTAL_NZ")
U32_C1HWNC0C0 = ("uint32", "C1HWNC0C0")
U32_NCHW = ("uint32", "NCHW")
U32_NHWC = ("uint32", "NHWC")
U32_HWCN = ("uint32", "HWCN")
U32_NDHWC = ("uint32", "NDHWC")
U32_ChannelLast = ("uint32", "ChannelLast")

I64_None = ("int64", "")
I64_Default = ("int64", "DefaultFormat")
I64_5HD = ("int64", "NC1HWC0")
I64_FracZ = ("int64", "FRACTAL_Z")
I64_FracNZ = ("int64", "FRACTAL_NZ")
I64_C1HWNC0C0 = ("int64", "C1HWNC0C0")
I64_NCHW = ("int64", "NCHW")
I64_NHWC = ("int64", "NHWC")
I64_HWCN = ("int64", "HWCN")
I64_NDHWC = ("int64", "NDHWC")
I64_ChannelLast = ("int64", "ChannelLast")

U64_None = ("uint64", "")
U64_Default = ("uint64", "DefaultFormat")
U64_5HD = ("uint64", "NC1HWC0")
U64_FracZ = ("uint64", "FRACTAL_Z")
U64_FracNZ = ("uint64", "FRACTAL_NZ")
U64_C1HWNC0C0 = ("uint64", "C1HWNC0C0")
U64_NCHW = ("uint64", "NCHW")
U64_NHWC = ("uint64", "NHWC")
U64_HWCN = ("uint64", "HWCN")
U64_NDHWC = ("uint64", "NDHWC")
U64_ChannelLast = ("uint64", "ChannelLast")

F16_None = ("float16", "")
F16_Default = ("float16", "DefaultFormat")
F16_5HD = ("float16", "NC1HWC0")
F16_FracZ = ("float16", "FRACTAL_Z")
F16_FracNZ = ("float16", "FRACTAL_NZ")
F16_C1HWNC0C0 = ("float16", "C1HWNC0C0")
F16_NCHW = ("float16", "NCHW")
F16_NHWC = ("float16", "NHWC")
F16_HWCN = ("float16", "HWCN")
F16_NDHWC = ("float16", "NDHWC")
F16_NCDHW = ("float16", "NCDHW")
F16_DHWCN = ("float16", "DHWCN")
F16_NDC1HWC0 = ("float16", "NDC1HWC0")
F16_FRACTAL_Z_3D = ("float16", "FRACTAL_Z_3D")
F16_FracZNLSTM = ("float16", "FRACTAL_ZN_LSTM")
F16_FracZNRNN = ("float16", "FRACTAL_ZN_RNN")
F16_ND_RNNBIAS = ("float16", "ND_RNN_BIAS")
F16_ChannelLast = ("float16", "ChannelLast")

F32_None = ("float32", "")
F32_Default = ("float32", "DefaultFormat")
F32_5HD = ("float32", "NC1HWC0")

```

(下页继续)

(续上页)

```

F32_FracZ = ("float32", "FRACTAL_Z")
F32_FracNZ = ("float32", "FRACTAL_NZ")
F32_C1HWNC0 = ("float32", "C1HWNC0")
F32_NCHW = ("float32", "NCHW")
F32_NHWC = ("float32", "NHWC")
F32_HWCN = ("float32", "HWCN")
F32_NDHWC = ("float32", "NDHWC")
F32_NCDHW = ("float32", "NCDHW")
F32_DHWCN = ("float32", "DHWCN")
F32_NDC1HWC0 = ("float32", "NDC1HWC0")
F32_FRACTAL_Z_3D = ("float32", "FRACTAL_Z_3D")
F32_FracZNLSTM = ("float32", "FRACTAL_ZN_LSTM")
F32_FracZNRNN = ("float32", "FRACTAL_ZN_RNN")
F32_ND_RNNBIAS = ("float32", "ND_RNN_BIAS")
F32_ChannelLast = ("float32", "ChannelLast")

F64_None = ("float64", "")
F64_Default = ("float64", "DefaultFormat")
F64_5HD = ("float64", "NC1HWC0")
F64_FracZ = ("float64", "FRACTAL_Z")
F64_FracNZ = ("float64", "FRACTAL_NZ")
F64_C1HWNC0 = ("float64", "C1HWNC0")
F64_NCHW = ("float64", "NCHW")
F64_NHWC = ("float64", "NHWC")
F64_HWCN = ("float64", "HWCN")
F64_NDHWC = ("float64", "NDHWC")
F64_ChannelLast = ("float64", "ChannelLast")

C64_Default = ("complex64", "DefaultFormat")
C128_Default = ("complex128", "DefaultFormat")

```

20.11.4 mindspore.ops.TBERegOp

class mindspore.ops.TBERegOp(*op_name*)

注册 TBE 算子信息的类。TBE (Tensor Boost Engine) 是 Ascend 算子开发工具, 在 TVM 框架基础上扩展, 进行自定义算子开发。

参数:

- **op_name** (str) - 表示算子名称。

样例:

```

>>> from mindspore.ops import TBERegOp, DataType
>>> op_name_op_info = TBERegOp("OpName") \
...     .fusion_type("ELEMWISE") \
...     .async_flag(False) \
...     .binfile_name("op_name.so") \
...     .compute_cost(10) \
...     .kernel_name("op_name") \
...     .partial_flag(True) \
...     .op_pattern("formatAgnostic") \
...     .need_check_supported(True) \
...     .dynamic_shape(True) \
...     .dynamic_rank_support(True) \

```

(下页继续)

```

...     .dynamic_compile_static(True) \
...     .attr("format", "required", "str", "all") \
...     .input(0, "x1", None, "required", None) \
...     .input(0, "x2", None, "required", None) \
...     .input(1, "axis", None, "required", None) \
...     .output(0, "y", True, "required", "all") \
...     .real_input_index([1, 0]) \
...     .input_to_attr_index([2]) \
...     .unknown_shape_formats(["ND", "ND", "ND", "ND"]) \
...     .reshape_type("NC") \
...     .is_dynamic_format(True) \
...     .dtype_format(DataType.F16_None, DataType.F16_None, DataType.F16_None, DataType.F16_
↪None) \
...     .dtype_format(DataType.F32_None, DataType.F32_None, DataType.F32_None, DataType.F32_
↪None) \
...     .dtype_format(DataType.I32_None, DataType.I32_None, DataType.I32_None, DataType.I32_
↪None) \
...     .get_op_info()
>>>

```

async_flag (*async_flag=False*)

定义算子的计算效率，标识是否支持异步计算。

参数：

- **async_flag** (bool, 可选) - 该算子是否支持异步计算的标识。默认值：False。

attr (*name=None, param_type=None, value_type=None, value=None, default_value=None, **kwargs*)

注册 TBE 算子的属性信息。

参数：

- **name** (str, 可选) - 表示算子属性的名称。默认值：None。
- **param_type** (str, 可选) - 表示算子属性的参数类型。默认值：None。
- **value_type** (str, 可选) - 表示算子属性的类型。默认值：None。
- **value** (str, 可选) - 表示算子属性的值。默认值：None。
- **default_value** (str, 可选) - 表示算子属性的默认值。默认值：None。
- **kwargs** (dict) - 表示算子属性的其他信息。

binfile_name (*binfile_name*)

设置算子底层的二进制文件名，此操作可选。

参数：

- **binfile_name** (str) - 表示算子底层的二进制文件名。

compute_cost (*compute_cost=10*)

定义算子的计算效率，即 tiling 模块中成本模型的值。

参数：

- **compute_cost** (int, 可选) - 表示计算成本的值。默认值：10。

dynamic_compile_static (*dynamic_compile_static=False*)

定义算子是否支持动静合一。

参数：

- **dynamic_compile_static** (bool, 可选) - 该算子是否支持动静合一的标识。默认值: False。

dynamic_rank_support (*dynamic_rank_support*)

定义算子是否支持 DynamicRank (动态维度)。

参数:

- **dynamic_rank_support** (bool, 可选) - 该算子是否支持 DynamicRank (动态维度) 的标识。True: 表示算子支持 DynamicRank。False: 表示算子不支持 DynamicRank。默认值: False。

dynamic_shape (*dynamic_shape=False*)

表示算子是否支持动态 shape。

参数:

- **dynamic_shape** (bool, 可选) - 该算子是否支持动态 shape 的标识。默认值: False。

input (*index=None, name=None, need_compile=None, param_type=None, shape=None, value_depend=None, **kwargs*)

注册 TBE 算子的输入信息。

参数:

- **index** (int, 可选) - 表示输入的顺序。默认值: None。
- **name** (str, 可选) - 表示当前输入的名称。默认值: None。
- **need_compile** (bool, 可选) - 表示输入是否需要编译。默认值: None。
- **param_type** (str, 可选) - 表示输入的类型。默认值: None。
- **shape** (str, 可选) - 表示输入的 shape。默认值: None。
- **value_depend** (str, 可选) - 表示输入是否为常量值。默认值: None。
- **kwargs** (dict) - 表示输入的其他信息。

input_to_attr_index (*input_to_attr_index*)

需要转换为属性的输入的索引。

参数:

- **input_to_attr_index** (list) - 索引。默认值: ()。

is_dynamic_format (*is_dynamic_format=False*)

表示算子是否需要 op_select_format 函数来动态选择合适的数据格式和数据类型。

参数:

- **is_dynamic_format** (bool, 可选) - 该算子是否需要 op_select_format 函数来动态选择合适的数据格式和数据类型的标识。默认值: False。

kernel_name (*kernel_name*)

表示算子名称。

参数:

- **kernel_name** (str) - 表示算子名称。

need_check_supported (*need_check_supported=False*)

表示算子是否需要检查支持。

参数:

- **need_check_supported** (bool, 可选) - 该算子否需要检查支持的标识。默认值: False。

op_pattern (*pattern=None*)

表示算子支持的行为类型。

参数:

- **pattern** (str, 可选) - 算子支持的行为类型, 如"broadcast"、"reduce"等。默认值: None。

output (*index=None, name=None, need_compile=None, param_type=None, shape=None, **kwargs*)

注册 TBE 算子的输出信息。

参数:

- **index** (int, 可选) - 表示输出的顺序。默认值: None。
- **name** (str, 可选) - 表示输出的名称。默认值: None。
- **need_compile** (bool, 可选) - 表示输出是否需要编译。默认值: None。
- **param_type** (str, 可选) - 表示输出的类型。默认值: None。
- **shape** (str, 可选) - 表示输出的 shape。默认值: None。
- **kwargs** (dict) - 表示输出的其他信息。

partial_flag (*partial_flag=True*)

定义算子的计算效率, 标识是否支持部分计算。

参数:

- **partial_flag** (bool, 可选) - 该算子是否支持部分计算的标识。默认值: True。

real_input_index (*real_input_index*)

算子前端输入到后端 TBE 算子输入的映射。

参数:

- **real_input_index** (list) - 真实输入的索引。默认值: ()。

reshape_type (*reshape_type*)

指定算子的补维方式。

参数:

- **reshape_type** (str) - 指定算子补维方式的值。例如: 输入的 shape 为 (2, 3), 指定 reshape_type="CH", 则补维之后的 shape 为 (1, 2, 3, 1), 即保留 CH 轴, NW 轴补 1。

unknown_shape_formats (*unknown_shape_formats*)

动态 Shape 场景下算子, 输入/输出 Tensor 的数据排布。

参数:

- **unknown_shape_formats** (list) - 表示动态 Shape 场景下, 算子输入/输出 Tensor 的数据排布。

20.11.5 mindspore.ops.get_vm_impl_fn

mindspore.ops.get_vm_impl_fn (*prim*)

通过 Primitive 对象或 Primitive 名称, 获取虚拟实现函数。

参数:

- **prim** (Union[Primitive, str]) - 算子注册的 Primitive 对象或名称。

说明: 该机制目前适用于调试。

返回:

函数，虚拟实现函数。

样例：

```
>>> from mindspore.ops import vm_impl_registry, get_vm_impl_fn
...
>>> @vm_impl_registry.register("Type")
... def vm_impl_dtype(self):
...     def vm_impl(x):
...         return type(x)
...     return vm_impl
...
>>> fn = get_vm_impl_fn("Type")
>>> out = fn(1.0)
>>> print(out)
<class 'float'>
```

20.12 自定义算子

接口名	概述	支持平台
<code>mindspore.ops.Custom</code>	<i>Custom</i> 算子是 MindSpore 自定义算子的统一接口。	Ascend GPU CPU
<code>mindspore.ops.CustomOpBuilder</code>	<i>CustomOpBuilder</i> 用于初始化和配置 MindSpore 的自定义算子。	Ascend CPU
<code>mindspore.ops.custom_info_register</code>	装饰器，用于将注册信息绑定到： <code>mindspore.ops.Custom</code> 的 <i>func</i> 参数。	Ascend GPU CPU
<code>mindspore.ops.kernel</code>	用于 MindSpore Hybrid DSL 函数书写的装饰器。	GPU CPU

20.12.1 mindspore.ops.Custom

class mindspore.ops.**Custom** (*func, bprop=None, out_dtype=None, func_type='hybrid', out_shape=None, reg_info=None*)

Custom 算子是 MindSpore 自定义算子的统一接口。用户可以利用该接口自行定义 MindSpore 内置算子库尚未包含的算子。根据输入函数的不同，用户可以创建多个自定义算子，并将其应用于神经网络中。关于自定义算子的详细说明和介绍，包括参数的正确书写，请参见 [自定义算子教程](#)。

警告：

- 这是一个实验性 API，后续可能修改或删除。

说明：不同自定义算子的函数类型（*func_type*）支持的平台类型不同。每种类型支持的平台如下：

- "aot": ["GPU", "CPU", "Ascend"].
- "pyfunc": ["CPU"].
- "julia": ["CPU"].

参数：

- **func** (Union[function, str]) - 自定义算子的函数表达。

- **function**: 如果 *func* 是函数类型, 那么 *func* 应该是一个 Python 函数, 它描述了用户定义的操作符的计算逻辑。
- **字符串**: 如果 *func* 是字符串类型, 那么 *str* 应该是包含函数名的文件路径。当 *func_type* 是 "aot" 或 "julia" 时, 可以使用这种方式。

1. 对于 "aot":

- a) GPU/CPU (仅 Linux) 平台

"aot" 意味着提前编译。在这种情况下, Custom 直接启动用户定义的 "xxx.so" 文件作为操作符。用户需要提前将手写的 "xxx.cu"/"xxx.cc" 文件编译成 "xxx.so", 并提供文件路径和函数名。

* "xxx.so" 文件生成:

- 1) GPU 平台: 给定用户定义的 "xxx.cu" 文件 (例如 "{path}/add.cu"), 使用 `nvcc` 命令进行编译 (例如 `nvcc --shared -Xcompiler -fPIC -o add.so add.cu`)。
- 2) CPU 平台: 给定用户定义的 "xxx.cc" 文件 (例如 "{path}/add.cc"), 使用 `g++/gcc` 命令进行编译 (例如 `g++ --shared -fPIC -o add.so add.cc`)。

* 定义 "xxx.cc"/"xxx.cu" 文件:

"aot" 是一个跨平台的标识符。"xxx.cc" 或 "xxx.cu" 中定义的函数具有相同的参数。通常, 该函数应该像这样:

```
int func(int nparam, void **params, int *ndims, int64_t **shapes, const char *
    **dtypes, void *stream, void *extra)
```

参数:

- *nparam(int)*: 输入和输出的总数; 假设操作符有 2 个输入和 3 个输出, 那么 *nparam*=5。
- *params(void **)*: 输入和输出指针的数组指针; 输入和输出的指针类型为 *void **; 假设操作符有 2 个输入和 3 个输出, 那么第一个输入的指针是 *params[0]*, 第二个输出的指针是 *params[3]*。
- *ndims(int *)*: 输入和输出维度数的数组指针; 假设 *params[i]* 是一个 1024x1024 的张量, *params[j]* 是一个 77x83x4 的张量, 那么 *ndims[i]*=2, *ndims[j]*=3。
- *shapes(int64_t **)*: 输入和输出形状 (*int64_t **) 的数组指针; 第 *i* 个输入的第 *j* 个维度的大小是 *shapes[i][j]* (其中 $0 \leq j < ndims[i]$); 假设 *params[i]* 是一个 2x3 的张量, *params[j]* 是一个 3x3x4 的张量, 那么 *shapes[i][0]*=2, *shapes[j][2]*=4。
- *dtypes(const char **)*: 输入和输出类型 (*const char **) 的数组指针; (例如: "float32"、"float16"、"float"、"float64"、"int"、"int8"、"int16"、"int32"、"int64"、"uint"、"uint8"、"uint16"、"uint32"、"uint64"、"bool")
- *stream(void *)*: 流指针, 仅在 CUDA 文件中使用。
- *extra(void *)*: 用于进一步扩展。

返回值 (int):

- 0: 如果这个 AOT 内核成功执行, MindSpore 将继续运行。
- 其他值: MindSpore 将引发异常并退出。

示例: 详见 `tests/st/ops/graph_kernel/custom/aot_test_files/` 中的详细信息。

* 在 Custom 中使用:

```
Custom(func="{dir_path}/{file_name}:{func_name}", ...)
```

例如: `Custom(func="/reorganize.so:CustomReorganize", out_shape=[1], out_dtype=mstype.float32, "aot")`

b) Ascend 平台

在 Ascend 平台使用 Custom 算子之前, 用户首先需要基于 Ascend C 开发自定义算子并编译。完整的开发和使用流程可参考教程 [AOT 类型自定义算子 \(Ascend 平台\)](#)。在入参 *func* 中传入算子的名字。根据 infer 函数的实现方式, 存在以下两种使用方式:

- * **Python infer**: 若算子的 infer 函数是 Python 实现, 即通过 *out_shape* 和 *out_dtype* 参数传入 infer shape 和 infer type 函数, 则指定 *func*="CustomName"。
- * **C++ infer**: 若算子的 infer 函数通过 C++ 实现, 则在 *func* 中传入 infer shape 或 infer type 实现文件的路径并用 : 隔开算子名字, 例如: *func*="add_custom_infer.cc:AddCustom"。

2. 对于"julia":

目前, "julia" 仅支持 CPU (仅限 Linux 平台)。对于 julia, 它使用 JIT 编译器 (即时编译器), 并且 julia 支持 C API 来调用 julia 代码。自定义功能可以直接将用户定义的"xxx.jl" 文件作为一个操作符来启动。用户需要编写一个包含模块和函数的"xxx.jl" 文件, 并提供该文件的路径以及模块名称和函数名称。

示例: 详情见 *tests/st/ops/graph_kernel/custom/julia_test_files/*

- * 在 Custom 中使用:

```
Custom(func="{dir_path}/{file_name}:{module_name}:{func_name}", ...)
```

例如: `Custom(func="./add.jl:Add:add", out_shape=[1], out_dtype=mstype.float32, "julia")`

- **out_shape** (Union[function, list, tuple]) - 自定义算子的输入的形狀或者输出形状的推导函数。默认值: None。
- **out_dtype** (Union[function, *mindspore.dtype*, tuple[*mindspore.dtype*]]) - 自定义算子的输入的数据类型或者输出数据类型的推导函数。默认值: None。
- **func_type** (str) - 自定义算子的函数类型, 必须是 ["aot", "pyfunc", "julia"] 中之一。
- **bprop** (function) - 自定义算子的反向函数。默认值: None。
- **reg_info** (Union[str, dict, list, tuple]) - 自定义算子的算子注册信息。默认值: None。

输入:

- **input** (Union(tuple, list)) - 输入要计算的 Tensor。

输出:

Tensor, 自定义算子的计算结果。

异常:

- **TypeError** - 如果输入 *func* 不合法, 或者 *func* 对应的注册信息类型不对。
- **ValueError** - *func_type* 的值不在列表内。
- **ValueError** - 算子注册信息不合法, 包括支持平台不匹配, 算子输入和属性与函数不匹配。

支持平台:

Ascend GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> from mindspore.ops import CustomRegOp, custom_info_register, DataType, kernel
>>> from mindspore import dtype as mstype
```

(下页继续)

(续上页)

```

>>> from mindspore.nn import Cell
>>> input_x = Tensor(np.ones([16, 16]).astype(np.float32))
>>> input_y = Tensor(np.ones([16, 16]).astype(np.float32))
>>>
>>> # Example, func_type = "pyfunc"
>>> def func_pyfunc(x1, x2):
...     return x1 + x2
>>>
>>> test_pyfunc = ops.Custom(func_pyfunc, lambda x, _: x, lambda x, _: x, "pyfunc")
>>> output = test_pyfunc(input_x, input_y)
>>> print(output.shape)
(16, 16)

```

20.12.2 mindspore.ops.CustomOpBuilder

class mindspore.ops.CustomOpBuilder (name, sources, backend=None, include_paths=None, cflags=None, ldflags=None, **kwargs)

CustomOpBuilder 用于初始化和配置 MindSpore 的自定义算子。用户可以通过该类定义和加载自定义算子模块，并将其应用到网络中。

一般情况下，用户仅需在类的构造函数中传入源文件和额外的编译选项，并调用 *load* 接口即可完成算子的编译和加载。但如果用户有特殊的定制需求，也可以通过继承该类并重写部分接口来实现。需要注意的是，重写接口后，构造函数中传入的部分参数可能会被忽略。

警告：这是一个实验性 API，后续可能修改或删除。

参数：

- **name** (str) - 自定义算子模块的唯一名称，用于标识算子。
- **sources** (Union[str, list[str]]) - 自定义算子的源文件，可以是单个文件路径或文件路径列表。
- **backend** (str, 可选) - 自定义算子的目标后端，例如“CPU”或“Ascend”。默认值：None。
- **include_paths** (list[str], 可选) - 编译过程中需要的额外包含路径。默认值：None。
- **cflags** (str, 可选) - 编译过程中使用的额外 C++ 编译选项。默认值：None。
- **ldflags** (str, 可选) - 链接过程中使用的额外链接选项。默认值：None。
- **kwargs** (dict, 可选) - 额外的关键字参数，用于扩展功能或自定义需求。

说明：

- 如果提供了 *backend* 参数，编译和链接步骤中会自动添加支持目标后端的默认编译和链接选项。默认选项可参考 *CustomOpBuilder* 代码中 *get_cflags* 和 *get_ldflags* 接口的实现。
- *sources* 参数必须指向有效的自定义算子源文件。

支持平台：

Ascend CPU

样例：


```
>>> from mindspore import ops
>>> builder = ops.CustomOpBuilder(
...     name="custom_op_cpu",
...     sources="custom_ops_impl/pybind_op_cpu.cpp",
...     backend="CPU"
... )
>>> my_ops = builder.load()
```

build()

编译自定义算子模块。

该方法会根据提供的源文件、包含路径、编译选项和链接选项，生成自定义算子的动态库文件。

返回：

str，编译生成的模块文件路径。

get_cflags()

获取编译自定义算子时的 C++ 编译选项。

返回：

list[str]，C++ 编译选项列表。

get_include_paths()

获取编译自定义算子时所需的头文件包含路径。

返回：

list[str]，包含路径的列表。

get_ldflags()

获取链接自定义算子时的链接选项。

返回：

list[str]，链接选项列表。

get_sources()

返回自定义算子的源文件路径。

返回：

Union[str, list[str]]，自定义算子的源文件路径，可能是字符串或字符串列表。

load()

编译并加载自定义算子模块。

返回：

Module，加载的自定义算子模块。

20.12.3 mindspore.ops.custom_info_register

`mindspore.ops.custom_info_register(*reg_info)`

装饰器，用于将注册信息绑定到：`mindspore.ops.Custom` 的 `func` 参数。

说明：`reg_info` 将添加到算子库'oplib'中。

参数：

- **reg_info** (tuple[str, dict]) - json 格式的算子注册信息。

返回：

function，返回算子信息注册的装饰器。

异常：

- **TypeError** - 如果 `reg_info` 不是 tuple。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore.ops import custom_info_register, CustomRegOp, DataType
>>> custom_func_ascend_info = CustomRegOp() \
...     .input(0, "x", "dynamic") \
...     .output(0, "y") \
...     .dtype_format(DataType.F16_Default, DataType.F16_Default) \
...     .dtype_format(DataType.F32_Default, DataType.F32_Default) \
...     .target("Ascend") \
...     .get_op_info()
>>>
>>> @custom_info_register(custom_func_ascend_info)
... def custom_func(x):
...     pass
```

20.12.4 mindspore.ops.kernel

`mindspore.ops.kernel(fn=None, reg_info=None, compile_attrs=None)`

用于 MindSpore Hybrid DSL 函数书写的装饰器。给用 MindSpore Hybrid DSL 书写的函数加上此装饰器后，它可以用作一个普通的 Python 函数。与此同时，它可以用于自定义算子 `mindspore.ops.Custom`（参数 `func_type` 可以设置为 "hybrid" 或者 "pyfunc"）的输入。使用 `hybrid` 类型的 `mindspore.ops.Custom` 自定义算子可以自动推导数据类型和形状。

参数：

- **fn** (Function) - 将被作为自定义算子运行的 Python 函数。默认 None。
- **reg_info** (tuple[str, dict]) - 包含算子注册信息的 dict 或 json 字符串。默认 None。
- **compile_attrs** (Dict) - 算子编译信息。默认 None。

返回：

Function，如果 `fn` 不是 None，那么返回一个用 Hybrid DSL 写的可执行函数；如果 `fn` 是 None，则返回一个装饰器，该装饰器只有 `fn` 一个参数。

支持平台:

GPU CPU

样例:

```
>>> import numpy as np
>>> from mindspore import ops, Tensor
>>> from mindspore.ops import kernel, DataType, CustomRegOp
...
>>> # Create a dict for the compile flags.
>>> attrs = {
...     "test1": True,
...     "test2": "good",
...     "test3": 12,
... }
>>> # Create the reg info json string.
>>> op_cpu_info = CustomRegOp() \
...     .input(0, "a") \
...     .input(0, "b") \
...     .output(0, "y") \
...     .dtype_format(DataType.F32_None, DataType.F32_None, DataType.F32_None) \
...     .target("CPU") \
...     .get_op_info()
>>>
>>> # Create inputs for the custom op.
>>> input_x = np.ones([4, 4]).astype(np.float32)
>>> input_y = np.ones([4, 4]).astype(np.float32)
...
>>> # Write a Hybrid DSL function through the decorator @kernel.
>>> # We can also pass the compile attrs and the reg info through the decorator.
>>> @kernel(reg_info=op_cpu_info, compile_attrs=attrs)
... def outer_product(a, b):
...     c = output_tensor(a.shape, a.dtype)
...
...     with block_realize(c):
...         for i0 in range(a.shape[0]):
...             for i1 in range(b.shape[1]):
...                 c[i0, i1] = 0.0
...                 for i2 in range(a.shape[1]):
...                     c[i0, i1] = c[i0, i1] + (a[i0, i2] * b[i2, i1])
...
...     return c
...
>>> # We can use the function directly as a python function.
>>> # In this case, the inputs should be numpy arrays.
>>> result = outer_product(input_x, input_y)
```

20.13 光谱算子

接口名	概述	支持平台
<code>mindspore.ops.BartlettWindow</code>	巴特利特窗口函数。	Ascend GPU CPU
<code>mindspore.ops.BlackmanWindow</code>	布莱克曼窗口函数。	Ascend GPU CPU

20.13.1 mindspore.ops.BartlettWindow

class mindspore.ops.BartlettWindow (*periodic=True, dtype=mstype.float32*)
巴特利特窗口函数。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.bartlett_window()`。

参数：

- **periodic** (bool, 可选) - 如果为 True，返回一个窗口作为周期函数使用。如果为 False，返回一个对称窗口。默认值：True。
- **dtype** (mindspore.dtype, 可选) - 输出数据类型，目前只支持 float16、float32 和 float64。默认值：mstype.float32。

输入：

- **window_length** (Tensor) - 返回窗口的大小，数据类型为 int32、int64。输入数据的值为在 [0, 1000000] 范围内的整数。

输出：

1D Tensor，大小为 `window_length`，数据类型由 `dtype` 指定。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import Tensor, ops
>>> from mindspore import dtype as mstype
>>> window_length = Tensor(5, mstype.int32)
>>> bartlett_window = ops.BartlettWindow(periodic=True, dtype=mstype.float32)
>>> output = bartlett_window(window_length)
>>> print(output)
[0.  0.4 0.8 0.8 0.4]
```

20.13.2 mindspore.ops.BlackmanWindow

class mindspore.ops.BlackmanWindow (*periodic=True, dtype=mstype.float32*)
布莱克曼窗口函数。

警告：这是一个实验性 API，后续可能修改或删除。

更多参考详见 `mindspore.ops.blackman_window()`。

参数：

- **periodic** (bool, 可选) - 如果为 True，返回一个窗口作为周期函数使用。如果为 False，返回一个对称窗口。默认值：True。
- **dtype** (mindspore.dtype, 可选) - 输出数据类型，目前只支持 float16、float32 和 float64。默认值：mstype.float32。

输入：

- **window_length** (Tensor) - 返回窗口的大小，数据类型为 int32，int64。输入数据的值为 [0, 1000000] 的整数。

输出：

1D Tensor，大小为 *window_length*，数据类型与 *dtype* 一致。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> from mindspore import Tensor, ops
>>> window_length = Tensor(10, mindspore.int32)
>>> blackman_window = ops.BlackmanWindow(periodic = True, dtype = mindspore.float32)
>>> output = blackman_window(window_length)
>>> print(output)
[-2.9802322e-08  4.0212840e-02  2.0077014e-01  5.0978714e-01
  8.4922993e-01  1.0000000e+00  8.4922981e-01  5.0978690e-01
  2.0077008e-01  4.0212870e-02]
```


Boost 能够自动优化网络性能，例如通过减少 BN、梯度冻结、累加梯度等方式实现网络加速。

说明：此特性为测试版本，我们仍在改进其功能。

class mindspore.boost.AutoBoost (level='00', boost_config_dict="")

MindSpore 自动优化算法库。

参数：

- **level** (str) - Boost 的配置级别，默认值："00"。
 - "00"：不变化。
 - "01"：启用 boost 模式，性能将提升约 20%，准确率保持不变。
 - "02"：启用 boost 模式，性能将提升约 30%，准确率下降小于 3%。
- **boost_config_dict** (dict) - 用户可配置的超参字典，建议的格式如下：

```
{
    "boost": {
        "mode": "auto",
        "less_bn": False,
        "grad_freeze": False,
        "adasum": False,
        "grad_accumulation": False,
        "dim_reduce": False},

    "common": {
        "gradient_split_groups": [50, 100],
        "device_number": 8},

    "less_bn": {
        "fn_flag": True,
        "gc_flag": True},
```

(下页继续)

```

"grad_freeze": {
    "param_groups": 10,
    "freeze_type": 1,
    "freeze_p": 0.7,
    "total_steps": 65536},

"dim_reduce": {
    "rho": 0.55,
    "gamma": 0.9,
    "alpha": 0.001,
    "sigma": 0.4,
    "n_components": 32,
    "pca_mat_path": None,
    "weight_load_dir": None,
    "timeout": 1800}

}

```

默认值: ""。

– boost:

- * mode (str): Boost 配置模式, 支持 ["auto", "manual", "enable_all", "disable_all"]。默认值: "auto"。
 - auto: 自动配置, 取决于 Model 类中的 *boost_level* 参数配置。
 - manual: 在 *boost_config_dict* 中人工配置。
 - enable_all: 开启所有 boost 算法。
 - disable_all: 关闭所有 boost 算法。
 - * less_bn (bool): 是否开启 LessBN 算法, 默认值: False。
 - * grad_freeze (bool): 是否开启梯度冻结算法, 默认值: False。
 - * adasum (bool): 是否开启自适应求和算法, 默认值: False。
 - * grad_accumulation (bool): 是否开启梯度累加算法, 默认值: False。
 - * dim_reduce (bool): 是否开启降维训练算法, 默认值: False。
- 如果开启 dim_reduce 算法, 其他算法会失效。如果开启 grad_freeze 算法, 同时关闭 dim_reduce, 其他算法会失效。

– common:

- * gradient_split_groups (list): 网络的梯度分割点, 默认值: [50, 100]。
- * device_number (int): 设备数, 默认值: 8。

– less_bn:

- * fn_flag (bool): 是否采用 fn 替换 fc, 默认值: True, 采用 fn 替换 fc。
- * gc_flag (bool): 是否启用 gc, 默认值: True, 启用 gc。

– grad_freeze:

- * param_groups (int): 参数分组数量, 默认值: 10。
- * freeze_type (int): 梯度冻结策略, 参数选择 [0, 1], 默认值: 1。
- * freeze_p (float): 梯度冻结概率, 默认值: 0.7。

* `total_steps` (int): 总训练步数, 默认值: 65536。

– `dim_reduce`:

`dim_reduce` 主要原理:

$$\begin{aligned} grad\_k &= pca\_mat \cdot grad \\ dk &= -bk \cdot grad\_k \\ sk &= rho^m \cdot dk \\ delta\_loss &= sigma \cdot grad\_k.T \cdot sk \end{aligned}$$

其中:

* `pca_mat` (array): 维度 $(k * n)$, k 是 `n_components` 的大小, n 是权重的大小。

* `bk` (array): 维度 $(k * k)$, `bk` 是拟牛顿法中的对称正定矩阵。

我们需要找到满足以下条件的 `m`:

$$new\_loss < old\_loss + delta\_loss$$

然后使用 `delta_grad` 去更新模型的权重:

$$\begin{aligned} grad\_k\_proj &= pca\_mat.T \cdot grad\_k \\ new\_grad\_momentum &= gamma \cdot old\_grad\_momentum + grad - grad\_k\_proj \\ delta\_grad &= alpha \cdot new\_grad\_momentum - pca\_mat.T \cdot sk \end{aligned}$$

* `rho` (float): 超参, 一般无需调整, 默认值: 0.55。

* `gamma` (float): 超参, 一般无需调整, 默认值: 0.9。

* `alpha` (float): 超参, 一般无需调整, 默认值: 0.001。

* `sigma` (float): 超参, 一般无需调整, 默认值: 0.4。

* `n_components` (int): PCA 后的维度, 默认值: 32。

* `pca_mat_path` (str): PCA 矩阵的加载路径, 使用绝对路径, 默认值: None。

* `weight_load_dir` (str): 以 `checkpoint` 形式保存的权重加载路径, 用于计算 PCA 矩阵, 默认值: None。

* `timeout` (int): 加载 PCA 矩阵的最长等待时间, 默认值: 1800 (s)。

用户可以通过加载 JSON 文件或者直接使用字典来配置 `boost_config_dict`。未配置的参数会使用默认值。

异常:

- **ValueError** - Boost 的模式不在 ["auto", "manual", "enable_all", "disable_all"] 这个列表中。

支持平台:

Ascend

样例:

```
>>> from mindspore.boost import AutoBoost
>>> #1) when configuring the dict directly:
>>> boost_config_dict = {"boost": {"mode": "auto"}}
>>> boost = AutoBoost("01", boost_config_dict)
>>>
>>> #2) when loading the dict from a json file:
>>> import json
>>> boost_json = "/path/boost_config.json"
```

(下页继续)

(续上页)

```
>>> with open(boost_json, 'r') as fp:
...     boost_config_dict = json.load(fp)
>>> boost = AutoBoost("01", boost_config_dict)
```

network_auto_process_eval (*network*)

使用 Boost 算法推理。

参数：

- **network** (Cell) - 推理网络。

network_auto_process_train (*network, optimizer*)

使用 Boost 算法训练。

参数：

- **network** (Cell) - 训练网络。
- **optimizer** (Cell) - 用于更新权重的优化器。

class mindspore.boost.OptimizerProcess (*opt*)

处理 Boost 的优化器。目前支持给优化器添加梯度中心化和创建新的优化器。

参数：

- **opt** (Cell) - 使用的优化器。

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, Parameter, nn
>>> from mindspore import ops
>>> from mindspore.boost import OptimizerProcess
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
float32))),
...                                     name='weight')
...         self.matmul = ops.MatMul()
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output
...
>>> size, in_features, out_features = 16, 16, 10
>>> network = Net(in_features, out_features)
>>> optimizer = nn.Momentum(network.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> optimizer_process = OptimizerProcess(optimizer)
>>> optimizer_process.add_grad_centralization(network)
>>> optimizer = optimizer_process.generate_new_optimizer()
```

add_grad_centralization (*network*)

添加梯度中心化。

参数：

- **network** (Cell) - 训练网络。

static build_gc_params_group (*params_dict, parameters*)
构建梯度中心化的分组合重。

参数：

- **params_dict** (dict) - 训练权重的字典。
- **parameters** (list) - 训练权重的列表。

static build_params_dict (*network*)
构建网络权重的字典。

参数：

- **network** (Cell) - 训练网络。

generate_new_optimizer ()
生成新的优化器。

class mindspore.boost.ParameterProcess

处理 Boost 网络的权重。当前支持创建分组参数和自动设置网络梯度切分点。

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, Parameter, nn
>>> from mindspore import ops
>>> from mindspore.boost import ParameterProcess
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
↪float32))),
...                                     name='weight')
...         self.weight2 = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
↪float32))),
...                                     name='weight2')
...         self.matmul = ops.MatMul()
...         self.matmul2 = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         output2 = self.matmul2(x, self.weight2)
...         return output + output2
...
>>> size, in_features, out_features = 16, 16, 10
>>> network = Net(in_features, out_features)
>>> new_parameter = network.trainable_params()[0]
>>> group_params = ParameterProcess.generate_group_params(new_parameter, network.trainable_
↪params())
```

assign_parameter_group (*parameters, split_point=None*)
设置分组合重。

参数：

- **parameters** (list) - 训练网络的权重。
- **split_point** (list) - 网络梯度切分点。默认为 None。

static generate_group_params (*parameters, origin_params*)

创建分组权重。

参数：

- **parameters** (list) - 训练网络的新权重。
- **origin_params** (list) - 训练网络的初始权重。

class mindspore.boost.BoostTrainOneStepCell (*network, optimizer, sens=None*)

Boost 网络训练封装类。

用优化器封装网络。使用输入训练网络来获取结果。反向图在 *construct* 函数中创建，以更新参数，并且支持多种不同的并行模式。

参数：

- **network** (Cell) - 训练网络，当前网络只支持单个输出。
- **optimizer** (Union[Cell]) - 用于更新网络参数的优化器。
- **sens** (numbers.Number) - 作为反向传播输入要填充的缩放数，默认值：None，取 1.0。

输入：

- ***inputs** (Tuple(Tensor)) - 网络的所有输入组成的元组，其 shape 为 $(N, \dots)$ 。

输出：

Tuple，包含三个 Tensor，分别为损失函数值、溢出状态和当前损失缩放系数。shape 为 $()$ 。

- **loss**(Tensor)，标量 Tensor。
- **overflow**(Tensor)，标量 Tensor，类型为 bool。
- **loss scaling value**(Tensor)，标量 Tensor。

异常：

- **TypeError** - 如果 *sens* 不是一个数字。

支持平台：

Ascend GPU CPU

样例：

```
>>> from mindspore import boost
>>> from mindspore import nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits()
>>> optim = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> #1) Using the WithLossCell existing provide
>>> loss_net = nn.WithLossCell(net, loss_fn)
>>> train_net = boost.BoostTrainOneStepCell(loss_net, optim)
>>>
>>> #2) Using user-defined WithLossCell
>>> class MyWithLossCell(nn.Cell):
...     def __init__(self, backbone, loss_fn):
...         super(MyWithLossCell, self).__init__(auto_prefix=False)
...         self._backbone = backbone
...         self._loss_fn = loss_fn
... 
```

(下页继续)

(续上页)

```

...     def construct(self, x, y, label):
...         out = self._backbone(x, y)
...         return self._loss_fn(out, label)
...
...     @property
...     def backbone_network(self):
...         return self._backbone
...
>>> loss_net = MyWithLossCell(net, loss_fn)
>>> train_net = boost.BoostTrainOneStepCell(loss_net, optim)

```

adasum_process (*loss, grads*)

使用 Adasum 算法训练。

参数：

- **loss** (Tensor) - 网络训练的 loss 值。shape 为 ()。
- **grads** (tuple(Tensor)) - 网络训练过程中的梯度。

返回：

Tensor, 网络训练过程中得到的 loss 值。shape 为 ()。

check_adasum_enable ()

Adasum 算法仅在多卡或者多机场景生效, 并且要求卡数符合 2 的 n 次方, 该函数用来判断 adasum 算法能否生效。

返回：

enable_adasum (bool), Adasum 算法是否生效。

check_dim_reduce_enable ()

获取当前是否使用降维二阶训练算法训练。

返回：

enable_dim_reduce (bool), 降维二阶训练算法是否生效。

gradient_accumulation_process (*loss, grads, sens, *inputs*)

使用梯度累加算法训练。

参数：

- **loss** (Tensor) - 网络训练的 loss 值。shape 为 ()。
- **grads** (tuple(Tensor)) - 网络训练过程中的梯度。
- **sens** (Tensor) - 作为反向传播输入要填充的缩放数。shape 为 ()。
- **inputs** (tuple(Tensor)) - 网络训练的输入。shape 为 (N,...)。

返回：

Tensor, 网络训练过程中得到的 loss 值, 其 shape 为 ()。

gradient_freeze_process (**inputs*)

使用梯度冻结算法训练。

参数：

- **inputs** (tuple(Tensor)) - 网络训练的输入, 其 shape 为 (N,...)。

返回：

Tensor, 网络训练过程中得到的 loss 值, 其 shape 为 ()。

class mindspore.boost.**BoostTrainOneStepWithLossScaleCell** (*network, optimizer, scale_sense*)

使用混合精度功能的 Boost 训练网络。

实现了包含损失缩放 (loss scale) 的单次训练。它使用网络、优化器和用于更新损失缩放系数 (loss scale) 的 Cell(或一个 Tensor) 作为参数。可在 host 侧或 device 侧更新损失缩放系数。BoostTrainOneStepWithLossScaleCell 会被编译成图, 其中 inputs 作为输入数据。张量类型参数 *scale_sense* 作为损失缩放时使用的值。如果需要在 host 侧更新, 使用 Tensor 作为 *scale_sense*。如果需要在 device 侧更新, 使用可更新损失缩放系数的 Cell 实例作为 *scale_sense*。

参数:

- **network** (Cell) - 训练网络, 当前网络只支持单个输出。
- **optimizer** (Cell) - 用于更新网络参数的优化器。
- **scale_sense** (Union[Tensor, Cell]) - 如果此值为 Cell 类型, *BoostTrainOneStepWithLossScaleCell* 会调用它来更新损失缩放系数。如果此值为 Tensor 类型, 可调用 *mindspore.nn.TrainOneStepWithLossScaleCell.set_sense_scale()* 来更新损失缩放系数, shape 为 () 或 (1,)。

输入:

- ***inputs** (Tuple(Tensor)) - 网络的所有输入组成的元组。shape 为 (N,...)。

输出:

Tuple, 包含三个 Tensor, 分别为损失函数值、溢出状态和当前损失缩放系数。

- **loss** (Tensor) - 标量 Tensor。shape 为 ()。
- **overflow** (Tensor) - 标量 Tensor, 类型为 bool。shape 为 ()。
- **loss scaling value** (Tensor) - 标量 Tensor。shape 为 ()。

异常:

- **TypeError** - *scale_sense* 既不是 Cell, 也不是 Tensor。
- **ValueError** - *scale_sense* 的 shape 既不是 (1,) 也不是 ()。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> from mindspore import Tensor, Parameter, nn
>>> from mindspore import ops
>>> from mindspore.nn import WithLossCell
>>> from mindspore import dtype as mstype
>>> from mindspore import boost
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
float32))),
...                                     name='weight')
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
```

(下页继续)

(续上页)

```

...         return output
...
>>> size, in_features, out_features = 16, 16, 10
>>> #1) when the type of scale_sense is Cell:
>>> net = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = WithLossCell(net, loss)
>>> manager = nn.DynamicLossScaleUpdateCell(loss_scale_value=2**12, scale_factor=2, scale_
↪window=1000)
>>> train_network = boost.BoostTrainOneStepWithLossScaleCell(net_with_loss, optimizer, scale_
↪sense=manager)
>>> input = Tensor(np.ones([out_features, in_features]), mstype.float32)
>>> labels = Tensor(np.ones([out_features,]), mstype.float32)
>>> output = train_network(input, labels)
>>>
>>> #2) when the type of scale_sense is Tensor:
>>> net = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = WithLossCell(net, loss)
>>> inputs = Tensor(np.ones([size, in_features]).astype(np.float32))
>>> label = Tensor(np.zeros([size, out_features]).astype(np.float32))
>>> scaling_sens = Tensor(np.full((1), np.finfo(np.float32).max), dtype=mstype.float32)
>>> train_network = boost.BoostTrainOneStepWithLossScaleCell(net_with_loss, optimizer, scale_
↪sense=scaling_sens)
>>> output = train_network(inputs, label)

```

class mindspore.boost.LessBN (*network, fn_flag=False*)

LessBN 算法, 可以在不损失网络精度的前提下, 自动减少网络中批归一化 (Batch Normalization) 的数量, 来提升网络性能。

参数:

- **network** (Cell) - 待训练的网络模型。
- **fn_flag** (bool) - 是否将网络中最后一个全连接层替换为全归一化层。默认值: False。

样例:

```

>>> import numpy as np
>>> from mindspore import Tensor, Parameter, nn
>>> from mindspore import ops
>>> from mindspore.nn import WithLossCell
>>> from mindspore import dtype as mstype
>>> from mindspore import boost
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
↪float32))),
...                                     name='weight')
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output

```

(下页继续)

(续上页)

```
>>> size, in_features, out_features = 16, 16, 10
>>> net = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = WithLossCell(net, loss)
>>> inputs = Tensor(np.ones([size, in_features]).astype(np.float32))
>>> label = Tensor(np.zeros([size, out_features]).astype(np.float32))
>>> train_network = boost.LessBN(net_with_loss)
>>> output = train_network(inputs, label)
```

class mindspore.boost.GradientFreeze(*param_groups, freeze_type, freeze_p, total_steps*)

梯度冻结算法，根据指定策略随机冻结某些层的梯度，来提升网络训练性能。冻结的层数和冻结的概率均可由用户配置。

参数：

- **param_groups** (Union[tuple, list]) - 梯度冻结训练的权重。
- **freeze_type** (int) - 梯度冻结训练的策略。
- **freeze_p** (float) - 梯度冻结训练的概率。
- **total_steps** (int) - 整个训练过程的总的步数。

样例：

```
>>> import numpy as np
>>> from mindspore import Tensor, Parameter, nn
>>> from mindspore import ops
>>> from mindspore.nn import WithLossCell
>>> from mindspore import dtype as mstype
>>> from mindspore import boost
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
... float32))),
...                                 name='weight')
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output
>>> size, in_features, out_features = 16, 16, 10
>>> net = Net(in_features, out_features)
>>> loss = nn.MSELoss()
>>> optimizer = nn.Momentum(net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> net_with_loss = WithLossCell(net, loss)
>>> gradient_freeze_class = boost.GradientFreeze(10, 1, 0.5, 2000)
>>> network, optimizer = gradient_freeze_class.freeze_generate(net_with_loss, optimizer)
>>> inputs = Tensor(np.ones([size, in_features]).astype(np.float32))
>>> label = Tensor(np.zeros([size, out_features]).astype(np.float32))
>>> output = network(inputs, label)
```

freeze_generate(*network, optimizer*)

生成梯度冻结的网络与优化器。

参数：

- **network** (Cell) - 训练网络。
- **optimizer** (Cell) - 用于更新权重的优化器。

generate_freeze_index_sequence (*parameter_groups_number, freeze_strategy, freeze_p, total_steps*)
生成梯度冻结每一步需要冻结的层数。

参数：

- **parameter_groups_number** (int) - 梯度冻结训练的权重个数。
- **freeze_strategy** (int) - 梯度冻结训练的策略。
- **freeze_p** (float) - 梯度冻结训练的概率。
- **total_steps** (int) - 整个训练过程的总的步数。

split_parameters_groups (*net, freeze_para_groups_number*)
拆分用于梯度冻结训练的权重。

参数：

- **net** (Cell) - 训练网络。
- **freeze_para_groups_number** (int) - 梯度冻结训练的权重个数。

class mindspore.boost.**FreezeOpt** (*opt, train_parameter_groups=None, train_strategy=None*)
支持梯度冻结训练的优化器。

参数：

- **opt** (Cell) - 非冻结优化器实例，如 *Momentum*，*SGD*。
- **train_parameter_groups** (Union[tuple, list]) - 梯度冻结训练的权重。默认值：None。
- **train_strategy** (Union[tuple(int), list(int), Tensor]) - 梯度冻结训练的策略。默认值：None。

支持平台：

Ascend

mindspore.boost.freeze_cell (*reducer_flag, network, optimizer, sens, grad, use_grad_accumulation, mean=None, degree=None, max_accumulation_step=1*)

提供带梯度冻结的网络 Cell。

参数：

- **reducer_flag** (bool) - 是否分布式训练。
- **network** (Cell) - 训练网络。
- **optimizer** (Cell) - 优化器。
- **sens** (numbers.Number) - 损失缩放系数。
- **grad** (tuple(Tensor)) - 网络梯度。
- **use_grad_accumulation** (bool) - 是否使用梯度累加。
- **mean** (bool) - 可选参数，梯度是否求平均，仅分布式训练时生效。默认值：None。
- **degree** (int) - 可选参数，device 卡数，仅分布式训练时生效。默认值：None。
- **max_accumulation_step** (int) - 可选参数，梯度累加步数。默认值：1。

样例：

```

>>> import numpy as np
>>> from mindspore import Tensor, Parameter, nn
>>> from mindspore import ops
>>> from mindspore.boost.grad_freeze import freeze_cell, FreezeOpt
>>>
>>> class Net(nn.Cell):
...     def __init__(self, in_features, out_features):
...         super(Net, self).__init__()
...         self.weight = Parameter(Tensor(np.ones([in_features, out_features]).astype(np.
float32))),
...                                     name='weight')
...         self.matmul = ops.MatMul()
...
...     def construct(self, x):
...         output = self.matmul(x, self.weight)
...         return output
...
>>> in_features, out_features = 16, 10
>>> network = Net(in_features, out_features)
>>> optimizer = nn.Momentum(network.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> optimizer = FreezeOpt(optimizer)
>>> grad = ops.GradOperation(get_by_list=True, sens_param=True)
>>> freeze_nets = freeze_cell(False, network, optimizer, 1.0, grad, False, None, None, 1)

```

class mindspore.boost.GradientAccumulation (*max_accumulation_step, optimizer*)
 梯度累加算法，在累加多个 step 的梯度之后，再用来更新网络权重，可以提高训练效率。

参数：

- **max_accumulation_step** (int) - 累加梯度的步数。
- **optimizer** (Cell) - 网络训练使用的优化器。

class mindspore.boost.AdaSum (*rank, device_number, group_number, parameter_tuple*)

Adaptive Summation(AdaSum) 是一种优化深度学习模型并行训练的算法，它可以提升不同规模集群训练的精度，减小不同规模集群调参难度。

参数：

- **rank** (int) - 总的训练的卡数。
- **device_number** (int) - 单机的卡数。
- **group_number** (int) - 分组的数量。
- **parameter_tuple** (Tuple(Parameter)) - 网络训练权重组成的元组。

输入：

- **delta_weights** (Tuple(Tensor)) - 梯度 tuple。
- **parameters** (Tuple(Parameter)) - 当前权重组成的元组。
- **old_parameters** (Tuple(Parameter)) - 旧的权重组成的元组。

输出：

- **adasum_parameters** (Tuple(Tensor)) - adasum 处理后更新的权重。

class mindspore.boost.DimReduce (*network, optimizer, weight, pca_mat_local, n_components, rho, gamma, alpha, sigma, rank, rank_size*)

降维训练 (dimension reduce training) 是一种优化深度学习模型训练的算法，它可以加速模型的收敛。

算法主要原理：

$$\begin{aligned} grad\_k &= pca\_mat \cdot grad \\ dk &= -bk \cdot grad\_k \\ sk &= rho^m \cdot dk \\ delta\_loss &= sigma \cdot grad\_k.T \cdot sk \end{aligned}$$

其中：

- **pca_mat** (array): PCA 矩阵, 维度 $(k * n)$, k 是 $n\_components$ 的大小, n 是权重的大小。
- **bk** (array): 维度 $(k * k)$, **bk** 是拟牛顿法中的对称正定矩阵。

我们需要找到满足以下条件的 m ：

$$new\_loss < old\_loss + delta\_loss$$

然后使用 **delta_grad** 去更新模型的权重：

$$\begin{aligned} grad\_k\_proj &= pca\_mat.T \cdot grad\_k \\ new\_grad\_momentum &= gamma \cdot old\_grad\_momentum + grad - grad\_k\_proj \\ delta\_grad &= alpha \cdot new\_grad\_momentum - pca\_mat.T \cdot sk \end{aligned}$$

参数：

- **network** (Cell) - 训练网络, 只支持单输出。
- **optimizer** (Union[Cell]) - 更新权重的优化器。
- **weight** (Tuple(Parameter)) - 网络权重组成的元组。
- **pca_mat_local** (numpy.ndarray) - 用于 PCA 操作的, 经过切分的 PCA 转换矩阵, 维度为 $k*n$, k 是切分的 $n\_components$ 的大小, n 是权重的大小。
- **n_components** (int) - PCA 的主成分维度 (components)。
- **rho** (float) - 超参。
- **gamma** (float) - 超参。
- **alpha** (float) - 超参。
- **sigma** (float) - 超参。
- **rank** (int) - Rank 编号。
- **rank_size** (int) - Rank 总数。

输入：

- **loss** (Tensor) - 网络 loss, shape 为 () 的标量 Tensor。
- **old_grad** (Tuple(Tensor)) - 网络权重提取组成的元组。
- **weight** (Tuple(Tensor)) - 网络权重组成的元组。
- **weight_clone** (Tuple(Tensor)) - 网络权重的副本。
- ***inputs** (Tuple(Tensor)) - 网络的所有输入组成的元组, shape 为 $(N, \dots)$ 。

输出：

- **loss** (Tensor) - 网络 loss, shape 为 () 的标量 Tensor。

class mindspore.boost.GroupLossScaleManager (*init_loss_scale, loss_scale_groups*)

增强型混合精度算法支持不同 loss scale 的多层应用和损失尺度的动态更新。

参数：

- **init_loss_scale** (Number) - 初始化 loss scale。
- **loss_scale_groups** (List) - 从参数列表里分离出来的 loss scale 组。

输入：

- **x** (Tensor) - 最后一个 operator 的输出。
- **layer1** (int) - 当前网络层的值。
- **layer2** (int) - 最后一个网络层的值。

输出：

- **out** (Tensor) - 带有分组 loss scale 标记的 Tensor，用来标记当前 Tensor 对应的 loss scale 组号。

支持平台：

Ascend

样例：

```
>>> import mindspore as ms
>>> from mindspore import boost, nn
>>>
>>> class Net(nn.Cell):
...     def __init__(self, enhanced_amp, num_class=10, num_channel=1):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(num_channel, 6, 5, pad_mode='valid')
...         self.conv2 = nn.Conv2d(6, 16, 5, pad_mode='valid')
...         self.fc1 = nn.Dense(16*5*5, 120, weight_init='ones')
...         self.fc2 = nn.Dense(120, 84, weight_init='ones')
...         self.fc3 = nn.Dense(84, num_class, weight_init='ones')
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.flatten = nn.Flatten()
...         self.enhanced_amp = enhanced_amp
...
...     def construct(self, x):
...         x = self.enhanced_amp(x, 0, 1)
...         x = self.max_pool2d(self.relu(self.conv1(x)))
...         x = self.max_pool2d(self.relu(self.conv2(x)))
...         x = self.flatten(x)
...         x = self.enhanced_amp(x, 1, 2)
...         x = self.relu(self.fc1(x))
...         x = self.relu(self.fc2(x))
...         x = self.fc3(x)
...         x = self.enhanced_amp(x, 2, 3)
...         return x
>>>
>>> loss_scale_manager = boost.GroupLossScaleManager(4096, [])
>>> net = Net(loss_scale_manager)
>>> param_group1 = []
>>> param_group2 = []
>>> for param in net.trainable_params():
...     if 'conv' in param.name:
...         param_group1.append(param)
...     else:
...         param_group2.append(param)
>>> loss_scale_manager.loss_scale_groups = [param_group1, param_group2]
```

(下页继续)

(续上页)

```

>>> loss = nn.SoftmaxCrossEntropyWithLogits()
>>> optim = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> boost_config_dict = {"boost": {"mode": "manual", "less_bn": False, "grad_freeze": False,
    ↪ "adasum": False,
    ...                      "grad_accumulation": False, "dim_reduce": False, "loss_scale_group":
    ↪ True}}
>>> model = ms.train.Model(net, loss_fn=loss, optimizer=optim, metrics=None,
    ...                      loss_scale_manager=loss_scale_manager,
    ...                      boost_level="O1", boost_config_dict=boost_config_dict)
>>> # Create the dataset taking MNIST as an example. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/mnist.py
>>> dataset = create_dataset()
>>> model.train(2, dataset)

```

get_loss_scale()

获取 loss scale 的值。

返回:

bool, *loss_scale* 的值。

get_update_cell()

返回 *mindspore.boost.GroupLossScaleManager* 实例。

返回:

mindspore.boost.GroupLossScaleManager 实例。

set_loss_scale_status(*loss_scale_number*, *init_loss_scale*)

生成动态 loss scale 元组，并设置溢出状态列表。

参数:

- **loss_scale_number** (int) - loss scale 的数量。
- **init_loss_scale** (float) - 已初始化的 loss scale。

update_loss_scale_status(*layer*, *update_ratio*)

更新动态 loss scale。

参数:

- **layer** (int) - 当前层。
- **update_ratio** (float) - 更新 loss scale 的当前比例。

输出:

float, 新 loss scale 的值。

用于构造概率网络的高级组件。

22.1 Bijections

接口名	概述	支持平台
<code>mindspore.nn.probability.bijection.Bijection</code>	Bijection 类。	Ascend GPU
<code>mindspore.nn.probability.bijection.Exp</code>	指数 Bijection (Exponential Bijection)。	Ascend GPU
<code>mindspore.nn.probability.bijection.GumbelCDF</code>	GumbelCDF Bijection。	Ascend GPU
<code>mindspore.nn.probability.bijection.Invert</code>	逆映射 Bijection (Invert Bijection)。	Ascend GPU CPU
<code>mindspore.nn.probability.bijection.PowerTransform</code>	乘方 Bijection (PowerTransform Bijection)。	Ascend GPU
<code>mindspore.nn.probability.bijection.ScalarAffine</code>	标量仿射 Bijection (Scalar Affine Bijection)。	Ascend GPU
<code>mindspore.nn.probability.bijection.Softplus</code>	Softplus Bijection。	Ascend GPU CPU

22.1.1 mindspore.nn.probability.bijector.Bijector

class mindspore.nn.probability.bijector.**Bijector** (*is_constant_jacobian=False, is_injective=True, name=None, dtype=None, param=None*)

Bijector 类。Bijector 描述了一种随机变量的映射方法。可以通过一个已有的随机变量 X 和一个映射函数 $g(x)$ 生成一个新的随机变量 $Y = g(X)$ 。

参数：

- **is_constant_jacobian** (bool) - Bijector 是否具有常数导数。默认值：False。
- **is_injective** (bool) - Bijector 是否为一对一映射。默认值：True。
- **name** (str) - Bijector 名称。默认值：None。
- **dtype** (mindspore.dtype) - Bijector 可以操作的分布的数据类型。默认值：None。
- **param** (dict) - 用于初始化 Bijector 的参数。默认值：None。

说明：

- Bijector 的 *dtype* 为 None 时，输入值必须是 float 类型，除此之外没有其他强制要求。
- 在初始化过程中，当 *dtype* 为 None 时，对参数的数据类型没有强制要求。但所有参数都应具有相同的 float 类型，否则将引发 `TypeError`。

参数类型跟随输入值的数据类型，具体来说：

- 当 *dtype* 为 None 时，Bijector 的参数将被强制转换为与输入值相同的类型。
- 当指定了 *dtype* 时，参数和输入值的 *dtype* 必须相同，否则将引发 `TypeError`。
- 只能使用 `mindspore.float_type` 数据类型来指定 Bijector 的 *dtype*。

支持平台：

Ascend GPU

cast_param_by_value (*value, para*)

将输入中的 *para* 的数据类型转换为与 *value* 相同的类型。一般用于 Bijector 的子类对自身参数进行数据类型转换。

参数：

- **value** (Tensor) - 输入数据。
- **para** (Tensor) - Bijector 参数。

返回：

Tensor，参数经过数据类型转换之后的值。

construct (*name, *args, **kwargs*)

重写 Cell 中的 *construct*。

说明：支持的函数名称包括：'forward'、'inverse'、'forward_log_jacobian'、'inverse_log_jacobian'。

参数：

- **name** (str) - 函数名称。

- **args** (list) - 函数所需的位置参数列表。
- **kwargs** (dict) - 函数所需的关键字参数字典。

返回：

Tensor, name 对应函数的值。

forward (*value*, **args*, ***kwargs*)

正映射，计算输入随机变量经过映射后的值。

参数：

- **value** (Tensor) - 输入随机变量的值。
- **args** (list) - 函数所需的位置参数列表。
- **kwargs** (dict) - 函数所需的关键字参数字典。

返回：

Tensor, 输出随机变量经过映射后的值。

forward_log_jacobian (*value*, **args*, ***kwargs*)

计算正映射导数的对数值。

参数：

- **value** (Tensor) - 输入随机变量的值。
- **args** (list) - 函数所需的位置参数列表。
- **kwargs** (dict) - 函数所需的关键字参数字典。

返回：

Tensor, 正映射导数的对数值。

inverse (*value*, **args*, ***kwargs*)

逆映射，计算输出随机变量对应的输入随机变量的值。

参数：

- **value** (Tensor) - 输出随机变量的值。
- **args** (list) - 函数所需的位置参数列表。
- **kwargs** (dict) - 函数所需的关键字参数字典。

返回：

Tensor, 输入随机变量的值。

inverse_log_jacobian (*value*, **args*, ***kwargs*)

计算逆映射导数的对数值。

参数：

- **value** (Tensor) - 输出随机变量的值。
- **args** (list) - 函数所需的位置参数列表。
- **kwargs** (dict) - 函数所需的关键字参数字典。

返回：

Tensor, 逆映射导数的对数值。

22.1.2 mindspore.nn.probability.bijector.Exp

class mindspore.nn.probability.bijector.**Exp** (*name='Exp'*)
指数 Bijector (Exponential Bijector)。此 Bijector 对应的映射函数为：

$$Y = \exp(X).$$

参数：

- **name** (str) - Bijector 名称。默认值：'Exp'。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>>
>>> # To initialize an Exp bijector.
>>> exp_bijector = nn.probability.bijector.Exp()
>>> value = Tensor([1, 2, 3], dtype=mindspore.float32)
>>> ans1 = exp_bijector.forward(value)
>>> print(ans1.shape)
(3,)
>>> ans2 = exp_bijector.inverse(value)
>>> print(ans2.shape)
(3,)
>>> ans3 = exp_bijector.forward_log_jacobian(value)
>>> print(ans3.shape)
(3,)
>>> ans4 = exp_bijector.inverse_log_jacobian(value)
>>> print(ans4.shape)
(3,)
```

forward (*value*)

正映射，计算输入随机变量经过映射后的值，即 $Y = \exp(X)$ 。

参数：

- **value** (Tensor) - 输入随机变量的值，公式中的 X 。

返回：

Tensor，输出随机变量映射后的值。

forward_log_jacobian (*value*)

计算正映射导数的对数值。

参数：

- **value** (Tensor) - 输入随机变量的值。

返回：

Tensor，正映射导数的对数值。

inverse (*value*)

逆映射，计算输出随机变量对应的输入随机变量的值，即 $X = \log(Y)$ 。

参数：

- **value** (Tensor) - 输出随机变量的值，公式中的 Y 。

返回：

Tensor，输入随机变量的值。

inverse_log_jacobian (*value*)

计算逆映射导数的对数值。

参数：

- **value** (Tensor) - 输出随机变量的值。

返回：

Tensor，逆映射导数的对数值。

22.1.3 mindspore.nn.probability.bijector.GumbelCDF

class mindspore.nn.probability.bijector.**GumbelCDF** (*loc=0.0, scale=1.0, name='GumbelCDF'*)

GumbelCDF Bijector。此 Bijector 对应的映射函数为：

$$Y = \exp(-\exp(\frac{-(X - loc)}{scale}))$$

参数：

- **loc** (float, list, numpy.ndarray, Tensor) - 位移因子，即上述公式中的 *loc*。默认值：0.0。
- **scale** (float, list, numpy.ndarray, Tensor) - 比例因子，即上述公式中的 *scale*。默认值：1.0。
- **name** (str) - Bijector 名称。默认值：'GumbelCDF'。

说明：

- *scale* 中元素必须大于零。
- 对于 *inverse* 和 *inverse_log_jacobian*，输入应在 (0, 1) 范围内。
- *loc* 和 *scale* 中元素的数据类型必须为 float。
- 如果 *loc*、*scale* 作为 numpy.ndarray 或 Tensor 传入，则它们必须具有相同的数据类型，否则将引发错误。

异常：

- **TypeError** - *loc* 或 *scale* 中元素的数据类型不为 float，或 *loc* 和 *scale* 中元素的数据类型不相同。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.bijector as msb
>>> from mindspore import Tensor
>>>
>>> # To initialize a GumbelCDF bijector of loc 1.0, and scale 2.0.
>>> gumbel_cdf = msb.GumbelCDF(1.0, 2.0)
```

(下页继续)

```
>>> # To use a GumbelCDF bijector in a network.
>>> x = Tensor([1, 2, 3], dtype=mindspore.float32)
>>> y = Tensor([0.1, 0.2, 0.3], dtype=mindspore.float32)
>>> ans1 = gumbel_cdf.forward(x)
>>> print(ans1.shape)
(3,)
>>> ans2 = gumbel_cdf.inverse(y)
>>> print(ans2.shape)
(3,)
>>> ans3 = gumbel_cdf.forward_log_jacobian(x)
>>> print(ans3.shape)
(3,)
>>> ans4 = gumbel_cdf.inverse_log_jacobian(y)
>>> print(ans4.shape)
(3,)
```

property loc

返回分布位置。

返回：

Tensor，分布的位置值。

property scale

返回分布比例。

返回：

Tensor，分布的比例值。

forward(value)

正映射，计算输入随机变量经过映射后的值。

参数：

- **value** (Tensor) - 输入随机变量的值。

返回：

Tensor，输入随机变量的值。

forward_log_jacobian(value)

计算正映射导数的对数值。

参数：

- **value** (Tensor) - 输入随机变量的值。

返回：

Tensor，正映射导数的对数值。

inverse(value)

逆映射，计算输出随机变量对应的输入随机变量的值。

参数：

- **value** (Tensor) - 输出随机变量的值。

返回：

Tensor，输入随机变量的值。

inverse_log_jacobian (*value*)

计算逆映射导数的对数值。

参数：

- **value** (Tensor) - 输出随机变量的值。

返回：

Tensor，逆映射导数的对数值。

22.1.4 mindspore.nn.probability.bijector.Invert

class mindspore.nn.probability.bijector.**Invert** (*bijector*, *name*=")

逆映射 Bijector (Invert Bijector)。计算输入 Bijector 的逆映射。如果正向映射（下面的 *bijector* 输入）对应的映射函数为 $Y = g(X)$ ，那么对应的逆映射 Bijector 的映射函数为 $Y = h(X) = g^{-1}(X)$ 。

参数：

- **bijector** (Bijector) - 基础 Bijector (Base Bijector)。
- **name** (str) - Bijector 名称。默认值：""。当设置为 "" 时，实际上是 'Invert' + Bijector.name。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.bijector as msb
>>> from mindspore import Tensor
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.origin = msb.ScalarAffine(scale=2.0, shift=1.0)
...         self.invert = msb.Invert(self.origin)
...
...     def construct(self, x_):
...         return self.invert.forward(x_)
>>> forward = Net()
>>> x = np.array([2.0, 3.0, 4.0, 5.0]).astype(np.float32)
>>> ans = forward(Tensor(x, dtype=mindspore.float32))
>>> print(ans.shape)
(4,)
```

property bijector

Bijector 类，返回基础 Bijector。

forward (*x*)

计算基础 Bijector 的逆映射，即 $Y = h(X) = g^{-1}(X)$ 。

参数：

- **x** (Tensor) - 基础 Bijector 的输出随机变量的值。

返回：

Tensor，基础 Bijector 的输入随机变量的值。

forward_log_jacobian(*x*)

计算基础 Bijector 的逆映射导数的对数值, 即 $\log dg^{-1}(x)/dx$ 。

参数:

- **x** (Tensor) - 基础 Bijector 的输出随机变量的值。

返回:

Tensor, 基类逆映射导数的对数值。

inverse(*y*)

计算基础 Bijector 的正映射, 即 $Y = g(X)$ 。

参数:

- **y** (Tensor) - 基础 Bijector 的输入随机变量的值。

返回:

Tensor, 基础 Bijector 的输出随机变量的值。

inverse_log_jacobian(*y*)

计算基础 Bijector 的正映射导数的对数, 即 $Y = \log dg(x)/dx$ 。

参数:

- **y** (Tensor) - 基础 Bijector 的输入随机变量的值。

返回:

Tensor, 基类正映射导数的对数值。

22.1.5 mindspore.nn.probability.bijector.PowerTransform

class mindspore.nn.probability.bijector.**PowerTransform**(*power=0., name='PowerTransform'*)

乘方 Bijector (PowerTransform Bijector)。此 Bijector 对应的映射函数为:

$$Y = g(X) = (1 + X * c)^{1/c}, X \geq -1/c$$

其中 $c \geq 0$ 。

PowerTransform Bijector 将输入从 $[-1/c, \inf]$ 映射到 $[0, \inf]$ 。

当 $c=0$ 时, 此 Bijector 等于 `mindspore.nn.probability.bijector.Exp` Bijector。

参数:

- **power** (float, list, numpy.ndarray, Tensor) - 比例因子。默认值: 0。
- **name** (str) - Bijector 名称。默认值: 'PowerTransform'。

说明: *power* 中元素的数据类型必须为 float。

异常:

- **ValueError** - *power* 中元素小于 0 或静态未知。
- **TypeError** - *power* 中元素的数据类型不是 float。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.bijector as msb
>>> from mindspore import Tensor
>>> # To initialize a PowerTransform bijector of power 0.5.
>>> powertransform = msb.PowerTransform(0.5)
>>> value = Tensor([1, 2, 3], dtype=mindspore.float32)
>>> ans1 = powertransform.forward(value)
>>> print(ans1.shape)
(3,)
>>> ans2 = powertransform.inverse(value)
>>> print(ans2.shape)
(3,)
>>> ans3 = powertransform.forward_log_jacobian(value)
>>> print(ans3.shape)
(3,)
>>> ans4 = powertransform.inverse_log_jacobian(value)
>>> print(ans4.shape)
(3,)
```

property power

返回指数。

返回:

Tensor, Bijector 的指数。

forward(value)

正映射, 计算输入随机变量经过映射后的值。

参数:

- **value** (Tensor) - 输入随机变量的值。

返回:

Tensor, 输出随机变量的值。

forward_log_jacobian(value)

计算正映射导数的对数值。

参数:

- **value** (Tensor) - 输入随机变量的值。

返回:

Tensor, 正映射导数的对数值。

inverse(value)

逆映射, 计算输出随机变量对应的输入随机变量的值。

参数:

- **value** (Tensor) - 输出随机变量的值。

返回:

Tensor, 输入随机变量的值。

inverse_log_jacobian(value)

计算逆映射导数的对数值。

参数：

- **value** (Tensor) - 输出随机变量的值。

返回：

Tensor，逆映射导数的对数值。

22.1.6 mindspore.nn.probability.bijector.ScalarAffine

class mindspore.nn.probability.bijector.**ScalarAffine** (*scale=1.0, shift=0.0, name='ScalarAffine'*)

标量仿射 Bijector (Scalar Affine Bijector)。此 Bijector 对应的映射函数为：

$$Y = a * X + b$$

其中 a 是比例因子， b 是移位因子。

参数：

- **scale** (float, list, numpy.ndarray, Tensor) - 比例因子，公式中的 a 。默认值：1.0。
- **shift** (float, list, numpy.ndarray, Tensor) - 移位因子，公式中的 b 。默认值：0.0。
- **name** (str) - Bijector 名称。默认值：'ScalarAffine'。

说明：*shift* 和 *scale* 中元素的数据类型必须为 float。如果 *shift*、*scale* 作为 numpy.ndarray 或 Tensor 传入，则它们必须具有相同的数据类型，否则将引发错误。

异常：

- **TypeError** - *shift* 或 *scale* 中元素的数据类型不为 float，或 *shift* 和 *scale* 的数据类型不相同。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>>
>>> # To initialize a ScalarAffine bijector of scale 1.0 and shift 2.
>>> scalaraffine = nn.probability.bijector.ScalarAffine(1.0, 2.0)
>>> value = Tensor([1, 2, 3], dtype=mindspore.float32)
>>> ans1 = scalaraffine.forward(value)
>>> print(ans1.shape)
(3,)
>>> ans2 = scalaraffine.inverse(value)
>>> print(ans2.shape)
(3,)
>>> ans3 = scalaraffine.forward_log_jacobian(value)
>>> print(ans3.shape)
()
>>> ans4 = scalaraffine.inverse_log_jacobian(value)
>>> print(ans4.shape)
()
```


property shift

返回映射的位置。

返回：

Tensor，映射的位置值。

property scale

返回映射的比例。

返回：

Tensor，映射的比例值。

forward (value)

正映射，计算输入随机变量经过映射后的值。

参数：

- **value** (Tensor) - 输入随机变量的值。

返回：

Tensor，输入随机变量经过映射后的值。

forward_log_jacobian (value)

计算正映射导数的对数值。

参数：

- **value** (Tensor) - 输入随机变量的值。

返回：

Tensor，正映射导数的对数值。

inverse (value)

逆映射，计算输出随机变量对应的输入随机变量的值。

参数：

- **value** (Tensor) - 输出随机变量的值。

返回：

Tensor，输入随机变量的值。

inverse_log_jacobian (value)

计算逆映射导数的对数值。

参数：

- **value** (Tensor) - 输出随机变量的值。

返回：

Tensor，逆映射导数的对数值。

22.1.7 mindspore.nn.probability.bijector.Softplus

class mindspore.nn.probability.bijector.**Softplus** (*sharpness=1.0, name='Softplus'*)
Softplus Bijector。此 Bijector 对应的映射函数为：

$$Y = \frac{\log(1 + e^{kX})}{k}$$

其中 k 是锐度因子。

参数：

- **sharpness** (float, list, numpy.ndarray, Tensor) - 锐度因子，上述公式中的 k 。默认值：1.0。
- **name** (str) - Bijector 名称。默认值：'Softplus'。

说明： *sharpness* 中元素的数据类型必须为 float。

异常：

- **TypeError** - sharpness 中元素的数据类型不为 float。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.bijector as msb
>>> from mindspore import Tensor
>>>
>>> # To initialize a Softplus bijector of sharpness 2.0.
>>> softplus = msb.Softplus(2.0)
>>> # To use a Softplus bijector in a network.
>>> value = Tensor([1, 2, 3], dtype=mindspore.float32)
>>> ans1 = softplus.forward(value)
>>> print(ans1.shape)
(3,)
>>> ans2 = softplus.inverse(value)
>>> print(ans2.shape)
(3,)
>>> ans3 = softplus.forward_log_jacobian(value)
>>> print(ans3.shape)
(3,)
>>> ans4 = softplus.inverse_log_jacobian(value)
>>> print(ans4.shape)
(3,)
```

property sharpness

返回映射的锐度因子。

返回：

Tensor，映射的锐度因子。

forward (*value*)

正映射，计算输入随机变量经过映射后的值。

参数:

- **value** (Tensor) - 输入随机变量的值。

返回:

Tensor, 输入随机变量经过映射后的值。

forward_log_jacobian (*value*)

计算正映射导数的对数值。

参数:

- **value** (Tensor) - 输入随机变量的值。

返回:

Tensor, 正映射导数的对数值。

inverse (*value*)

逆映射, 计算输出随机变量对应的输入随机变量的值。

参数:

- **value** (Tensor) - 输出随机变量的值。

返回:

Tensor, 对应的输入随机变量的值。

inverse_log_jacobian (*value*)

计算逆映射导数的对数值。

参数:

- **value** (Tensor) - 输出随机变量的值。

返回:

Tensor, 逆映射导数的对数值。

22.2 Distributions

接口名	概述	支持平台
<i>mindspore.nn.probability.distribution.Bernoulli</i>	伯努利分布 (Bernoulli Distribution)。	Ascend GPU
<i>mindspore.nn.probability.distribution.Beta</i>	Beta 分布 (Beta Distribution)。	Ascend
<i>mindspore.nn.probability.distribution.Categorical</i>	分类分布。	Ascend GPU
<i>mindspore.nn.probability.distribution.Cauchy</i>	柯西分布 (Cauchy distribution)。	Ascend
<i>mindspore.nn.probability.distribution.Distribution</i>	所有分布的基类。	Ascend GPU
<i>mindspore.nn.probability.distribution.Exponential</i>	指数分布 (Exponential Distribution)。	Ascend GPU
<i>mindspore.nn.probability.distribution.Gamma</i>	伽马分布 (Gamma distribution)。	Ascend

下页继续

表 2 - 续上页

<code>mindspore.nn.probability.distribution.Geometric</code>	几何分布 (Geometric Distribution)。	Ascend GPU
<code>mindspore.nn.probability.distribution.Gumbel</code>	Gumbel 分布 (Gumbel distribution)。	Ascend GPU
<code>mindspore.nn.probability.distribution.HalfNormal</code>	半正态分布 (HalfNormal distribution)。	CPU
<code>mindspore.nn.probability.distribution.Laplace</code>	拉普拉斯分布 (Laplace distribution)。	Ascend GPU CPU
<code>mindspore.nn.probability.distribution.Logistic</code>	Logistic 分布 (Logistic distribution)。	Ascend GPU
<code>mindspore.nn.probability.distribution.LogNormal</code>	对数正态分布 (LogNormal distribution)。	Ascend GPU
<code>mindspore.nn.probability.distribution.Normal</code>	正态分布 (Normal distribution)。	Ascend GPU
<code>mindspore.nn.probability.distribution.Poisson</code>	泊松分布 (Poisson Distribution)。	Ascend
<code>mindspore.nn.probability.distribution.StudentT</code>	StudentT 分布 (StudentT distribution)。	CPU
<code>mindspore.nn.probability.distribution.TransformedDistribution</code>	转换分布 (Transformed Distribution)。	Ascend GPU CPU
<code>mindspore.nn.probability.distribution.Uniform</code>	均匀分布 (Uniform Distribution)。	Ascend GPU CPU

22.2.1 mindspore.nn.probability.distribution.Bernoulli

class mindspore.nn.probability.distribution.**Bernoulli** (*probs=None, seed=None, dtype=mstype.int32, name='Bernoulli'*)

伯努利分布 (Bernoulli Distribution)。离散随机分布，取值范围为 $\{0, 1\}$ ，概率质量函数为 $P(X = 0) = p, P(X = 1) = 1 - p$ 。

参数：

- **probs** (float, list, numpy.ndarray, Tensor) - 结果是 1 的概率。默认值：None。
- **seed** (int) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype) - 采样结果的数据类型。默认值：mstype.int32。
- **name** (str) - 分布的名称。默认值：'Bernoulli'。

说明：*probs* 中元素必须是合适的概率 ($0 < p < 1$)。*dist_spec_args* 是 *probs*。

异常：

- **ValueError** - *probs* 中元素小于等于 0 或大于等于 1。

支持平台：

Ascend GPU

样例：

```

>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Bernoulli distribution of the probability 0.5.
>>> b1 = msd.Bernoulli(0.5, dtype=mindspore.int32)
>>> # A Bernoulli distribution can be initialized without arguments.
>>> # In this case, `probs` must be passed in through arguments during function calls.
>>> b2 = msd.Bernoulli(dtype=mindspore.int32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1, 0, 1], dtype=mindspore.int32)
>>> probs_a = Tensor([0.6], dtype=mindspore.float32)
>>> probs_b = Tensor([0.2, 0.3, 0.4], dtype=mindspore.float32)
>>> # Private interfaces of probability functions corresponding to public interfaces,
    ↪including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`, are the
    ↪same as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     probs1 (Tensor): the probability of success. Default: self.probs.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing `prob` by the name of the function.
>>> ans = b1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate `prob` with respect to distribution b.
>>> ans = b1.prob(value, probs_b)
>>> print(ans.shape)
(3,)
>>> # `probs` must be passed in during function calls.
>>> ans = b2.prob(value, probs_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `sd`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     probs1 (Tensor): the probability of success. Default: self.probs.
>>> # Examples of `mean`. `sd`, `var`, and `entropy` are similar.
>>> ans = b1.mean() # return 0.5
>>> print(ans.shape)
()
>>> ans = b1.mean(probs_b) # return probs_b
>>> print(ans.shape)
(3,)
>>> # `probs` must be passed in during function calls.
>>> ans = b2.mean(probs_a)
>>> print(ans.shape)
(1,)
>>> # Interfaces of `kl_loss` and `cross_entropy` are the same as follows:
>>> # Args:
>>> #     dist (str): the name of the distribution. Only 'Bernoulli' is supported.
>>> #     probs1_b (Tensor): the probability of success of distribution b.
>>> #     probs1_a (Tensor): the probability of success of distribution a. Default: self.
    ↪probs.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = b1.kl_loss('Bernoulli', probs_b)
>>> print(ans.shape)

```

(下页继续)

```

(3,)
>>> ans = b1.kl_loss('Bernoulli', probs_b, probs_a)
>>> print(ans.shape)
(3,)
>>> # An additional `probs_a` must be passed in.
>>> ans = b2.kl_loss('Bernoulli', probs_b, probs_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.
>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ().
>>> #     probs1 (Tensor): the probability of success. Default: self.probs.
>>> ans = b1.sample()
>>> print(ans.shape)
()
>>> ans = b1.sample((2, 3))
>>> print(ans.shape)
(2, 3)
>>> ans = b1.sample((2, 3), probs_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = b2.sample((2, 3), probs_a)
>>> print(ans.shape)
(2, 3, 1)

```

property probs

返回结果为 1 的概率。

返回:

Tensor, 结果为 1 的概率。

cdf (value, probs1)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数:

- **value** (Tensor) - 要计算的值。
- **probs1** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 累积分布函数的值。

cross_entropy (dist, probs1_b, probs1_a)

计算分布 a 和 b 之间的交叉熵。

参数:

- **dist** (str) - 分布的类型。
- **probs1_b** (Tensor) - 分布 b 的伯努利实验成功的概率。
- **probs1_a** (Tensor) - 分布 a 的伯努利实验成功的概率。默认值: None。

返回:

Tensor, 交叉熵的值。

entropy (probs1=None)

计算熵。

参数：

- **probs1** (Tensor) - 分布的伯努利实验成功的概率。默认值：None。

返回：

Tensor, 熵的值。

kl_loss (*dist, probs1_b, probs1_a*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数：

- **dist** (str) - 分布的类型。
- **probs1_b** (Tensor) - 分布 b 的伯努利实验成功的概率。
- **probs1_a** (Tensor) - 分布 a 的伯努利实验成功的概率。默认值：None。

返回：

Tensor, KL 散度。

log_cdf (*value, probs1*)

计算给定值对应的累积分布函数的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **probs1** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor, 累积分布函数的对数。

log_prob (*value, probs1*)

计算给定值对应的概率的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **probs1** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor, 概率的对数。

log_survival (*value, probs1*)

计算给定值对应的生存函数的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **probs1** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor, 生存函数的对数。

mean (*probs1*)

计算期望。

参数：

- **probs1** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回:

Tensor, 概率分布的期望。

mode (*probs1*)

计算众数。

参数:

- **probs1** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 概率分布的众数。

prob (*value*, *probs1*)

计算给定值的概率。对于离散分布是计算概率质量函数 (Probability Mass Function)。

参数:

- **value** (Tensor) - 要计算的值。
- **probs1** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 概率值。

sample (*shape*, *probs1*)

采样函数。

参数:

- **shape** (tuple) - 样本的 shape。
- **probs1** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 根据概率分布采样的样本。

sd (*probs1*)

计算标准差。

参数:

- **probs1** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 概率分布的标准差。

survival_function (*value*, *probs1*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **probs1** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 生存函数的值。

var (*probs1*)

计算方差。

参数:

- **probs1** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 概率分布的方差。

22.2.2 mindspore.nn.probability.distribution.Beta

class mindspore.nn.probability.distribution.**Beta** (*concentration1=None, concentration0=None, seed=None, dtype=mstype.float32, name='Beta'*)

Beta 分布 (Beta Distribution)。连续随机分布, 取值范围为 [0, 1], 概率密度函数为:

$$f(x, \alpha, \beta) = x^{\alpha}(1-x)^{\beta-1}/B(\alpha, \beta)$$

其中 B 为 Beta 函数。

参数:

- **concentration1** (int, float, list, numpy.ndarray, Tensor) - Beta 分布的 α 。默认值: None。
- **concentration0** (int, float, list, numpy.ndarray, Tensor) - Beta 分布的 β 。默认值: None。
- **seed** (int) - 采样时使用的种子。如果为 None, 则使用全局种子。默认值: None。
- **dtype** (mindspore.dtype) - 采样结果的数据类型。默认值: mstype.float32。
- **name** (str) - 分布的名称。默认值: 'Beta'。

说明:

- *concentration1* 和 *concentration0* 中元素必须大于零。
- *dist_spec_args* 是 *concentration1* 和 *concentration0*。
- *dtype* 必须是 float, 因为 Beta 分布是连续的。

异常:

- **ValueError** - *concentration1* 或者 *concentration0* 中元素小于或等于 0。
- **TypeError** - *dtype* 不是 float 或 float 的子类。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Beta distribution of the concentration1 3.0 and the concentration0 4.0.
>>> b1 = msd.Beta([3.0], [4.0], dtype=mindspore.float32)
>>> # A Beta distribution can be initialized without arguments.
>>> # In this case, `concentration1` and `concentration0` must be passed in through
    arguments.
>>> b2 = msd.Beta(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
```

(下页继续)

(续上页)

```

>>> value = Tensor([0.1, 0.5, 0.8], dtype=mindspore.float32)
>>> concentration1_a = Tensor([2.0], dtype=mindspore.float32)
>>> concentration0_a = Tensor([2.0, 2.0, 2.0], dtype=mindspore.float32)
>>> concentration1_b = Tensor([1.0], dtype=mindspore.float32)
>>> concentration0_b = Tensor([1.0, 1.5, 2.0], dtype=mindspore.float32)
>>> # Private interfaces of probability functions corresponding to public interfaces,
    including
>>> # `prob` and `log_prob`, have the same arguments as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     concentration1 (Tensor): the concentration1 of the distribution. Default: self._
    concentration1.
>>> #     concentration0 (Tensor): the concentration0 of the distribution. Default: self._
    concentration0.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing 'prob' by the name of the function
>>> ans = b1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to the distribution b.
>>> ans = b1.prob(value, concentration1_b, concentration0_b)
>>> print(ans.shape)
(3,)
>>> # `concentration1` and `concentration0` must be passed in during function calls
>>> ans = b2.prob(value, concentration1_a, concentration0_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `sd`, `mode`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     concentration1 (Tensor): the concentration1 of the distribution. Default: self._
    concentration1.
>>> #     concentration0 (Tensor): the concentration0 of the distribution. Default: self._
    concentration0.
>>> # Example of `mean`, `sd`, `mode`, `var`, and `entropy` are similar.
>>> ans = b1.mean()
>>> print(ans.shape)
(1,)
>>> ans = b1.mean(concentration1_b, concentration0_b)
>>> print(ans.shape)
(3,)
>>> # `concentration1` and `concentration0` must be passed in during function calls.
>>> ans = b2.mean(concentration1_a, concentration0_a)
>>> print(ans.shape)
(3,)
>>> # Interfaces of 'kl_loss' and 'cross_entropy' are the same:
>>> # Args:
>>> #     dist (str): the type of the distributions. Only "Beta" is supported.
>>> #     concentration1_b (Tensor): the concentration1 of distribution b.
>>> #     concentration0_b (Tensor): the concentration0 of distribution b.
>>> #     concentration1_a (Tensor): the concentration1 of distribution a.
>>> #     Default: self._concentration1.
>>> #     concentration0_a (Tensor): the concentration0 of distribution a.
>>> #     Default: self._concentration0.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = b1.kl_loss('Beta', concentration1_b, concentration0_b)

```

(下页继续)

(续上页)

```

>>> print(ans.shape)
(3,)
>>> ans = b1.kl_loss('Beta', concentration1_b, concentration0_b, concentration1_a,
↳concentration0_a)
>>> print(ans.shape)
(3,)
>>> # Additional `concentration1` and `concentration0` must be passed in.
>>> ans = b2.kl_loss('Beta', concentration1_b, concentration0_b, concentration1_a,
↳concentration0_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.
>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ()
>>> #     concentration1 (Tensor): the concentration1 of the distribution. Default: self._
↳concentration1.
>>> #     concentration0 (Tensor): the concentration0 of the distribution. Default: self._
↳concentration0.
>>> ans = b1.sample()
>>> print(ans.shape)
(1,)
>>> ans = b1.sample((2, 3))
>>> print(ans.shape)
(2, 3, 1)
>>> ans = b1.sample((2, 3), concentration1_b, concentration0_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = b2.sample((2, 3), concentration1_a, concentration0_a)
>>> print(ans.shape)
(2, 3, 3)

```

property concentration0

返回 concentration0 (也称为 Beta 分布的 beta)。

返回:

Tensor, concentration0 的值。

property concentration1

返回 concentration1 (也称为 Beta 分布的 alpha)。

返回:

Tensor, concentration1 的值。

cdf (value, concentration1, concentration0)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration1** (Tensor) - Beta 分布的 alpha。默认值: None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值: None。

返回:

Tensor, 累积分布函数的值。

cross_entropy (*dist, concentration1_b, concentration0_b, concentration1, concentration0*)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。
- **concentration1_b** (Tensor) - Beta 分布 b 的 alpha。
- **concentration0_b** (Tensor) - Beta 分布 b 的 beta。
- **concentration1** (Tensor) - Beta 分布 a 的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布 a 的 beta。默认值：None。

返回：

Tensor, 交叉熵的值。

entropy (*concentration1, concentration0*)

计算熵。

参数：

- **concentration1** (Tensor) - Beta 分布的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值：None。

返回：

Tensor, 熵的值。

kl_loss (*dist, concentration1_b, concentration0_b, concentration1, concentration0*)

计算 KL 散度, 即 KL(a||b)。

参数：

- **dist** (str) - 分布的类型。
- **concentration1_b** (Tensor) - Beta 分布 b 的 alpha。
- **concentration0_b** (Tensor) - Beta 分布 b 的 beta。
- **concentration1** (Tensor) - Beta 分布 a 的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布 a 的 beta。默认值：None。

返回：

Tensor, KL 散度。

log_cdf (*value, concentration1, concentration0*)

计算给定值的累积分布函数的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **concentration1** (Tensor) - Beta 分布的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值：None。

返回：

Tensor, 累积分布函数的对数。

log_prob (*value, concentration1, concentration0*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration1** (Tensor) - Beta 分布的 alpha。默认值: None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值: None。

返回:

Tensor, 概率的对数。

log_survival (*value, concentration1, concentration0*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration1** (Tensor) - Beta 分布的 alpha。默认值: None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值: None。

返回:

Tensor, 生存函数的对数。

mean (*concentration1, concentration0*)

计算期望。

参数:

- **concentration1** (Tensor) - Beta 分布的 alpha。默认值: None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值: None。

返回:

Tensor, 概率分布的期望。

mode (*concentration1, concentration0*)

计算众数。

参数:

- **concentration1** (Tensor) - Beta 分布的 alpha。默认值: None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值: None。

返回:

Tensor, 概率分布的众数。

prob (*value, concentration1, concentration0*)

计算给定值下的概率。对于连续分布是计算概率密度函数 (Probability Density Function)。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration1** (Tensor) - Beta 分布的 alpha。默认值: None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值: None。

返回:

Tensor, 概率值。

sample (*shape*, *concentration1*, *concentration0*)

采样函数。

参数：

- **shape** (tuple) - 样本的 shape。
- **concentration1** (Tensor) - Beta 分布的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值：None。

返回：

Tensor，根据概率分布采样的样本。

sd (*concentration1*, *concentration0*)

计算标准差。

参数：

- **concentration1** (Tensor) - Beta 分布的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值：None。

返回：

Tensor，概率分布的标准差。

survival_function (*value*, *concentration1*, *concentration0*)

计算给定值对应的生存函数。

参数：

- **value** (Tensor) - 要计算的值。
- **concentration1** (Tensor) - Beta 分布的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值：None。

返回：

Tensor，生存函数的值。

var (*concentration1*, *concentration0*)

计算方差。

参数：

- **concentration1** (Tensor) - Beta 分布的 alpha。默认值：None。
- **concentration0** (Tensor) - Beta 分布的 beta。默认值：None。

返回：

Tensor，概率分布的方差。

22.2.3 mindspore.nn.probability.distribution.Categorical

class mindspore.nn.probability.distribution.**Categorical** (*probs=None, seed=None, dtype=mstype.int32, name='Categorical'*)

分类分布。离散随机分布，取值范围为 $\{1, 2, \dots, k\}$ ，概率质量函数为 $P(X = i) = p_i, i = 1, \dots, k$ 。

参数：

- **probs** (Tensor, list, numpy.ndarray) - 事件概率。默认值：None。
- **seed** (int) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值：mstype.int32。
- **name** (str) - 分布的名称。默认值：'Categorical'。

说明：*probs* 的秩必须至少为 1，值是合适的概率，并且总和为 1。

异常：

- **ValueError** - *probs* 的秩为 0，或者其中所有元素的和不等于 1。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Categorical distribution of probs [0.5, 0.5]
>>> ca1 = msd.Categorical(probs=[0.2, 0.8], dtype=mindspore.int32)
>>> # A Categorical distribution can be initialized without arguments.
>>> # In this case, `probs` must be passed in through arguments during function calls.
>>> ca2 = msd.Categorical(dtype=mindspore.int32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1, 0], dtype=mindspore.int32)
>>> probs_a = Tensor([0.5, 0.5], dtype=mindspore.float32)
>>> probs_b = Tensor([0.35, 0.65], dtype=mindspore.float32)
>>> # Private interfaces of probability functions corresponding to public interfaces,
    including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`, are the
    same as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     probs (Tensor): event probabilities. Default: self.probs.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing `prob` by the name of the function.
>>> ans = ca1.prob(value)
>>> print(ans.shape)
(2,)
>>> # Evaluate `prob` with respect to distribution b.
>>> ans = ca1.prob(value, probs_b)
>>> print(ans.shape)
(2,)
```

(下页继续)

(续上页)

```

>>> # `probs` must be passed in during function calls.
>>> ans = ca2.prob(value, probs_a)
>>> print(ans.shape)
(2,)
>>> # Functions `mean`, `sd`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     probs (Tensor): event probabilities. Default: self.probs.
>>> # Examples of `mean`, `sd`, `var`, and `entropy` are similar.
>>> ans = ca1.mean() # return 0.8
>>> print(ans.shape)
(1,)
>>> ans = ca1.mean(probs_b)
>>> print(ans.shape)
(1,)
>>> # `probs` must be passed in during function calls.
>>> ans = ca2.mean(probs_a)
>>> print(ans.shape)
(1,)
>>> # Interfaces of `kl_loss` and `cross_entropy` are the same as follows:
>>> # Args:
>>> #     dist (str): the name of the distribution. Only 'Categorical' is supported.
>>> #     probs_b (Tensor): event probabilities of distribution b.
>>> #     probs (Tensor): event probabilities of distribution a. Default: self.probs.
>>> # Examples of `kl_loss`, `cross_entropy` is similar.
>>> ans = ca1.kl_loss('Categorical', probs_b)
>>> print(ans.shape)
()
>>> ans = ca1.kl_loss('Categorical', probs_b, probs_a)
>>> print(ans.shape)
()
>>> # An additional `probs` must be passed in.
>>> ans = ca2.kl_loss('Categorical', probs_b, probs_a)
>>> print(ans.shape)
()

```

property probs

返回事件发生的概率。

返回：

Tensor，事件发生的概率。

cdf (*value, probs*)

计算给定值的累积分布函数（Cumulative Distribution Function, CDF）。

参数：

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 事件发生的概率。默认值：None。

返回：

Tensor，累积分布函数的值。

cross_entropy (*dist, probs_b, probs*)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。

- **probs_b** (Tensor) - 分布 b 的事件发生概率。
- **probs** (Tensor) - 分布 a 的事件发生概率。默认值: None。

返回:

Tensor, 交叉熵的值。

entropy (*probs*)

计算熵。

参数:

- **probs** (Tensor) - 事件发生的概率。默认值: None。

返回:

Tensor, 熵的值。

kl_loss (*dist, probs_b, probs*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **probs_b** (Tensor) - 分布 b 的事件发生概率。
- **probs** (Tensor) - 分布 a 的事件发生概率。默认值: None。

返回:

Tensor, KL 散度。

log_cdf (*value, probs*)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 事件发生的概率。默认值: None。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value, probs*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 事件发生的概率。默认值: None。

返回:

Tensor, 概率的对数。

log_survival (*value, probs*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 事件发生的概率。默认值: None。

返回：

Tensor，生存函数的对数。

mean (*probs*)

计算期望。

参数：

- **probs** (Tensor) - 事件发生的概率。默认值：None。

返回：

Tensor，概率分布的期望。

mode (*probs*)

计算众数。

参数：

- **probs** (Tensor) - 事件发生的概率。默认值：None。

返回：

Tensor，概率分布的众数。

prob (*value, probs*)

计算给定值的概率。对于离散分布是计算概率质量函数（Probability Mass Function）。

参数：

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 事件发生的概率。默认值：None。

返回：

Tensor，概率值。

sample (*shape, probs*)

采样函数。

参数：

- **shape** (tuple) - 样本的 shape。
- **probs** (Tensor) - 事件发生的概率。默认值：None。

返回：

Tensor，根据概率分布采样的样本。

sd (*probs*)

计算标准差。

参数：

- **probs** (Tensor) - 事件发生的概率。默认值：None。

返回：

Tensor，概率分布的标准差。

survival_function (*value, probs*)

计算给定值对应的生存函数。

参数：

- **value** (Tensor) - 要计算的值。

- **probs** (Tensor) - 事件发生的概率。默认值: `None`。

返回:

Tensor, 生存函数的值。

var (*probs*)

计算方差。

参数:

- **probs** (Tensor) - 事件发生的概率。默认值: `None`。

返回:

Tensor, 概率分布的方差。

22.2.4 mindspore.nn.probability.distribution.Cauchy

class mindspore.nn.probability.distribution.**Cauchy** (*loc=None, scale=None, seed=None, dtype=mstype.float32, name='Cauchy'*)

柯西分布 (Cauchy distribution)。连续随机分布, 取值范围为所有实数, 概率密度函数为:

$$f(x, a, b) = 1/\pi b (1 - ((x - a)/b)^2)$$

其中 a, b 分别为柯西分布的位置参数和比例参数。

参数:

- **loc** (int, float, list, numpy.ndarray, Tensor) - 柯西分布的位置, 公式中的 a 。默认值: `None`。
- **scale** (int, float, list, numpy.ndarray, Tensor) - 柯西分布的比例, 公式中的 b 。默认值: `None`。
- **seed** (int) - 采样时使用的种子。如果为 `None`, 则使用全局种子。默认值: `None`。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值: `mstype.float32`。
- **name** (str) - 分布的名称。默认值: `'Cauchy'`。

说明:

- *scale* 中的元素必须大于零。
- *dtype* 必须是 `float`, 因为柯西分布是连续的。
- GPU 后端不支持柯西分布。

异常:

- **ValueError** - *scale* 中元素小于或等于 0。
- **TypeError** - *dtype* 不是 `float` 或 `float` 的子类。

支持平台:

Ascend

样例:

```

>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Cauchy distribution of loc 3.0 and scale 4.0.
>>> cauchy1 = msd.Cauchy(3.0, 4.0, dtype=mindspore.float32)
>>> # A Cauchy distribution can be initialized without arguments.
>>> # In this case, 'loc' and 'scale' must be passed in through arguments.
>>> cauchy2 = msd.Cauchy(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1.0, 2.0, 3.0], dtype=mindspore.float32)
>>> loc_a = Tensor([2.0], dtype=mindspore.float32)
>>> scale_a = Tensor([2.0, 2.0, 2.0], dtype=mindspore.float32)
>>> loc_b = Tensor([1.0], dtype=mindspore.float32)
>>> scale_b = Tensor([1.0, 1.5, 2.0], dtype=mindspore.float32)
>>> # Private interfaces of probability functions corresponding to public interfaces,
    ↪including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`,
>>> # have the same arguments as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     loc (Tensor): the location of the distribution. Default: self.loc.
>>> #     scale (Tensor): the scale of the distribution. Default: self.scale.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing 'prob' by the name of the function
>>> ans = cauchy1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to distribution b.
>>> ans = cauchy1.prob(value, loc_b, scale_b)
>>> print(ans.shape)
(3,)
>>> # `loc` and `scale` must be passed in during function calls
>>> ans = cauchy2.prob(value, loc_a, scale_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mode` and `entropy` have the same arguments.
>>> # Args:
>>> #     loc (Tensor): the location of the distribution. Default: self.loc.
>>> #     scale (Tensor): the scale of the distribution. Default: self.scale.
>>> # Example of `mode`.
>>> ans = cauchy1.mode() # return 3.0
>>> print(ans.shape)
()
>>> ans = cauchy1.mode(loc_b, scale_b) # return loc_b
>>> print(ans.shape)
(3,)
>>> # `loc` and `scale` must be passed in during function calls.
>>> ans = cauchy2.mode(loc_a, scale_a)
>>> print(ans.shape)
(3,)
>>> # Interfaces of 'kl_loss' and 'cross_entropy' are the same:
>>> # Args:
>>> #     dist (str): the type of the distributions. Only "Cauchy" is supported.
>>> #     loc_b (Tensor): the loc of distribution b.
>>> #     scale_b (Tensor): the scale distribution b.

```

(下页继续)

(续上页)

```

>>> #     loc (Tensor): the loc of distribution a. Default: self.loc.
>>> #     scale (Tensor): the scale distribution a. Default: self.scale.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = cauchy1.kl_loss('Cauchy', loc_b, scale_b)
>>> print(ans.shape)
(3,)
>>> ans = cauchy1.kl_loss('Cauchy', loc_b, scale_b, loc_a, scale_a)
>>> print(ans.shape)
(3,)
>>> # Additional `loc` and `scale` must be passed in.
>>> ans = cauchy2.kl_loss('Cauchy', loc_b, scale_b, loc_a, scale_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.
>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ()
>>> #     loc (Tensor): the location of the distribution. Default: self.loc.
>>> #     scale (Tensor): the scale of the distribution. Default: self.scale.
>>> ans = cauchy1.sample()
>>> print(ans.shape)
()
>>> ans = cauchy1.sample((2,3))
>>> print(ans.shape)
(2, 3)
>>> ans = cauchy1.sample((2,3), loc_b, scale_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = cauchy2.sample((2,3), loc_a, scale_a)
>>> print(ans.shape)
(2, 3, 3)

```

property loc

返回分布位置。

返回:

Tensor, 分布位置的值。

property scale

返回分布比例。

返回:

Tensor, 分布比例的值。

cdf (value, loc, scale)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 累积分布函数的值。

cross_entropy (*dist, loc_b, scale_b, loc, scale*)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 分布 b 的位置参数。
- **scale_b** (Tensor) - 分布 b 的比例参数。
- **loc** (Tensor) - 分布 a 的位置参数。默认值：None。
- **scale** (Tensor) - 分布 a 的比例参数。默认值：None。

返回：

Tensor, 交叉熵的值。

entropy (*loc, scale*)

计算熵。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 熵的值。

kl_loss (*dist, loc_b, scale_b, loc, scale*)

计算 KL 散度, 即 KL(a||b)。

参数：

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 分布 b 的位置参数。
- **scale_b** (Tensor) - 分布 b 的比例参数。
- **loc** (Tensor) - 分布 a 的位置参数。默认值：None。
- **scale** (Tensor) - 分布 a 的比例参数。默认值：None。

返回：

Tensor, KL 散度。

log_cdf (*value, loc, scale*)

计算给定值的累积分布函数的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 累积分布函数的对数。

log_prob (*value, loc, scale*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 概率的对数。

log_survival (*value, loc, scale*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 生存函数的对数。

mean (*loc, scale*)

计算期望。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 概率分布的期望。

mode (*loc, scale*)

计算众数。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 概率分布的众数。

prob (*value, loc, scale*)

计算给定值的概率。对于连续分布是计算概率密度函数 (Probability Density Function)。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 概率值。

sample (*shape, loc, scale*)

计算采样函数。

参数：

- **shape** (tuple) - 样本的 shape。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，根据概率分布采样的样本。

sd (*loc, scale*)

计算标准差。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，概率分布的标准差。

survival_function (*value, loc, scale*)

计算给定值对应的生存函数。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，生存函数的值。

var (*loc, scale*)

计算方差。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，概率分布的方差。

22.2.5 mindspore.nn.probability.distribution.Distribution

class mindspore.nn.probability.distribution.**Distribution** (*seed, dtype, name, param*)
所有分布的基类。

参数：

- **seed** (int) - 采样时使用的种子。如果为 None，则使用 0。
- **dtype** (mindspore.dtype) - 事件样例的类型。
- **name** (str) - 分布的名称。
- **param** (dict) - 用于初始化分布的参数。

说明：派生类必须重写 `_mean`、`_prob` 和 `_log_prob` 等操作。必填参数必须通过 `args` 或 `kwargs` 传入，如 `_prob` 的 `value`。`dist_spec_args` 作为可选参数可以用来指定新的分布参数。

每种分类都有自己的 `dist_spec_args`。例如正态分布的 `dist_spec_args` 为 `mean` 和 `sd`，而指数分布的 `dist_spec_args` 为 `rate`。

所有方法都包含一个 `dist_spec_args` 作为可选参数。传入 `dist_spec_args` 可以让该方法基于新的分布的参数值进行运算，但如此做不会改变原始分布的参数。

支持平台：

Ascend GPU

cdf (*value, *args, **kwargs*)
计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数：

- **value** (Tensor) - 要计算的值。
- **args** (list) - 位置参数列表，具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典，具体需要的参数根据子类的实现确定。

说明：可以通过 `args` 或 `kwargs` 传递其 `dist_spec_args` 来选择性地将 `Distribution` 传递给函数。

返回：

Tensor，累积分布函数的值。

construct (*name, *args, **kwargs*)
重写 Cell 中的 `construct`。

说明：支持的函数包括：'prob'、'log_prob'、'cdf'、'log_cdf'、'survival_function'、'log_survival'、'var'、'sd'、'mode'、'mean'、'entropy'、'kl_loss'、'cross_entropy'、'sample'、'get_dist_args'、'get_dist_type'。

参数：

- **name** (str) - 函数名称。
- **args** (list) - 函数所需的位置参数列表。
- **kwargs** (dict) - 函数所需的关键字参数字典。

返回:

Tensor, name 对应函数的值。

cross_entropy (*dist*, **args*, ***kwargs*)

计算分布 a 和 b 之间的交叉熵。

参数:

- **dist** (str) - 分布的类型。
- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明: Distribution b 的 *dist_spec_args* 必须通过 *args* 或 *kwargs* 传递给函数。传入 Distribution a 的 *dist_spec_args* 是可选的。

返回:

Tensor, 交叉熵的值。

entropy (**args*, ***kwargs*)

计算熵。

参数:

- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明: 可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回:

Tensor, 熵的值。

get_dist_args (**args*, ***kwargs*)

返回分布的参数列表。

参数:

- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明: *dist_spec_args* 必须以列表或者字典的形式传入。传递给子类的参数的顺序应该与通过 *_add_parameter* 初始化默认参数的顺序相同。如果某个 *dist_spec_args* 为 None, 那么将返回默认值。

返回:

list[Tensor], 参数列表。

get_dist_type ()

返回分布类型。

返回:

string, 分布类型名称。

kl_loss (*dist*, *args, **kwargs)
计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明: Distribution b 的 *dist_spec_args* 必须通过 *args* 或 *kwargs* 传递给函数。传入 Distribution a 的 *dist_spec_args* 是可选的。

返回:

Tensor, KL 散度。

log_cdf (*value*, *args, **kwargs)
计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明: 可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value*, *args, **kwargs)
计算给定值对应的概率的对数 (pdf 或 pmf)。

参数:

- **value** (Tensor) - 要计算的值。
- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明: 可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回:

Tensor, 累积分布函数的对数。

log_survival (*value*, *args, **kwargs)
计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **args** (list) - 位置参数列表，具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典，具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回：

Tensor，生存函数的对数。

mean (*args, **kwargs)

计算期望。

参数：

- **args** (list) - 位置参数列表，具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典，具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回：

Tensor，概率分布的期望。

mode (*args, **kwargs)

计算众数。

参数：

- **args** (list) - 位置参数列表，具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典，具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回：

Tensor，概率分布的众数。

prob (value, *args, **kwargs)

计算给定值下的概率。对于离散分布是计算概率质量函数（Probability Mass Function），而对于连续分布是计算概率密度函数（Probability Density Function）。

参数：

- **value** (Tensor) - 要计算的值。
- **args** (list) - 位置参数列表，具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典，具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回：

Tensor, 概率值。

sample (*args, **kwargs)

采样函数。

参数：

- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回：

Tensor, 根据概率分布采样的样本。

sd (*args, **kwargs)

计算标准差。

参数：

- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回：

Tensor, 概率分布的标准差。

survival_function (value, *args, **kwargs)

计算给定值对应的生存函数。

参数：

- **value** (Tensor) - 要计算的值。
- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。
- **kwargs** (dict) - 关键字参数字典, 具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 Distribution 传递给函数。

返回：

Tensor, 生存函数的值。

var (*args, **kwargs)

计算方差。

参数：

- **args** (list) - 位置参数列表, 具体需要的参数根据子类的实现确定。

- **kwargs** (dict) - 关键字参数字典，具体需要的参数根据子类的实现确定。

说明：可以通过 *args* 或 *kwargs* 传递其 *dist_spec_args* 来选择性地将 *Distribution* 传递给函数。

返回：

Tensor，概率分布的方差。

22.2.6 mindspore.nn.probability.distribution.Exponential

class mindspore.nn.probability.distribution.**Exponential** (*rate=None, seed=None, dtype=mstype.float32, name='Exponential'*)

指数分布 (Exponential Distribution)。连续随机分布，取值范围为所有正实数 $[0, \text{inf})$ ，概率密度函数为

$$f(x, \lambda) = \lambda \exp(-\lambda x)$$

其中 λ 为指数分布的率参数。

参数：

- **rate** (int, float, list, numpy.ndarray, Tensor, 可选) - 率参数。公式中的 λ 。默认值：None。
- **seed** (int, 可选) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype, 可选) - 事件样例的类型。默认值：mstype.float32。
- **name** (str, 可选) - 分布的名称。默认值：'Exponential'。

说明：

- *rate* 中的元素必须大于 0。
 - *dtype* 必须是 float，因为指数分布是连续的。
-

异常：

- **ValueError** - *rate* 中元素小于 0。
- **TypeError** - *dtype* 不是 float 或 float 的子类。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Exponential distribution of the probability 0.5.
>>> e1 = msd.Exponential(0.5, dtype=mindspore.float32)
>>> # An Exponential distribution can be initialized without arguments.
>>> # In this case, `rate` must be passed in through `args` during function calls.
>>> e2 = msd.Exponential(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
```

(下页继续)

(续上页)

```

>>> value = Tensor([1, 2, 3], dtype=mindspore.float32)
>>> rate_a = Tensor([0.6], dtype=mindspore.float32)
>>> rate_b = Tensor([0.2, 0.5, 0.4], dtype=mindspore.float32)
>>> # Private interfaces of probability functions corresponding to public interfaces,
    ↳ including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`, are the
    ↳ same as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     rate (Tensor): the rate of the distribution. Default: self.rate.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing `prob` by the name of the function.
>>> ans = e1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to distribution b.
>>> ans = e1.prob(value, rate_b)
>>> print(ans.shape)
(3,)
>>> # `rate` must be passed in during function calls.
>>> ans = e2.prob(value, rate_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `sd`, `var`, and `entropy` have the same arguments as follows.
>>> # Args:
>>> #     rate (Tensor): the rate of the distribution. Default: self.rate.
>>> # Examples of `mean`. `sd`, `var`, and `entropy` are similar.
>>> ans = e1.mean() # return 2
>>> print(ans.shape)
()
>>> ans = e1.mean(rate_b) # return 1 / rate_b
>>> print(ans.shape)
(3,)
>>> # `rate` must be passed in during function calls.
>>> ans = e2.mean(rate_a)
>>> print(ans.shape)
(1,)
>>> # Interfaces of `kl_loss` and `cross_entropy` are the same.
>>> # Args:
>>> #     dist (str): The name of the distribution. Only 'Exponential' is supported.
>>> #     rate_b (Tensor): the rate of distribution b.
>>> #     rate_a (Tensor): the rate of distribution a. Default: self.rate.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = e1.kl_loss('Exponential', rate_b)
>>> print(ans.shape)
(3,)
>>> ans = e1.kl_loss('Exponential', rate_b, rate_a)
>>> print(ans.shape)
(3,)
>>> # An additional `rate` must be passed in.
>>> ans = e2.kl_loss('Exponential', rate_b, rate_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.
>>> # Args:

```

(下页继续)

(续上页)

```

>>> #      shape (tuple): the shape of the sample. Default: ()
>>> #      probs1 (Tensor): the rate of the distribution. Default: self.rate.
>>> ans = e1.sample()
>>> print(ans.shape)
()
>>> ans = e1.sample((2, 3))
>>> print(ans.shape)
(2, 3)
>>> ans = e1.sample((2, 3), rate_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = e2.sample((2, 3), rate_a)
>>> print(ans.shape)
(2, 3, 1)

```

property rate

返回指数分布的率参数 *rate*。

返回：

Tensor，率参数的值。

cdf (*value*, *rate=None*)

基于给定值计算累积分布函数 (Cumulative Distribution Function, CDF)。

参数：

- **value** (Tensor) - 待计算的值。
- **rate** (Tensor, 可选) - 分布的率参数。默认值：None。

返回：

Tensor，累积分布函数的值。

cross_entropy (*dist*, *rate_b*, *rate=None*)

计算分布 *a* 和 *b* 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。
- **rate_b** (Tensor) - 对比分布 *b* 的率参数。
- **rate** (Tensor, 可选) - 分布 *a* 的率参数。默认值：None。

返回：

Tensor，交叉熵的值。

entropy (*rate=None*)

计算熵。

参数：

- **rate** (Tensor, 可选) - 分布的率参数。默认值：None。

返回：

Tensor，熵的值。

kl_loss (*dist*, *rate_b*, *rate=None*)

计算分布 *a* 和分布 *b* 之间的 KL 散度，即 $KL(a||b)$ 。

参数：

- **dist** (str) - 分布的类型。
- **rate_b** (Tensor) - 对比分布 **b** 的率参数。
- **rate** (Tensor, 可选) - 分布 **a** 的率参数。默认值：None。

返回：

Tensor, KL 散度。

log_cdf (value, rate=None)

计算给定值对应的累积分布函数的对数。

参数：

- **value** (Tensor) - 待计算的值。
- **rate** (Tensor, 可选) - 分布的率参数。默认值：None。

返回：

Tensor, 累积分布函数的对数。

log_prob (value, rate=None)

计算给定值对应的概率的对数。

参数：

- **value** (Tensor) - 待计算的值。
- **rate** (Tensor, 可选) - 分布的率参数。默认值：None。

返回：

Tensor, 累积分布函数的对数。

log_survival (value, rate=None)

计算给定值对应的生存函数的对数。

参数：

- **value** (Tensor) - 待计算的值。
- **rate** (Tensor, 可选) - 分布的率参数。默认值：None。

返回：

Tensor, 生存函数的对数。

mean (rate=None)

计算期望。

参数：

- **rate** (Tensor, 可选) - 分布的率参数。默认值：None。

返回：

Tensor, 概率分布的期望。

mode (rate=None)

计算众数。

参数：

- **rate** (Tensor, 可选) - 分布的率参数。默认值：None。

返回:

Tensor, 概率分布的众数。

prob (*value*, *rate=None*)

计算给定值下的概率。对于连续随机分布是计算概率密度函数 (Probability Density Function)。

参数:

- **value** (Tensor) - 待计算的值。
- **rate** (Tensor, 可选) - 分布的率参数。默认值: None。

返回:

Tensor, 概率值。

sample (*shape*, *rate=None*)

计算采样函数。

参数:

- **shape** (tuple) - 期望得到的样本 shape。
- **rate** (Tensor, 可选) - 分布的率参数。默认值: None。

返回:

Tensor, 根据概率分布采样的样本。

sd (*rate=None*)

计算标准差。

参数:

- **rate** (Tensor, 可选) - 分布的率参数。默认值: None。

返回:

Tensor, 概率分布的标准差。

survival_function (*value*, *rate=None*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 待计算的值。
- **rate** (Tensor, 可选) - 分布的率参数。默认值: None。

返回:

Tensor, 生存函数的值。

var (*rate=None*)

计算方差。

参数:

- **rate** (Tensor, 可选) - 分布的率参数。默认值: None。

返回:

Tensor, 概率分布的方差。

22.2.7 mindspore.nn.probability.distribution.Gamma

class mindspore.nn.probability.distribution.**Gamma** (*concentration=None, rate=None, seed=None, dtype=mstype.float32, name='Gamma'*)

伽马分布 (Gamma distribution)。连续随机分布，取值范围为 (0, inf)，概率密度函数为：

$$f(x, \alpha, \beta) = \beta^\alpha / \Gamma(\alpha) x^{\alpha-1} \exp(-\beta x).$$

其中 G 为 Gamma 函数， α 和 β 为分别 Gamma 函数的浓度参数和逆尺度参数。

参数：

- **concentration** (int, float, list, numpy.ndarray, Tensor) - 浓度参数，也被称为伽马分布的 α 。默认值：None。
- **rate** (int, float, list, numpy.ndarray, Tensor) - 逆尺度参数，也被称为伽马分布的 β 。默认值：None。
- **seed** (int) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值：mstype.float32。
- **name** (str) - 分布的名称。默认值：'Gamma'。

说明：

- *concentration* 和 *rate* 中的元素必须大于零。
- *dtype* 必须是 float，因为伽马分布是连续的。

异常：

- **ValueError** - *concentration* 或者 *rate* 中元素小于或等于 0。
- **TypeError** - *dtype* 不是 float 或 float 的子类。

支持平台：

Ascend

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Gamma distribution of the concentration 3.0 and the rate 4.0.
>>> g1 = msd.Gamma([3.0], [4.0], dtype=mindspore.float32)
>>> # A Gamma distribution can be initialized without arguments.
>>> # In this case, `concentration` and `rate` must be passed in through arguments.
>>> g2 = msd.Gamma(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1.0, 2.0, 3.0], dtype=mindspore.float32)
>>> concentration_a = Tensor([2.0], dtype=mindspore.float32)
>>> rate_a = Tensor([2.0, 2.0, 2.0], dtype=mindspore.float32)
>>> concentration_b = Tensor([1.0], dtype=mindspore.float32)
>>> rate_b = Tensor([1.0, 1.5, 2.0], dtype=mindspore.float32)
>>>
>>> # Private interfaces of probability functions corresponding to public interfaces,
>>> # including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`,
```

(下页继续)

(续上页)

```

>>> # have the same arguments as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     concentration (Tensor): the concentration of the distribution. Default: self._
    ↪concentration.
>>> #     rate (Tensor): the rate of the distribution. Default: self._rate.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing 'prob' by the name of the function
>>> ans = g1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to the distribution b.
>>> ans = g1.prob(value, concentration_b, rate_b)
>>> print(ans.shape)
(3,)
>>> # `concentration` and `rate` must be passed in during function calls for g2.
>>> ans = g2.prob(value, concentration_a, rate_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `sd`, `mode`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     concentration (Tensor): the concentration of the distribution. Default: self._
    ↪concentration.
>>> #     rate (Tensor): the rate of the distribution. Default: self._rate.
>>> # Example of `mean`, `sd`, `mode`, `var`, and `entropy` are similar.
>>> ans = g1.mean()
>>> print(ans.shape)
(1,)
>>> ans = g1.mean(concentration_b, rate_b)
>>> print(ans.shape)
(3,)
>>> # `concentration` and `rate` must be passed in during function calls.
>>> ans = g2.mean(concentration_a, rate_a)
>>> print(ans.shape)
(3,)
>>> # Interfaces of 'kl_loss' and 'cross_entropy' are the same:
>>> # Args:
>>> #     dist (str): the type of the distributions. Only "Gamma" is supported.
>>> #     concentration_b (Tensor): the concentration of distribution b.
>>> #     rate_b (Tensor): the rate of distribution b.
>>> #     concentration_a (Tensor): the concentration of distribution a. Default: self._
    ↪concentration.
>>> #     rate_a (Tensor): the rate of distribution a. Default: self._rate.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = g1.kl_loss('Gamma', concentration_b, rate_b)
>>> print(ans.shape)
(3,)
>>> ans = g1.kl_loss('Gamma', concentration_b, rate_b, concentration_a, rate_a)
>>> print(ans.shape)
(3,)
>>> # Additional `concentration` and `rate` must be passed in.
>>> ans = g2.kl_loss('Gamma', concentration_b, rate_b, concentration_a, rate_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.

```

(下页继续)

(续上页)

```

>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ()
>>> #     concentration (Tensor): the concentration of the distribution. Default: self._
    ↪concentration.
>>> #     rate (Tensor): the rate of the distribution. Default: self._rate.
>>> ans = g1.sample()
>>> print(ans.shape)
(1,)
>>> ans = g1.sample((2,3))
>>> print(ans.shape)
(2, 3, 1)
>>> ans = g1.sample((2,3), concentration_b, rate_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = g2.sample((2,3), concentration_a, rate_a)
>>> print(ans.shape)
(2, 3, 3)

```

property concentration

返回分布的浓度（也称为伽马分布的 α ）。

返回：

Tensor，分布的浓度。

property rate

返回分布的逆尺度（也称为伽马分布的 β ）。

返回：

Tensor，分布的逆尺度。

cdf (value, concentration, rate)

计算给定值的累积分布函数（Cumulative Distribution Function, CDF）。

参数：

- **value** (Tensor) - 要计算的值。
- **concentration** (Tensor) - 分布的 α 。默认值：None。
- **rate** (Tensor) - 分布的 β 。默认值：None。

返回：

Tensor，累积分布函数的值。

cross_entropy (dist, concentration_b, rate_b, concentration, rate)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。
- **concentration_b** (Tensor) - 分布 b 的 α 。
- **rate_b** (Tensor) - 分布 b 的 β 。
- **concentration** (Tensor) - 分布 a 的 α 。默认值：None。
- **rate** (Tensor) - 分布 a 的 β 。默认值：None。

返回：

Tensor, 交叉熵的值。

entropy (*concentration, rate*)

计算熵。

参数:

- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 熵的值。

kl_loss (*dist, concentration_b, rate_b, concentration, rate*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **concentration_b** (Tensor) - 分布 b 的 α 。
- **rate_b** (Tensor) - 分布 b 的 β 。
- **concentration** (Tensor) - 分布 a 的 α 。默认值: None。
- **rate** (Tensor) - 分布 a 的 β 。默认值: None。

返回:

Tensor, KL 散度。

log_cdf (*value, concentration, rate*)

计算给定值的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value, concentration, rate*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 概率的对数。

log_survival (*value, concentration, rate*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 生存函数的对数。

mean (*concentration, rate*)

计算期望。

参数:

- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 概率分布的期望。

mode (*concentration, rate*)

计算众数。

参数:

- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 概率分布的众数。

prob (*value, concentration, rate*)

计算给定值下的概率。对于连续分布是计算概率密度函数 (Probability Density Function)。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 概率值。

sample (*shape, concentration, rate*)

采样函数。

参数:

- **shape** (tuple) - 样本的 shape。
- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 根据概率分布采样的样本。

sd (*concentration, rate*)

计算标准差。

参数:

- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 概率分布的标准差。

survival_function (*value, concentration, rate*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 生存函数的值。

var (*concentration, rate*)

计算方差。

参数:

- **concentration** (Tensor) - 分布的 α 。默认值: None。
- **rate** (Tensor) - 分布的 β 。默认值: None。

返回:

Tensor, 概率分布的方差。

22.2.8 mindspore.nn.probability.distribution.Geometric

class mindspore.nn.probability.distribution.**Geometric** (*probs=None, seed=None, dtype=mstype.int32, name='Geometric'*)

几何分布 (Geometric Distribution)。

它代表在第一次成功之前有 k 次失败, 即在第一次成功实现时, 总共有 $k+1$ 个伯努利试验。离散随机分布, 取值范围为正自然数集, 概率质量函数为 $P(X = i) = p(1 - p)^{i-1}, i = 1, 2, \dots$ 。

参数:

- **probs** (float, list, numpy.ndarray, Tensor) - 成功的概率。默认值: None。
- **seed** (int) - 采样时使用的种子。如果为 None, 则使用全局种子。默认值: None。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值: `mstype.int32`。
- **name** (str) - 分布的名称。默认值: 'Geometric'。

说明: *probs* 必须是合适的概率 ($0 < p < 1$)。 *dist_spec_args* 是 *probs*。

异常:

- **ValueError** - *probs* 中元素小于等于 0 或者大于等于 1。

支持平台:

Ascend GPU

样例:

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Geometric distribution of the probability 0.5.
>>> g1 = msd.Geometric(0.5, dtype=mindspore.int32)
>>> # A Geometric distribution can be initialized without arguments.
>>> # In this case, `probs` must be passed in through arguments during function calls.
>>> g2 = msd.Geometric(dtype=mindspore.int32)
>>>
>>> # Here are some tensors used below for testing
>>> value = Tensor([1, 0, 1], dtype=mindspore.int32)
>>> probs_a = Tensor([0.6], dtype=mindspore.float32)
>>> probs_b = Tensor([0.2, 0.5, 0.4], dtype=mindspore.float32)
>>>
>>> # Private interfaces of probability functions corresponding to public interfaces,
>>> ↪including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`,
>>> # have the same arguments as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     probs1 (Tensor): the probability of success of a Bernoulli trial. Default: self.
>>> ↪probs.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing `prob` by the name of the function.
>>> ans = g1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to distribution b.
>>> ans = g1.prob(value, probs_b)
>>> print(ans.shape)
(3,)
>>> # `probs` must be passed in during function calls.
>>> ans = g2.prob(value, probs_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `sd`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     probs1 (Tensor): the probability of success of a Bernoulli trial. Default: self.
>>> ↪probs.
>>> # Examples of `mean`. `sd`, `var`, and `entropy` are similar.
>>> ans = g1.mean() # return 1.0
>>> print(ans.shape)
()
>>> ans = g1.mean(probs_b)
>>> print(ans.shape)
(3,)
>>> # Probs must be passed in during function calls
>>> ans = g2.mean(probs_a)
>>> print(ans.shape)
(1,)
>>> # Interfaces of 'kl_loss' and 'cross_entropy' are the same.
>>> # Args:
```

(下页继续)

(续上页)

```

>>> # dist (str): the name of the distribution. Only 'Geometric' is supported.
>>> # probs1_b (Tensor): the probability of success of a Bernoulli trial of distribution.
↪ b.
>>> # probs1_a (Tensor): the probability of success of a Bernoulli trial of distribution.
↪ a.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = g1.kl_loss('Geometric', probs_b)
>>> print(ans.shape)
(3,)
>>> ans = g1.kl_loss('Geometric', probs_b, probs_a)
>>> print(ans.shape)
(3,)
>>> # An additional `probs` must be passed in.
>>> ans = g2.kl_loss('Geometric', probs_b, probs_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.
>>> # Args:
>>> # shape (tuple): the shape of the sample. Default: ()
>>> # probs1 (Tensor): the probability of success of a Bernoulli trial. Default: self.
↪ probs.
>>> ans = g1.sample()
>>> print(ans.shape)
()
>>> ans = g1.sample((2, 3))
>>> print(ans.shape)
(2, 3)
>>> ans = g1.sample((2, 3), probs_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = g2.sample((2, 3), probs_a)
>>> print(ans.shape)
(2, 3, 1)

```

property probs

返回伯努利试验成功的概率。

返回：

Tensor，伯努利试验成功的概率值。

cdf (value, probs)

计算给定值的累积分布函数（Cumulative Distribution Function, CDF）。

参数：

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor，累积分布函数的值。

cross_entropy (dist, probs_b, probs)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。

- **probs_b** (Tensor) - 分布 **b** 的伯努利实验成功的概率。
- **probs** (Tensor) - 分布 **a** 的伯努利实验成功的概率。默认值: `None`。

返回:

Tensor, 交叉熵的值。

entropy (*probs*)

计算熵。

参数:

- **probs** (Tensor) - 伯努利实验成功的概率。默认值: `None`。

返回:

Tensor, 熵的值。

kl_loss (*dist, probs_b, probs*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **probs_b** (Tensor) - 分布 **b** 的伯努利实验成功的概率。
- **probs** (Tensor) - 分布 **a** 的伯努利实验成功的概率。默认值: `None`。

返回:

Tensor, KL 散度。

log_cdf (*value, probs*)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 伯努利实验成功的概率。默认值: `None`。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value, probs*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 伯努利实验成功的概率。默认值: `None`。

返回:

Tensor, 概率的对数。

log_survival (*value, probs*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 伯努利实验成功的概率。默认值: `None`。

返回：

Tensor，生存函数的对数。

mean (*probs*)

计算期望。

参数：

- **probs** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor，概率分布的期望。

mode (*probs*)

计算众数。

参数：

- **probs** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor，概率分布的众数。

prob (*value, probs*)

计算给定值的概率。对于离散分布是计算概率质量函数（Probability Mass Function）。

参数：

- **value** (Tensor) - 要计算的值。
- **probs** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor，概率值。

sample (*shape, probs*)

采样函数。

参数：

- **shape** (tuple) - 样本的 shape。
- **probs** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor，根据概率分布采样的样本。

sd (*probs*)

计算标准差。

参数：

- **probs** (Tensor) - 伯努利实验成功的概率。默认值：None。

返回：

Tensor，概率分布的标准差。

survival_function (*value, probs*)

计算给定值对应的生存函数。

参数：

- **value** (Tensor) - 要计算的值。

- **probs** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 生存函数的值。

var (*probs*)

计算方差。

参数:

- **probs** (Tensor) - 伯努利实验成功的概率。默认值: None。

返回:

Tensor, 概率分布的方差。

22.2.9 mindspore.nn.probability.distribution.Gumbel

class mindspore.nn.probability.distribution.**Gumbel** (*loc, scale, seed=0, dtype=mstype.float32, name='Gumbel'*)

Gumbel 分布 (Gumbel distribution)。连续随机分布, 取值范围为所有实数, 概率密度函数为:

$$f(x, a, b) = 1/b \exp(\exp(-(x - a)/b) - x)$$

其中 a, b 为分别为 Gumbel 分布的位置参数和比例参数。

参数:

- **loc** (int, float, list, numpy.ndarray, Tensor) - Gumbel 分布的位置, 公式中的 a 。
- **scale** (int, float, list, numpy.ndarray, Tensor) - Gumbel 分布的比例, 公式中的 b 。
- **seed** (int) - 采样时使用的种子。如果为 None, 则使用全局种子。默认值: 0。
- **dtype** (mindspore.dtype) - 分布类型。默认值: `mstype.float32`。
- **name** (str) - 分布的名称。默认值: 'Gumbel'。

说明:

- *scale* 必须大于零。
- *dtype* 必须是 float, 因为 Gumbel 分布是连续的。
- GPU 后端不支持 *kl_loss* 和 *cross_entropy*。

异常:

- **ValueError** - *scale* 中元素小于等于 0。
- **TypeError** - *dtype* 不是 float 或 float 的子类。

支持平台:

Ascend GPU

样例:

```

>>> import mindspore
>>> import numpy as np
>>> import mindspore.nn.probability.distribution as msd
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> class Prob(nn.Cell):
...     def __init__(self):
...         super(Prob, self).__init__()
...         self.gum = msd.Gumbel(np.array([0.0]), np.array([[1.0], [2.0]]), dtype=mindspore.
↪float32)
...
...     def construct(self, x_):
...         return self.gum.prob(x_)
>>> value = np.array([1.0, 2.0]).astype(np.float32)
>>> pdf = Prob()
>>> output = pdf(Tensor(value, dtype=mindspore.float32))

```

property loc

返回分布位置。

返回：

Tensor，分布的位置值。

property scale

返回分布比例。

返回：

Tensor，分布的比例值。

cdf (*value, loc, scale*)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，累积分布函数的值。

cross_entropy (*dist, loc_b, scale_b, loc, scale*)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 分布 b 的位置参数。
- **scale_b** (Tensor) - 分布 b 的比例参数。
- **loc** (Tensor) - 分布 a 的位置参数。默认值：None。
- **scale** (Tensor) - 分布 a 的比例参数。默认值：None。

返回：

Tensor，交叉熵的值。

entropy (*loc, scale*)

计算熵。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 熵的值。

kl_loss (*dist, loc_b, scale_b, loc, scale*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数：

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 分布 b 的位置参数。
- **scale_b** (Tensor) - 分布 b 的比例参数。
- **loc** (Tensor) - 分布 a 的位置参数。默认值：None。
- **scale** (Tensor) - 分布 a 的比例参数。默认值：None。

返回：

Tensor, KL 散度。

log_cdf (*value, loc, scale*)

计算给定值对应的累积分布函数的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 累积分布函数的对数。

log_prob (*value, loc, scale*)

计算给定值对应的概率的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 概率的对数。

log_survival (*value, loc, scale*)

计算给定值对应的生存函数的对数。

参数：

- **value** (Tensor) - 要计算的值。

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 生存函数的对数。

mean (*loc, scale*)

计算期望。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率分布的期望。

mode (*loc, scale*)

计算众数。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率分布的众数。

prob (*value, loc, scale*)

计算给定值的概率。对于连续分布是计算概率密度函数 (Probability Density Function)。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率值。

sample (*shape, loc, scale*)

采样函数。

参数:

- **shape** (tuple) - 样本的 shape。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 根据概率分布采样的样本。

sd (*loc, scale*)

计算标准差。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 概率分布的标准差。

survival_function (*value, loc, scale*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 生存函数的值。

var (*loc, scale*)

计算方差。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 概率分布的方差。

22.2.10 mindspore.nn.probability.distribution.HalfNormal

class mindspore.nn.probability.distribution.**HalfNormal** (*mean=None, sd=None, seed=None, dtype=mstype.float32, name='HalfNormal'*)

半正态分布 (HalfNormal distribution)。连续随机分布, 取值范围为 $[\mu, \text{inf})$, 概率密度函数为:

$$f(x, \mu, \sigma) = 1/\sigma\sqrt{2\pi} \exp(-(x - \mu)^2/2\sigma^2).$$

其中 μ, σ 为分别为半正态分布的期望与标准差。

参数:

- **mean** (Union[int, float, list, numpy.ndarray, Tensor], 可选) - 半正态分布的期望, 公式中的 μ 。如果输入为 `None`, 那么分布的期望将在运行时传入。默认值: `None`。
- **sd** (Union[int, float, list, numpy.ndarray, Tensor], 可选) - 半正态分布的标准差, 公式中的 σ 。如果输入为 `None`, 那么分布的标准差将在运行时传入。默认值: `None`。
- **seed** (int, 可选) - 采样时使用的种子。如果为 `None`, 则使用全局种子。默认值: `None`。
- **dtype** (mindspore.dtype, 可选) - 事件样例的类型。默认值: `mstype.float32`。
- **name** (str, 可选) - 分布的名称。默认值: `'HalfNormal'`。

说明:

- *sd* 必须大于 0。

- `dtype` 必须是 `float`，因为半正态分布是连续的。
- 如果在方法函数调用中传入参数 `mean` 或者 `sd`，则计算中会使用传参值，否则就会使用初始化时的参数值。

异常：

- **ValueError** - `sd` 中元素不大于 0。
- **TypeError** - `dtype` 不是 `float` 或 `float` 的子类。

支持平台：

CPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore.nn.probability.distribution import HalfNormal
>>> from mindspore import Tensor
>>> # To initialize a HalfNormal distribution of the mean 3.0 and the standard deviation 4.0.
>>> n1 = HalfNormal(3.0, 4.0, dtype=mindspore.float32)
>>> # A HalfNormal distribution can be initialized without arguments.
>>> # In this case, `mean` and `sd` must be passed in through arguments.
>>> hn = HalfNormal(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1.0, 2.0, 3.0], dtype=mindspore.float32)
>>> mean_a = Tensor([2.0], dtype=mindspore.float32)
>>> sd_a = Tensor([2.0, 2.0, 2.0], dtype=mindspore.float32)
>>> mean_b = Tensor([1.0], dtype=mindspore.float32)
>>> sd_b = Tensor([1.0, 1.5, 2.5], dtype=mindspore.float32)
>>> ans = n1.log_prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to the distribution b.
>>> ans = n1.log_prob(value, mean_b, sd_b)
>>> print(ans.shape)
(3,)
>>> # `mean` and `sd` must be passed in during function calls
>>> ans = hn.log_prob(value, mean_a, sd_a)
>>> print(ans.shape)
(3,)
```

log_prob (*value*, *mean=None*, *sd=None*)

计算半正态分布给定值对应的概率的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor, 可选) - 分布的期望。默认值：None。
- **sd** (Tensor, 可选) - 分布的标准差。默认值：None。

返回：

Tensor，概率密度函数的对数。

22.2.11 mindspore.nn.probability.distribution.Laplace

class mindspore.nn.probability.distribution.**Laplace** (*mean=None, sd=None, seed=None, dtype=mstype.float32, name='Laplace'*)

拉普拉斯分布 (Laplace distribution)。连续随机分布，取值范围为 $(-\infty, \infty)$ ，概率密度函数为

$$f(x, \mu, b) = 1/(2 * b) * \exp(-abs(x - \mu)/b).$$

其中 μ, b 分别为拉普拉斯分布的期望与扩散度。

参数：

- **mean** (Union[int, float, list, numpy.ndarray, Tensor], 可选) - 拉普拉斯分布的平均值。如果输入为 None，那么分布的平均值将在运行时传入。如果设置为默认值：None。
- **sd** (Union[int, float, list, numpy.ndarray, Tensor], 可选) - 拉普拉斯分布的扩散度。如果输入为 None，那么分布的扩散度将在运行时传入。默认值：None。
- **seed** (int, 可选) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype, 可选) - 事件样例的类型。默认值：mstype.float32。
- **name** (str, 可选) - 分布的名称。默认值：'Laplace'。

说明：

- *sd* 必须大于 0。
- *dtype* 必须是 float，因为拉普拉斯分布是连续的。
- 如果在方法函数调用中传入参数 *mean* 或者 *sd*，则计算中会使用其传参值，否则就会使用初始化时的参数值。

异常：

- **ValueError** - *sd* 中元素不大于 0。
- **TypeError** - *dtype* 不是 float 的子类。

支持平台：

Ascend GPU CPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> from mindspore.nn.probability.distribution import Laplace
>>> from mindspore import Tensor
>>> # To initialize a Laplace distribution of the mean 3.0 and the scale 4.0.
>>> n1 = Laplace(3.0, 4.0, dtype=mindspore.float32)
>>> # A Laplace distribution can be initialized without arguments.
>>> # In this case, `mean` and `sd` must be passed in through arguments.
>>> n2 = Laplace(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1.0, 2.0, 3.0], dtype=mindspore.float32)
>>> mean_a = Tensor([2.0], dtype=mindspore.float32)
>>> sd_a = Tensor([2.0, 2.0, 2.0], dtype=mindspore.float32)
>>> mean_b = Tensor([1.0], dtype=mindspore.float32)
>>> sd_b = Tensor([1.0, 1.5, 2.0], dtype=mindspore.float32)
```

(下页继续)

(续上页)

```
>>> ans = n1.log_prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to the distribution b.
>>> ans = n1.log_prob(value, mean_b, sd_b)
>>> print(ans.shape)
(3,)
>>> # `mean` and `sd` must be passed in during function calls
>>> ans = n2.log_prob(value, mean_a, sd_a)
>>> print(ans.shape)
(3,)
```

log_prob (value, mean=None, sd=None)

计算拉普拉斯分布给定值对应的概率的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor, 可选) - 分布的期望。默认值：None。
- **sd** (Tensor, 可选) - 分布的扩散度。默认值：None。

返回：

Tensor, 概率密度函数的对数。

22.2.12 mindspore.nn.probability.distribution.Logistic

class mindspore.nn.probability.distribution.**Logistic** (loc=None, scale=None, seed=None, dtype=mstype.float32, name='Logistic')

Logistic 分布 (Logistic distribution)。连续随机分布，取值范围为 $(-\infty, \infty)$ ，概率密度函数为：

$$f(x, a, b) = 1/b \exp(\exp(-(x - a)/b) - x).$$

其中 a, b 为分别为 Logistic 分布的位置参数和比例参数。

参数：

- **loc** (float, list, numpy.ndarray, Tensor) - Logistic 分布的位置，公式中的 a 。默认值：None。
- **scale** (float, list, numpy.ndarray, Tensor) - Logistic 分布的比例，公式中的 b 。默认值：None。
- **seed** (int) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值：mstype.float32。
- **name** (str) - 分布的名称。默认值：'Logistic'。

说明：

- *scale* 必须大于零。
- *dtype* 必须是 float，因为 Logistic 分布是连续的。

异常：

- **ValueError** - *scale* 中元素小于或等于 0。

- **TypeError** - `dtype` 不是 float 或 float 的子类。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Logistic distribution of loc 3.0 and scale 4.0.
>>> l1 = msd.Logistic(3.0, 4.0, dtype=mindspore.float32)
>>> # A Logistic distribution can be initialized without arguments.
>>> # In this case, `loc` and `scale` must be passed in through arguments.
>>> l2 = msd.Logistic(dtype=mindspore.float32)
>>>
>>> # Here are some tensors used below for testing
>>> value = Tensor([1.0, 2.0, 3.0], dtype=mindspore.float32)
>>> loc_a = Tensor([2.0], dtype=mindspore.float32)
>>> scale_a = Tensor([2.0, 2.0, 2.0], dtype=mindspore.float32)
>>> loc_b = Tensor([1.0], dtype=mindspore.float32)
>>> scale_b = Tensor([1.0, 1.5, 2.0], dtype=mindspore.float32)
>>>
>>> # Private interfaces of probability functions corresponding to public interfaces,
>>> # including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`,
>>> # have the same arguments as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     loc (Tensor): the location of the distribution. Default: self.loc.
>>> #     scale (Tensor): the scale of the distribution. Default: self.scale.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing 'prob' by the name of the function
>>> ans = l1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to distribution b.
>>> ans = l1.prob(value, loc_b, scale_b)
>>> print(ans.shape)
(3,)
>>> # `loc` and `scale` must be passed in during function calls
>>> ans = l1.prob(value, loc_a, scale_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `mode`, `sd`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     loc (Tensor): the location of the distribution. Default: self.loc.
>>> #     scale (Tensor): the scale of the distribution. Default: self.scale.
>>> # Example of `mean`. `mode`, `sd`, `var`, and `entropy` are similar.
>>> ans = l1.mean()
>>> print(ans.shape)
()
>>> ans = l1.mean(loc_b, scale_b)
>>> print(ans.shape)
(3,)
```

(下页继续)

(续上页)

```

>>> # `loc` and `scale` must be passed in during function calls.
>>> ans = l1.mean(loc_a, scale_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.
>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ()
>>> #     loc (Tensor): the location of the distribution. Default: self.loc.
>>> #     scale (Tensor): the scale of the distribution. Default: self.scale.
>>> ans = l1.sample()
>>> print(ans.shape)
()
>>> ans = l1.sample((2, 3))
>>> print(ans.shape)
(2, 3)
>>> ans = l1.sample((2, 3), loc_b, scale_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = l1.sample((2, 3), loc_a, scale_a)
>>> print(ans.shape)
(2, 3, 3)

```

property loc

返回分布位置。

返回：

Tensor，分布的位置值。

property scale

返回分布比例。

返回：

Tensor，分布的比例值。

cdf (value, loc, scale)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，累积分布函数的值。

cross_entropy (dist, loc_b, scale_b, loc, scale)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 分布 b 的位置参数。
- **scale_b** (Tensor) - 分布 b 的比例参数。
- **loc** (Tensor) - 分布 a 的位置参数。默认值：None。

- **scale** (Tensor) - 分布 a 的比例参数。默认值: None。

返回:

Tensor, 交叉熵的值。

entropy (*loc, scale*)

计算熵。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 熵的值。

kl_loss (*dist, loc_b, scale_b, loc, scale*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 分布 b 的位置参数。
- **scale_b** (Tensor) - 分布 b 的比例参数。
- **loc** (Tensor) - 分布 a 的位置参数。默认值: None。
- **scale** (Tensor) - 分布 a 的比例参数。默认值: None。

返回:

Tensor, KL 散度。

log_cdf (*value, loc, scale*)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value, loc, scale*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率的对数。

log_survival (*value, loc, scale*)

计算给定值对应的生存函数的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，生存函数的对数。

mean (*loc, scale*)

计算期望。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，概率分布的期望。

mode (*loc, scale*)

计算众数。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，概率分布的众数。

prob (*value, loc, scale*)

计算给定值的概率。对于连续分布是计算概率密度函数（Probability Density Function）。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor，概率值。

sample (*shape, loc, scale*)

采样函数。

参数：

- **shape** (tuple) - 样本的 shape。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 根据概率分布采样的样本。

sd (*loc, scale*)

计算标准差。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率分布的标准差。

survival_function (*value, loc, scale*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 生存函数的值。

var (*loc, scale*)

计算方差。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率分布的方差。

22.2.13 mindspore.nn.probability.distribution.LogNormal

class mindspore.nn.probability.distribution.**LogNormal** (*loc=None, scale=None, seed=0, dtype=mstype.float32, name='LogNormal'*)

对数正态分布 (LogNormal distribution)。连续随机分布, 取值范围为 (0, inf), 概率密度函数为

$$f(x, \mu, \sigma) = 1/x\sigma\sqrt{2\pi} \exp(-(\ln(x) - \mu)^2/2\sigma^2).$$

其中 μ, σ 为分别为基础正态分布的平均值和标准差。服从对数正态分布的随机变量的对数服从正态分布。它被构造为正态分布的指数变换。

参数:

- **loc** (int, float, list, numpy.ndarray, Tensor) - 基础正态分布的平均值。默认值: None。
- **scale** (int, float, list, numpy.ndarray, Tensor) - 基础正态分布的标准差。默认值: None。
- **seed** (int) - 采样时使用的种子。如果为 None, 则使用全局种子。默认值: 0。
- **dtype** (mindspore.dtype) - 分布类型。默认值: mstype.float32。
- **name** (str) - 分布的名称。默认值: 'LogNormal'。

说明:

- *scale* 必须大于零。
- *dtype* 必须是 `float`, 因为对数正态分布是连续的。

异常:

- **ValueError** - *scale* 中元素小于 0。
- **TypeError** - *dtype* 不是 `float` 的子类。

支持平台:

Ascend GPU

样例:

```
>>> import numpy as np
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> class Prob(nn.Cell):
...     def __init__(self):
...         super(Prob, self).__init__()
...         self.ln = msd.LogNormal(np.array([0.3]), np.array([[0.2], [0.4]]),
... dtype=mindspore.float32)
...     def construct(self, x_):
...         return self.ln.prob(x_)
>>> pdf = Prob()
>>> output = pdf(Tensor([1.0, 2.0], dtype=mindspore.float32))
>>> print(output.shape)
(2, 2)
```

property loc

返回分布位置。

返回:

Tensor, 分布的位置值。

property scale

返回分布比例。

返回:

Tensor, 分布的比例值。

cdf (*value, loc, scale*)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: `None`。
- **scale** (Tensor) - 分布比例参数。默认值: `None`。

返回:

Tensor, 累积分布函数的值。

cross_entropy (*dist, loc_b, scale_b, loc, scale*)

计算分布 a 和 b 之间的交叉熵。

参数:

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 对比分布位置参数。
- **scale_b** (Tensor) - 对比分布比例参数。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 交叉熵的值。

entropy (*loc, scale*)

计算熵。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 熵的值。

kl_loss (*dist, loc_b, scale_b, loc, scale*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **loc_b** (Tensor) - 对比分布位置参数。
- **scale_b** (Tensor) - 对比分布比例参数。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, KL 散度。

log_cdf (*value, loc, scale*)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value, loc, scale*)

计算给定值对应的概率的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 累积分布函数的对数。

log_survival (*value, loc, scale*)

计算给定值对应的生存函数的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 生存函数的对数。

mean (*loc, scale*)

计算期望。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 概率分布的期望。

mode (*loc, scale*)

计算众数。

参数：

- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 概率分布的众数。

prob (*value, loc, scale*)

计算给定值下的概率。对于连续分布是计算概率密度函数 (Probability Density Function)。

参数：

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值：None。
- **scale** (Tensor) - 分布比例参数。默认值：None。

返回：

Tensor, 概率值。

sample (*shape, loc, scale*)

采样函数。

参数:

- **shape** (tuple) - 样本的 shape。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 根据概率分布采样的样本。

sd (*loc, scale*)

计算标准差。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率分布的标准差。

survival_function (*value, loc, scale*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 生存函数的值。

var (*loc, scale*)

计算方差。

参数:

- **loc** (Tensor) - 分布位置参数。默认值: None。
- **scale** (Tensor) - 分布比例参数。默认值: None。

返回:

Tensor, 概率分布的方差。

22.2.14 mindspore.nn.probability.distribution.Normal

class mindspore.nn.probability.distribution.**Normal** (*mean=None, sd=None, seed=None, dtype=mstype.float32, name='Normal'*)

正态分布 (Normal distribution)。连续随机分布，取值范围为 $(-\infty, \infty)$ ，概率密度函数为

$$f(x, \mu, \sigma) = 1/\sigma\sqrt{2\pi} \exp(-(x - \mu)^2/2\sigma^2).$$

其中 μ, σ 分别为正态分布的期望与标准差。

参数：

- **mean** (int, float, list, numpy.ndarray, Tensor) - 正态分布的平均值。默认值：None。
- **sd** (int, float, list, numpy.ndarray, Tensor) - 正态分布的标准差。默认值：None。
- **seed** (int) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值：mstype.float32。
- **name** (str) - 分布的名称。默认值：'Normal'。

说明：

- *sd* 必须大于零。
- *dtype* 必须是 float，因为正态分布是连续的。

异常：

- **ValueError** - *sd* 中元素小于 0。
- **TypeError** - *dtype* 不是 float 的子类。

支持平台：

Ascend GPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Normal distribution of the mean 3.0 and the standard deviation 4.0.
>>> n1 = msd.Normal(3.0, 4.0, dtype=mindspore.float32)
>>> # A Normal distribution can be initialized without arguments.
>>> # In this case, `mean` and `sd` must be passed in through arguments.
>>> n2 = msd.Normal(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1.0, 2.0, 3.0], dtype=mindspore.float32)
>>> mean_a = Tensor([2.0], dtype=mindspore.float32)
>>> sd_a = Tensor([2.0, 2.0], dtype=mindspore.float32)
>>> mean_b = Tensor([1.0], dtype=mindspore.float32)
>>> sd_b = Tensor([1.0, 1.5, 2.0], dtype=mindspore.float32)
>>> # Private interfaces of probability functions corresponding to public interfaces,
>>> # including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`,
>>> # have the same arguments as follows.
```

(下页继续)

(续上页)

```

>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     mean (Tensor): the mean of the distribution. Default: self._mean_value.
>>> #     sd (Tensor): the standard deviation of the distribution. Default: self._sd_value.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing 'prob' by the name of the function
>>> ans = n1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to the distribution b.
>>> ans = n1.prob(value, mean_b, sd_b)
>>> print(ans.shape)
(3,)
>>> # `mean` and `sd` must be passed in during function calls
>>> ans = n2.prob(value, mean_a, sd_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `sd`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     mean (Tensor): the mean of the distribution. Default: self._mean_value.
>>> #     sd (Tensor): the standard deviation of the distribution. Default: self._sd_value.
>>> # Example of `mean`. `sd`, `var`, and `entropy` are similar.
>>> ans = n1.mean() # return 0.0
>>> print(ans.shape)
()
>>> ans = n1.mean(mean_b, sd_b) # return mean_b
>>> print(ans.shape)
(3,)
>>> # `mean` and `sd` must be passed in during function calls.
>>> ans = n2.mean(mean_a, sd_a)
>>> print(ans.shape)
(3,)
>>> # Interfaces of 'kl_loss' and 'cross_entropy' are the same:
>>> # Args:
>>> #     dist (str): the type of the distributions. Only "Normal" is supported.
>>> #     mean_b (Tensor): the mean of distribution b.
>>> #     sd_b (Tensor): the standard deviation of distribution b.
>>> #     mean_a (Tensor): the mean of distribution a. Default: self._mean_value.
>>> #     sd_a (Tensor): the standard deviation of distribution a. Default: self._sd_value.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = n1.kl_loss('Normal', mean_b, sd_b)
>>> print(ans.shape)
(3,)
>>> ans = n1.kl_loss('Normal', mean_b, sd_b, mean_a, sd_a)
>>> print(ans.shape)
(3,)
>>> # Additional `mean` and `sd` must be passed in.
>>> ans = n2.kl_loss('Normal', mean_b, sd_b, mean_a, sd_a)
>>> print(ans.shape)
(3,)
>>> # Examples of `sample`.
>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ()
>>> #     mean (Tensor): the mean of the distribution. Default: self._mean_value.
>>> #     sd (Tensor): the standard deviation of the distribution. Default: self._sd_value.

```

(下页继续)

(续上页)

```

>>> ans = n1.sample()
>>> print(ans.shape)
()
>>> ans = n1.sample((2, 3))
>>> print(ans.shape)
(2, 3)
>>> ans = n1.sample((2, 3), mean_b, sd_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = n2.sample((2, 3), mean_a, sd_a)
>>> print(ans.shape)
(2, 3, 3)

```

property mean

返回分布期望。

返回：

Tensor, 分布的期望。

property sd

返回分布的标准差。

返回：

Tensor, 分布的标准差。

cdf (*value, mean, sd*)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数：

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor) - 分布的期望。默认值：None。
- **sd** (Tensor) - 分布的标准差。默认值：None。

返回：

Tensor, 累积分布函数的值。

cross_entropy (*dist, mean_b, sd_b, mean, sd*)

计算分布 a 和 b 之间的交叉熵。

参数：

- **dist** (str) - 分布的类型。
- **mean_b** (Tensor) - 对比分布的期望。
- **sd_b** (Tensor) - 对比分布的标准差。
- **mean** (Tensor) - 分布的期望。默认值：None。
- **sd** (Tensor) - 分布的标准差。默认值：None。

返回：

Tensor, 交叉熵的值。

entropy (*mean, sd*)

计算熵。

参数：

- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 熵的值。

kl_loss (*dist, mean_b, sd_b, mean, sd*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **mean_b** (Tensor) - 对比分布的期望。
- **sd_b** (Tensor) - 对比分布的标准差。
- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, KL 散度。

log_cdf (*value, mean, sd*)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value, mean, sd*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 累积分布函数的对数。

log_survival (*value, mean, sd*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 生存函数的对数。

mode (*mean, sd*)

计算众数。

参数:

- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 概率分布的众数。

prob (*value, mean, sd*)

计算给定值下的概率。对于连续分布是计算概率密度函数 (Probability Density Function)。

参数:

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 概率值。

sample (*shape, mean, sd*)

采样函数。

参数:

- **shape** (tuple) - 样本的 shape。
- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 根据概率分布采样的样本。

survival_function (*value, mean, sd*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 生存函数的值。

var (*mean, sd*)

计算方差。

参数:

- **mean** (Tensor) - 分布的期望。默认值: None。
- **sd** (Tensor) - 分布的标准差。默认值: None。

返回:

Tensor, 概率分布的方差。

22.2.15 mindspore.nn.probability.distribution.Poisson

class mindspore.nn.probability.distribution.**Poisson** (*rate=None, seed=None, dtype=mstype.float32, name='Poisson'*)

泊松分布 (Poisson Distribution)。离散随机分布, 取值范围为正自然数集, 概率质量函数为:

$$P(X = k) = \lambda^k \exp(-\lambda) / k!, k = 1, 2, \dots$$

其中 λ 为率参数。

参数:

- **rate** (list, numpy.ndarray, Tensor) - 泊松分布的率参数, 公式中的 λ 。默认值: None。
- **seed** (int) - 采样时使用的种子。如果为 None, 则使用全局种子。默认值: None。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值: mstype.float32。
- **name** (str) - 分布的名称。默认值: 'Poisson'。

说明: *rate* 必须大于 0。 *dist_spec_args* 是 *rate*。

异常:

- **ValueError** - *rate* 中元素小于等于 0。

支持平台:

Ascend

样例:

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize an Poisson distribution of the rate 0.5.
>>> p1 = msd.Poisson([0.5], dtype=mindspore.float32)
>>> # An Poisson distribution can be initialized without arguments.
>>> # In this case, `rate` must be passed in through `args` during function calls.
>>> p2 = msd.Poisson(dtype=mindspore.float32)
>>>
>>> # Here are some tensors used below for testing
>>> value = Tensor([1, 2, 3], dtype=mindspore.int32)
>>> rate_a = Tensor([0.6], dtype=mindspore.float32)
>>> rate_b = Tensor([0.2, 0.5, 0.4], dtype=mindspore.float32)
>>>
>>> # Private interfaces of probability functions corresponding to public interfaces,
>>> # including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`, are the
>>> # same as follows.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
```

(下页继续)

(续上页)

```

>>> #     rate (Tensor): the rate of the distribution. Default: self.rate.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing `prob` by the name of the function.
>>> ans = p1.prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to distribution b.
>>> ans = p1.prob(value, rate_b)
>>> print(ans.shape)
(3,)
>>> # `rate` must be passed in during function calls.
>>> ans = p2.prob(value, rate_a)
>>> print(ans.shape)
(3,)
>>> # Functions `mean`, `mode`, `sd`, and `var` have the same arguments as follows.
>>> # Args:
>>> #     rate (Tensor): the rate of the distribution. Default: self.rate.
>>> # Examples of `mean`, `sd`, `mode`, and `var` are similar.
>>> ans = p1.mean() # return 2
>>> print(ans.shape)
(1,)
>>> ans = p1.mean(rate_b) # return 1 / rate_b
>>> print(ans.shape)
(3,)
>>> # `rate` must be passed in during function calls.
>>> ans = p2.mean(rate_a)
>>> print(ans.shape)
(1,)
>>> # Examples of `sample`.
>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ()
>>> #     probs1 (Tensor): the rate of the distribution. Default: self.rate.
>>> ans = p1.sample()
>>> print(ans.shape)
(1, )
>>> ans = p1.sample((2,3))
>>> print(ans.shape)
(2, 3, 1)
>>> ans = p1.sample((2,3), rate_b)
>>> print(ans.shape)
(2, 3, 3)
>>> ans = p2.sample((2,3), rate_a)
>>> print(ans.shape)
(2, 3, 1)

```

property rate

返回分布的率参数。

返回：

Tensor, 率参数的值。

cdf (value, rate)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数：

- **value** (Tensor) - 要计算的值。

- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 累积分布函数的值。

log_cdf (*value*, *rate*)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value*, *rate*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 概率的对数。

log_survival (*value*, *rate*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 生存函数的对数。

mean (*rate*)

计算期望。

参数:

- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 概率分布的期望。

mode (*rate*)

计算众数。

参数:

- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 概率分布的众数。

prob (*value*, *rate*)

计算给定值的概率。对于离散分布是计算概率质量函数 (Probability Mass Function)。

参数:

- **value** (Tensor) - 要计算的值。
- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 概率值。

sample (*shape, rate*)

采样函数。

参数:

- **shape** (tuple) - 样本的 shape。
- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 根据概率分布采样的样本。

sd (*rate*)

计算标准差。

参数:

- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 概率分布的标准差。

survival_function (*value, rate*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 生存函数的值。

var (*rate*)

计算方差。

参数:

- **rate** (Tensor) - 率参数。默认值: `None`。

返回:

Tensor, 概率分布的方差。

22.2.16 mindspore.nn.probability.distribution.StudentT

class mindspore.nn.probability.distribution.**StudentT** (*df=None, mean=None, sd=None, seed=None, dtype=mstype.float32, name='StudentT'*)

StudentT 分布 (StudentT distribution)。连续随机分布，取值范围为 $(-\infty, \infty)$ ，概率密度函数为

$$f(x, \nu, \mu, \sigma) = (1 + y^2 / \nu)^{-(0.5 * (\nu + 1))} / Z$$

其中 $y = (x - \mu) / \sigma$ 、 $Z = \text{abs}(\sigma) * \sqrt{\nu * \pi} * \Gamma(0.5 * \nu) / \Gamma(0.5 * (\nu + 1))$ 和 ν, μ, σ 为分别为 StudentT 分布的自由度，期望与标准差。

参数：

- **df** (Union[int, float, list, numpy.ndarray, Tensor], 可选) - StudentT 分布的自由度。如果输入为 None，那么分布的自由度将在运行时传入。默认值：None。
- **mean** (Union[int, float, list, numpy.ndarray, Tensor], 可选) - StudentT 分布的平均值。如果输入为 None，那么分布的平均值将在运行时传入。默认值：None。
- **sd** (Union[int, float, list, numpy.ndarray, Tensor], 可选) - StudentT 分布的标准差。如果输入为 None，那么分布的标准差将在运行时传入。默认值：None。
- **seed** (int, 可选) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype, 可选) - 事件样例的类型。默认值：mstype.float32。
- **name** (str, 可选) - 分布的名称。默认值：'StudentT'。

说明：

- **df** 必须大于 0。
- **sd** 必须大于 0。
- **dtype** 必须是 float，因为 StudentT 分布是连续的。
- 如果在方法函数调用中传入参数 **df**、**mean** 或者 **sd**，则计算中会使用传参值，否则就会使用初始化时的参数值。

异常：

- **ValueError** - **df** 中元素不大于 0。
- **ValueError** - **sd** 中元素不大于 0。
- **TypeError** - **dtype** 不是 float 的子类。

支持平台：

CPU

样例：

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a StudentT distribution of the df 2.0, the mean 3.0 and the standard
    ↪ deviation 4.0.
>>> n1 = msd.StudentT(2.0, 3.0, 4.0, dtype=mindspore.float32)
```

(下页继续)

(续上页)

```

>>> # A StudentT distribution can be initialized without arguments.
>>> # In this case, `df`, `mean` and `sd` must be passed in through arguments.
>>> n2 = msd.StudentT(dtype=mindspore.float32)
>>> # Here are some tensors used below for testing
>>> value = Tensor([1.0, 2.0, 3.0], dtype=mindspore.float32)
>>> df_a = Tensor([2.0], dtype=mindspore.float32)
>>> mean_a = Tensor([2.0], dtype=mindspore.float32)
>>> sd_a = Tensor([2.0, 2.0, 2.0], dtype=mindspore.float32)
>>> df_b = Tensor([1.0], dtype=mindspore.float32)
>>> mean_b = Tensor([1.0], dtype=mindspore.float32)
>>> sd_b = Tensor([1.0, 1.5, 2.0], dtype=mindspore.float32)
>>> ans = n1.log_prob(value)
>>> print(ans.shape)
(3,)
>>> # Evaluate with respect to the distribution b.
>>> ans = n1.log_prob(value, df_b, mean_b, sd_b)
>>> print(ans.shape)
(3,)
>>> # `mean` and `sd` must be passed in during function calls
>>> ans = n2.log_prob(value, df_a, mean_a, sd_a)
>>> print(ans.shape)
(3,)

```

log_prob (*value*, *df=None*, *mean=None*, *sd=None*)

计算 StudentT 分布给定值对应的概率的对数。

参数：

- **value** (Tensor) - 要计算的值。
- **df** (Tensor, 可选) - 分布的自由度。默认值：None。
- **mean** (Tensor, 可选) - 分布的期望。默认值：None。
- **sd** (Tensor, 可选) - 分布的扩散度。默认值：None。

返回：

Tensor, 概率密度函数的对数。

22.2.17 mindspore.nn.probability.distribution.TransformedDistribution

class mindspore.nn.probability.distribution.**TransformedDistribution** (*bijector*, *distribution*,
seed=None,
name='transformed_distribution')

转换分布 (Transformed Distribution)。该类包含一个 Bijector 和一个分布，并通过 Bijector 定义的操作将原始分布转换为新分布。如果原始分布为 X ，Bijector 的映射函数为 $g(x)$ ，那么对应的转换分布为 $Y = g(X)$ 。

参数：

- **bijector** (Bijector) - 要执行的转换。
- **distribution** (Distribution) - 原始分布。必须具有 float 数据类型。
- **seed** (int) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。如果在初始化 TransformedDistribution 对象时给出了此种子，则对象的采样函数将使用此种子；否则，将使用基础分布的种子。
- **name** (str) - 转换分布的名称。默认值：'transformed_distribution'。

说明：用于初始化原始分布的参数不能为 `None`。例如，由于未指定 `mean` 和 `sd`，因此无法使用 `mynormal = msd.Normal(dtype=mindspore.float32)` 初始化 `TransformedDistribution`。`batch_shape` 为原始分布的 batch 的 shape。`broad-cast_shape` 为原始分布和概率映射经过广播后的 shape。`is_scalar_batch` 为 `True` 当且仅当原始分布和概率映射的 batch 均为标量。`default_parameters`、`parameter_names` 和 `parameter_type` 由原始分布的数据类型决定。衍生类可以通过调用 `reset_parameters` 后再调用 `add_parameter` 来添加参数以覆盖 `default_parameters` 和 `parameter_names`。

异常：

- **`TypeError`** - bijector 不是 `Bijector` 类。
- **`TypeError`** - distribution 不是 `Distribution` 类。

支持平台：

Ascend GPU CPU

样例：

```
>>> import numpy as np
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> import mindspore.nn.probability.bijector as msb
>>> from mindspore import Tensor
>>> class Net(nn.Cell):
...     def __init__(self, shape, dtype=mindspore.float32, seed=0, name='transformed_
    ↪distribution'):
...         super(Net, self).__init__()
...         # create TransformedDistribution distribution
...         self.exp = msb.Exp()
...         self.normal = msd.Normal(0.0, 1.0, dtype=dtype)
...         self.lognormal = msd.TransformedDistribution(self.exp, self.normal, seed=seed,
    ↪name=name)
...         self.shape = shape
...
...     def construct(self, value):
...         cdf = self.lognormal.cdf(value)
...         sample = self.lognormal.sample(self.shape)
...         return cdf, sample
>>> shape = (2, 3)
>>> net = Net(shape=shape, name="LogNormal")
>>> x = np.array([2.0, 3.0, 4.0, 5.0]).astype(np.float32)
>>> tx = Tensor(x, dtype=mindspore.float32)
>>> cdf, sample = net(tx)
>>> print(sample.shape)
(2, 3)
```

property bijector

返回概率映射函数。

返回：

`Bijector`，概率映射函数。

property distribution

返回变化前的概率分布。

返回：

Distribution, 变化前的概率分布。

property dtype

返回分布的数据类型。

返回:

mindspore.dtype, 分布的数据类型。

property is_linear_transformation

返回概率映射函数是否为线性映射。

返回:

Bool, 概率映射函数为线性映射则返回 True, 否则返回 False。

cdf (value)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数:

- **value** (Tensor) - 要计算的值。

返回:

Tensor, 累积分布函数的值。

log_cdf (value)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。

返回:

Tensor, 累积分布函数的对数。

log_prob (value)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。

返回:

Tensor, 累积分布函数的对数。

log_survival (value)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。

返回:

Tensor, 生存函数的对数。

mean ()

计算期望。

返回:

Tensor, 概率分布的期望。

prob (*value*)

计算给定值下的概率。

参数：

- **value** (Tensor) - 要计算的值。

返回：

Tensor, 概率值。

sample (*shape*)

采样函数。

参数：

- **shape** (tuple) - 样本的 shape。

返回：

Tensor, 根据概率分布采样的样本。

survival_function (*value*)

计算给定值对应的生存函数。

参数：

- **value** (Tensor) - 要计算的值。

返回：

Tensor, 生存函数的值。

22.2.18 mindspore.nn.probability.distribution.Uniform

class mindspore.nn.probability.distribution.**Uniform** (*low=None, high=None, seed=None, dtype=mstype.float32, name='Uniform'*)

均匀分布 (Uniform Distribution)。连续随机分布，取值范围为 $[a, b]$ ，概率密度函数为：

$$f(x, a, b) = 1/(b - a)$$

其中 a, b 为分别为均匀分布的下界和上界。

参数：

- **low** (int, float, list, numpy.ndarray, Tensor) - 分布的下限，公式中的 a 。默认值：None。
- **high** (int, float, list, numpy.ndarray, Tensor) - 分布的上限，公式中的 b 。默认值：None。
- **seed** (int) - 采样时使用的种子。如果为 None，则使用全局种子。默认值：None。
- **dtype** (mindspore.dtype) - 事件样例的类型。默认值：mstype.float32。
- **name** (str) - 分布的名称。默认值：'Uniform'。

说明：

- low 必须小于 $high$ 。
- $dtype$ 必须是 float 类型，因为均匀分布是连续的。

异常：

- **ValueError** - *low* 大于等于 *high*。
- **TypeError** - *dtype* 不是 float 或 float 的子类。

支持平台:

Ascend GPU CPU

样例:

```
>>> import mindspore
>>> import mindspore.nn as nn
>>> import mindspore.nn.probability.distribution as msd
>>> from mindspore import Tensor
>>> # To initialize a Uniform distribution of the lower bound 0.0 and the higher bound 1.0.
>>> u1 = msd.Uniform(0.0, 1.0, dtype=mindspore.float32)
>>> # A Uniform distribution can be initialized without arguments.
>>> # In this case, `high` and `low` must be passed in through arguments during function
    ↳ calls.
>>> u2 = msd.Uniform(dtype=mindspore.float32)
>>>
>>> # Here are some tensors used below for testing
>>> value = Tensor([0.5, 0.8], dtype=mindspore.float32)
>>> low_a = Tensor([0., 0.], dtype=mindspore.float32)
>>> high_a = Tensor([2.0, 4.0], dtype=mindspore.float32)
>>> low_b = Tensor([-1.5], dtype=mindspore.float32)
>>> high_b = Tensor([2.5, 5.], dtype=mindspore.float32)
>>> # Private interfaces of probability functions corresponding to public interfaces,
    ↳ including
>>> # `prob`, `log_prob`, `cdf`, `log_cdf`, `survival_function`, and `log_survival`, have
    ↳ the same arguments.
>>> # Args:
>>> #     value (Tensor): the value to be evaluated.
>>> #     low (Tensor): the lower bound of the distribution. Default: self.low.
>>> #     high (Tensor): the higher bound of the distribution. Default: self.high.
>>> # Examples of `prob`.
>>> # Similar calls can be made to other probability functions
>>> # by replacing 'prob' by the name of the function.
>>> ans = u1.prob(value)
>>> print(ans.shape)
(2,)
>>> # Evaluate with respect to distribution b.
>>> ans = u1.prob(value, low_b, high_b)
>>> print(ans.shape)
(2,)
>>> # `high` and `low` must be passed in during function calls.
>>> ans = u2.prob(value, low_a, high_a)
>>> print(ans.shape)
(2,)
>>> # Functions `mean`, `sd`, `var`, and `entropy` have the same arguments.
>>> # Args:
>>> #     low (Tensor): the lower bound of the distribution. Default: self.low.
>>> #     high (Tensor): the higher bound of the distribution. Default: self.high.
>>> # Examples of `mean`. `sd`, `var`, and `entropy` are similar.
>>> ans = u1.mean() # return 0.5
>>> print(ans.shape)
()
>>> ans = u1.mean(low_b, high_b) # return (low_b + high_b) / 2
>>> print(ans.shape)
```

(下页继续)

(续上页)

```

(2,)
>>> # `high` and `low` must be passed in during function calls.
>>> ans = u2.mean(low_a, high_a)
>>> print(ans.shape)
(2,)
>>> # Interfaces of 'kl_loss' and 'cross_entropy' are the same.
>>> # Args:
>>> #     dist (str): the type of the distributions. Should be "Uniform" in this case.
>>> #     low_b (Tensor): the lower bound of distribution b.
>>> #     high_b (Tensor): the upper bound of distribution b.
>>> #     low_a (Tensor): the lower bound of distribution a. Default: self.low.
>>> #     high_a (Tensor): the upper bound of distribution a. Default: self.high.
>>> # Examples of `kl_loss`. `cross_entropy` is similar.
>>> ans = u1.kl_loss('Uniform', low_b, high_b)
>>> print(ans.shape)
(2,)
>>> ans = u1.kl_loss('Uniform', low_b, high_b, low_a, high_a)
>>> print(ans.shape)
(2,)
>>> # Additional `high` and `low` must be passed in.
>>> ans = u2.kl_loss('Uniform', low_b, high_b, low_a, high_a)
>>> print(ans.shape)
(2,)
>>> # Examples of `sample`.
>>> # Args:
>>> #     shape (tuple): the shape of the sample. Default: ()
>>> #     low (Tensor): the lower bound of the distribution. Default: self.low.
>>> #     high (Tensor): the upper bound of the distribution. Default: self.high.
>>> ans = u1.sample()
>>> print(ans.shape)
()
>>> ans = u1.sample((2, 3))
>>> print(ans.shape)
(2, 3)
>>> ans = u1.sample((2, 3), low_b, high_b)
>>> print(ans.shape)
(2, 3, 2)
>>> ans = u2.sample((2, 3), low_a, high_a)
>>> print(ans.shape)
(2, 3, 2)

```

property high

返回分布的上限。

返回：

Tensor，分布的上限值。

property low

返回分布的下限。

返回：

Tensor，分布的下限值。

cdf (value, high, low)

计算给定值的累积分布函数 (Cumulative Distribution Function, CDF)。

参数：

- **value** (Tensor) - 要计算的值。
- **high** (Tensor) - 分布的上限值。默认值: `None`。
- **low** (Tensor) - 分布的下限值。默认值: `None`。

返回:

Tensor, 累积分布函数的值。

cross_entropy (*dist, high_b, low_b, high, low*)

计算分布 a 和 b 之间的交叉熵。

参数:

- **dist** (str) - 分布的类型。
- **high_b** (Tensor) - 分布 b 的上限值。
- **low_b** (Tensor) - 分布 b 的下限值。
- **high** (Tensor) - 分布 a 的上限值。默认值: `None`。
- **low** (Tensor) - 分布 a 的下限值。默认值: `None`。

返回:

Tensor, 交叉熵的值。

entropy (*high, low*)

计算熵。

参数:

- **high** (Tensor) - 分布的上限值。默认值: `None`。
- **low** (Tensor) - 分布的下限值。默认值: `None`。

返回:

Tensor, 熵的值。

kl_loss (*dist, high_b, low_b, high, low*)

计算 KL 散度, 即 $KL(a||b)$ 。

参数:

- **dist** (str) - 分布的类型。
- **high_b** (Tensor) - 分布 b 的上限值。
- **low_b** (Tensor) - 分布 b 的下限值。
- **high** (Tensor) - 分布 a 的上限值。默认值: `None`。
- **low** (Tensor) - 分布 a 的下限值。默认值: `None`。

返回:

Tensor, KL 散度。

log_cdf (*value, high, low*)

计算给定值对应的累积分布函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **high** (Tensor) - 分布的上限值。默认值: `None`。

- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 累积分布函数的对数。

log_prob (*value, high, low*)

计算给定值对应的概率的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 概率的对数。

log_survival (*value, high, low*)

计算给定值对应的生存函数的对数。

参数:

- **value** (Tensor) - 要计算的值。
- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 生存函数的对数。

mean (*high, low*)

计算期望。

参数:

- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 概率分布的期望。

mode (*high, low*)

计算众数。

参数:

- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 概率分布的众数。

prob (*value, high, low*)

计算给定值的概率。对于连续分布是计算概率密度函数 (Probability Density Function)。

参数:

- **value** (Tensor) - 要计算的值。

- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 概率值。

sample (*shape, high, low*)

采样函数。

参数:

- **shape** (tuple) - 样本的 shape。
- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 根据概率分布采样的样本。

sd (*high, low*)

计算标准差。

参数:

- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 概率分布的标准差。

survival_function (*value, high, low*)

计算给定值对应的生存函数。

参数:

- **value** (Tensor) - 要计算的值。
- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 生存函数的值。

var (*high, low*)

计算方差。

参数:

- **high** (Tensor) - 分布的上限值。默认值: None。
- **low** (Tensor) - 分布的下限值。默认值: None。

返回:

Tensor, 概率分布的方差。

MindSpore 的 ReWrite 模块允许用户通过自定义规则，修改网络的前向计算过程，支持插入、删除和替换语句等操作。

关于如何快速使用 ReWrite，请参考 [使用 ReWrite 修改网络](#)。

class mindspore.rewrite.Node (node: NodeImpl)

节点 (Node) 在神经网络的计算图中可以被理解为一个基本的数据结构单元，它代表了网络中的某个操作或计算步骤。每个节点通常对应着源码中的一个语句或表达式，它包含了执行该操作所需的信息，如操作类型、输入数据、输出结果以及与其他节点的连接关系。

节点可以表达前向计算过程的 Cell 调用语句、Primitive 调用语句、算术运算语句、返回语句等。

参数：

- **node** (NodeImpl) - Node 的内部实现实例。建议调用 Node 类下的指定方法来创建 Node，例如 `create_call_cell`，而不是直接调用 Node 的构造函数。用户不需关心 NodeImpl 是什么，只需作为句柄看待。

static `create_call_cell` (cell: Cell, targets: List[Union[ScopedValue, str]], args: List[ScopedValue] = None, kwargs: Dict[str, ScopedValue] = None, name: str = "", is_sub_net: bool = False)

通过该接口可以根据 cell 对象创建一个 Node 实例。节点对应的源代码格式：

```
targets = self.name(*args, **kwargs)。
```

参数：

- **cell** (Cell) - 该节点对应的前向计算的 Cell 对象。
- **targets** (List[Union[ScopedValue, str]]) - 表示输出名称。在源代码中作为节点的输出变量名。
- **args** (List[ScopedValue]) - 该节点的参数名称。用作源代码中代码语句的参数。默认值：None，表示 cell 没有参数输入。
- **kwargs** (Dict[str, ScopedValue]) - 用来说明带有关键字的形参的输入参数名称。输入名称在源代码中作为语句表达式中的 `kwargs`。key 的类型必须是 str，value 的类型必须是 ScopedValue。默认值：None，表示 cell 没有 kwargs 输入。
- **name** (str) - 表示节点的名称。用作源代码中的字段名称。当未提供名称时，ReWrite 将根据 target 生成一个默认名称。Rewrite 将在插入节点时检查并确保名称的唯一性。默认值：""。

- **is_sub_net** (bool) - 表示 *cell* 是否是一个网络。如果 *is_sub_net* 为 True , Rewrite 将尝试将 *cell* 解析为 *TreeNode* , 否则为 *CallCell* 节点。默认值: False 。

返回:

Node 实例。

异常:

- **TypeError** - 如果参数 *cell* 不是 *Cell* 类型。
- **TypeError** - 如果参数 *targets* 不是 list 类型。
- **TypeError** - 如果参数 *targets* 的成员不是 str 或者 *ScopedValue* 类型。
- **TypeError** - 如果参数 *args* 不是 *ScopedValue* 类型。
- **TypeError** - 如果参数 *kwargs* 的 *key* 不是 str 类型或者 *value* 不是 *ScopedValue* 类型。

样例:

```
>>> from mindspore.rewrite import SymbolTree, ScopedValue
>>> import mindspore.nn as nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> position = stree.after(node)
>>> new_node = node.create_call_cell(cell=nn.ReLU(), targets=['x'],
...                                args=[ScopedValue.create_naming_value('x')], name=
↳ 'new_relu')
>>> stree.insert(position, new_node)
>>> print(type(new_node))
<class 'mindspore.rewrite.api.node.Node'>
```

static create_call_function (*function: FunctionType, targets: List[Union[ScopedValue, str]], args: List[ScopedValue] = None, kwargs: Dict[str, ScopedValue] = None*)

通过该接口可以根据一个函数调用来创建一个 Node 实例。

说明: 函数内部的代码不会被解析。

参数:

- **function** (FunctionType) - 被调用的函数定义。
- **targets** (List[Union[ScopedValue, str]]) - 表示输出名称。在源代码中作为节点的输出变量名。
- **args** (List[ScopedValue]) - 该节点参数名称。用作源代码中代码语句的参数。默认值: None , 表示 *function* 没有参数输入。
- **kwargs** (Dict[str, ScopedValue]) - 用来说明带有关键字的形参的输入参数名称。输入名称在源代码中作为语句表达式中的 *kwargs*。 *key* 的类型必须是 str , *value* 的类型必须是 *ScopedValue*。默认值: None , 表示 *function* 没有 *kwargs* 输入。

返回:

Node 实例。

异常:

- **TypeError** - 如果参数 *function* 不是 *FunctionType* 类型。

- **TypeError** - 如果参数 *targets* 不是 list 类型。
- **TypeError** - 如果参数 *targets* 的成员不是 str 或者 ScopedValue 类型。
- **TypeError** - 如果参数 *args* 不是 ScopedValue 类型。
- **TypeError** - 如果参数 *kwargs* 的 *key* 不是 str 类型或者 *value* 不是 ScopedValue 类型。

样例:

```
>>> from mindspore.rewrite import SymbolTree, ScopedValue
>>> import mindspore.nn as nn
>>> from mindspore import ops
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> position = stree.after(node)
>>> new_node = node.create_call_function(function=ops.abs, targets=['x'],
...                                     args=[ScopedValue.create_naming_value('x')])
>>> stree.insert(position, new_node)
>>> print(new_node.get_node_type())
NodeType.CallFunction
```

get_args()

获取当前节点参数列表。

返回:

参数列表, 参数类型为 ScopedValue。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> print(node.get_args())
[x]
```

get_inputs()

获取一个节点列表, 列表里的节点的输出作为当前节点的输入。

返回:

节点列表。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv2")
>>> inputs = node.get_inputs()
>>> print([input.get_name() for input in inputs])
['max_pool2d']
```

get_instance_type()

获取当前节点对应的代码语句里调用的对象类型。

- 如果当前节点的 *node_type* 是 *CallCell*，表示该节点的语句调用了 *Cell* 类型对象。
- 如果当前节点的 *node_type* 是 *CallPrimitive*，表示该节点的语句调用了 *Primitive* 类型对象。
- 如果当前节点的 *node_type* 是 *Tree*，表示该节点的语句调用了 *网络类型* 的对象。
- 如果当前节点的 *node_type* 是 *Python*、*Input*、*Output*、*CallMethod*，返回的对象类型是 *NoneType*。

返回：

当前节点对应的代码语句里调用的对象类型。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> instance_type = node.get_instance_type()
>>> print(instance_type)
<class 'mindspore.nn.layer.conv.Conv2d'>
```

get_kwargs()

获取当前节点的关键字参数列表。

返回：

一个包含关键字参数的字典，*key* 的类型为 *str*，*value* 的类型为 *ScopedValue*。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> from mindspore import nn
>>>
>>> class ReLUNet(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.relu = nn.ReLU()
...
...     def construct(self, input):
...         output = self.relu(input=input)
...         return output
>>>
>>> net = ReLUNet()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("relu")
>>> print(node.get_kwargs())
{'input': input}
```

get_name()

获取当前节点的名称。当节点被插入到 *SymbolTree* 时，节点的名称在 *SymbolTree* 中应该是唯一的。

返回：

节点的名称，类型为 *str*。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> name = node.get_name()
>>> print(name)
conv1
```

get_node_type()

获取当前节点的类型。节点类型详见 `mindspore.rewrite.NodeType`。

返回：

`NodeType`，当前节点的类型。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> node_type = node.get_node_type()
>>> print(node_type)
NodeType.CallCell
```

get_sub_tree()

获取类型为 `NodeType.Tree` 的节点里保存的符号树。节点类型详见 `mindspore.rewrite.NodeType`。

返回：

保存在 `Tree` 类型节点里的符号树。

异常：

- **TypeError** - 如果当前节点的类型不是 `NodeType.Tree`。
- **AttributeError** - 如果当前 `Tree` 类型节点里没有保存符号树。

样例：

```
>>> import mindspore.nn as nn
>>> from mindspore.rewrite import SymbolTree
>>>
>>> class SubNet(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.relu = nn.ReLU()
...
...     def construct(self, x):
...         x = self.relu(x)
...         return x
...
>>> class Net(nn.Cell):
...     def __init__(self):
...         super().__init__()
...         self.subnet = SubNet()
```

(下页继续)

(续上页)

```

...
...     def construct(self, x):
...         x = self.subnet(x)
...         return x
...
>>>
>>> net = Net()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("subnet")
>>> print(type(node.get_sub_tree()))
<class 'mindspore.rewrite.api.symbol_tree.SymbolTree'>

```

get_symbol_tree()

获取当前节点所属的 SymbolTree。

返回：

SymbolTree，如果当前节点不属于任何 SymbolTree，则返回 None。

样例：

```

>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> print(type(node.get_symbol_tree()))
<class 'mindspore.rewrite.api.symbol_tree.SymbolTree'>

```

get_targets()

获取当前节点的输出值列表。

返回：

输出值列表，参数类型为 ScopedValue。

get_users()

获取一个节点列表，列表里的节点使用当前节点的输出作为输入。

返回：

节点列表。

样例：

```

>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> users = node.get_users()
>>> print([user.get_name() for user in users])
['relu']

```

set_arg(index: int, arg: Union[ScopedValue, str])

设置当前节点的输入参数。

参数：

- **index** (int) - 要设置的参数索引。
- **arg** (Union[ScopedValue, str]) - 新参数的值。

异常:

- **TypeError** - 如果参数 *index* 不是 int 类型。
- **TypeError** - 如果参数 *arg* 不是 str 或者 ScopedValue 类型。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("relu_3")
>>> node.set_arg(0, "fc1")
>>> print(node.get_args())
[fc1]
```

set_arg_by_node (*arg_idx*: int, *src_node*: 'Node', *out_idx*: Optional[int] = None)

将另一个节点设置为当前节点的输入。

参数:

- **arg_idx** (int) - 要设置的参数索引。
- **src_node** (Node) - 输入的节点。
- **out_idx** (int, 可选) - 指定输入节点的哪个输出作为当前节点输入, 则取第一个输出。默认值: None。

异常:

- **TypeError** - 如果参数 *arg_idx* 不是 int 类型。
- **ValueError** - 如果参数 *arg_idx* 超出了当前节点参数数量。
- **TypeError** - 如果参数 *src_node* 不是 Node 类型。
- **TypeError** - 如果参数 *out_idx* 不是 int 类型。
- **ValueError** - 如果参数 *out_idx* 超出了 *src_node* 的输出数量。
- **ValueError** - 当 *out_idx* 为 None 或者没有给 *out_idx* 赋值时, 参数 *src_node* 有多个输出。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> src_node = stree.get_node("fc1")
>>> dst_node = stree.get_node("relu_3")
>>> dst_node.set_arg_by_node(0, src_node, 0)
>>> print(dst_node.get_args())
[fc1_var]
```

class mindspore.rewrite.NodeType

NodeType 表示 Node 的类型。

- **Unknown**: 未初始化的节点类型。

- **CallCell**: *CallCell* 节点表示在前向计算中调用 **Cell** 对象。
- **CallPrimitive**: *CallPrimitive* 节点代表在前向计算中调用 **Primitive** 对象。
- **CallFunction**: *CallFunction* 节点代表在前向计算中调用了一个函数。
- **CallMethod**: *CallMethod* 不能对应到 **Cell** 或者 **Primitive** 的节点。
- **Python**: *Python* 节点代表不支持的 *ast* 节点或无需解析的 *ast* 节点。
- **Input**: *Input* 节点代表 **SymbolTree** 的输入, 对应方法的参数。
- **Output**: *Output* 节点代表 **SymbolTree** 的输出, 对应方法的 *return* 语句。
- **Tree**: *Tree* 节点代表前向计算中调用了别的网络。
- **CellContainer**: *CellContainer* 节点代表在前向计算中调用 `mindspore.nn.SequentialCell` 函数。
- **MathOps**: *MathOps* 节点代表在前向计算中的一个运算操作, 如加法运算或比较运算。
- **ControlFlow**: *ControlFlow* 节点代表一个控制流语句, 如 *if* 语句。

class mindspore.rewrite.ScopedValue (arg_type: ValueType, scope: str = "", value=None)

ScopedValue 表示具有完整范围的值。

ScopedValue 用于表示: 左值, 如赋值语句的目标, 或可调用对象, 如调用语句的 *func*, 或右值, 如赋值语句的 *args* 和 *kwargs*。

参数:

- **arg_type** (ValueType) - 当前值的类型。
- **scope** (str, 可选) - 字符串表示当前值的范围。以 "self.var1" 为例, 这个 *var1* 的作用域是 "self"。默认值: ""。
- **value** - 当前 ScopedValue 中保存的值。值的类型对应于 *arg_type*。默认值: None。

static create_name_values (names: Union[List[str], Tuple[str]], scopes: Union[List[str], Tuple[str]] = None)

创建 ScopedValue 的列表。

参数:

- **names** (List[str] or Tuple[str]) - 引用变量的名称, 类型为 str 的列表或元组。
- **scopes** (List[str] or Tuple[str], 可选) - 引用变量的范围, 类型为 str 的列表或元组。默认值: None, 表示没有指定作用范围。

返回:

ScopedValue 的实例列表。

异常:

- **TypeError** - 如果 *names* 不是 *list* 或 *tuple* 或者其中的元素不是 str 类型。
- **TypeError** - 如果 *scopes* 不是 *list* 或 *tuple* 或者其中的元素不是 str 类型。
- **ValueError** - 如果 *names* 的长度不等于 *scopes* 的长度, 而作用域不是 None。

样例:

```
>>> from mindspore.rewrite import ScopedValue
>>> variables = ScopedValue.create_name_values(names=["z", "z_1"], scopes=["self", "self
↳"])
>>> print(variables)
[self.z, self.z_1]
```


classmethod `create_naming_value` (*name: str, scope: str = ""*)

创建一个使用变量名称命名的 `ScopedValue`。`NamingValue` 表示对另一个变量的引用。

参数：

- **name** (str) - 表示变量的字符串。
- **scope** (str, 可选) - 表示变量范围的字符串，默认值：""，表示没有指定作用范围。

返回：

`ScopedValue` 的实例。

异常：

- **TypeError** - 如果 *name* 不是 str 类型。
- **TypeError** - 如果 *scope* 不是 str 类型。

样例：

```
>>> from mindspore.rewrite import ScopedValue
>>> variable = ScopedValue.create_naming_value("conv", "self")
>>> print(variable)
self.conv
```

classmethod `create_variable_value` (*value*)

创建一个保存变量的 `ScopedValue`。`ScopedValue` 的类型由值的类型决定。`ScopedValue` 的范围是空的。

参数：

- **value** - 要转换为 `ScopedValue` 的值。

返回：

`ScopedValue` 的实例。

样例：

```
>>> from mindspore.rewrite import ScopedValue
>>> variable = ScopedValue.create_variable_value(2)
>>> print(variable)
2
```

class `mindspore.rewrite.SymbolTree` (*handler: SymbolTreeImpl*)

`SymbolTree` 保存了一个网络的信息，包括网络前向计算过程的语句，和语句输入输出之间的拓扑关系。

网络里的语句以节点的形式保存在 `SymbolTree` 中，通过对 `SymbolTree` 里的节点进行处理，可以实现网络代码的删除、插入、替换等操作，并得到修改后的网络代码及网络实例。

参数：

- **handler** (`SymbolTreeImpl`) - `SymbolTree` 内部实现实例。建议调用 `SymbolTree` 下的 `create` 方法来创建 `SymbolTree`，而不直接调用 `SymbolTree` 的构造函数。用户不需关心 `SymbolTreeImpl` 是什么，只需作为句柄看待。

after (*node: Union[Node, str]*)

返回一个位置信息，位置为 *node* 之后。该接口的返回值作为插入操作的参数使用。

参数：

- **node** (`Union[Node, str]`) - 指定插入位置在哪个节点之后，可以是 `Node` 或者 `Node` 的名称。

返回：

`Position`，指定插入节点的位置。

异常:

- **TypeError** - 参数不是 Node 或 str 类型。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> for node in stree.nodes():
...     if node.get_name() == "conv1":
...         position = stree.after(node)
```

before (*node*: Union[Node, str])

返回一个位置信息, 位置为 *node* 之前。该接口的返回值作为插入操作的参数使用。

参数:

- **node** (Union[Node, str]) - 指定插入位置在哪个节点之前, 可以是 Node 或者 Node 的名称。

返回:

Position, 指定插入节点的位置。

异常:

- **TypeError** - 参数不是 Node 或 str 类型。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> for node in stree.nodes():
...     if node.get_name() == "conv1":
...         position = stree.before(node)
```

classmethod create (*network*)

通过传入网络实例 *network*, 创建一个 SymbolTree 对象。

该接口会解析传入的网络实例, 将前向计算过程的每一条源码语句展开, 并解析为节点, 存储在 SymbolTree 中。具体流程如下:

1. 获取网络实例对应的源码代码。
2. 对网络进行 AST 解析, 获取网络里各个语句的 AST 节点 (抽象语法树)。
3. 将网络前向计算过程里的复杂语句展开为多个简单语句。
4. 创建 SymbolTree 对象, 每个 SymbolTree 对应一个网络实例。
5. 使用 rewrite 节点存储网络前向计算过程的每条语句, 节点记录了语句的输入、输出等信息。
6. 将 rewrite 节点保存到 SymbolTree 里, 同时更新和维护节点间的拓扑连接关系。
7. 返回网络实例对应的 SymbolTree 对象。

如果网络的前向计算过程里调用了类型为 `mindspore.nn.Cell` 的用户自定义网络, `rewrite` 会为对应语句生成类型为 `NodeType.Tree` 的节点, 这类节点内部保存了一个新的 SymbolTree, 这个 SymbolTree 解析并维护着自定义网络的节点信息。

如果网络的前向计算过程里调用了以下类型的语句，`rewrite` 会将该语句所对应的内部语句进行解析，并生成对应节点：

- `mindspore.nn.SequentialCell`
- 函数调用（不包括 Python 内置函数和三方库函数）
- 控制流语句（如 `if` 语句）

说明：由于网络在 `rewrite` 操作期间，控制流的具体执行分支还处于未知状态，因此控制流内部的节点和外部的节点之间不会建立拓扑信息。用户在控制流外部使用 `mindspore.rewrite.Node.get_inputs()` 和 `mindspore.rewrite.Node.get_users()` 接口获取节点时，无法获取控制流内部的节点。用户在控制流内部使用这些接口，也无法获取控制流外部的节点。因此用户在进行网络修改时，需要手动处理好控制流内部和外部的节点信息。

当前 `rewrite` 模块存在以下语法限制：

- 仅支持类型为 `mindspore.nn.Cell` 的网络作为 `rewrite` 模块的输入。
- 暂不支持对单行控制流语法（如单行 `if-else`、单行 `for` 循环等）进行解析。
- 暂不支持对装饰器语法进行解析。
- 暂不支持对局部类和内嵌类进行解析，即类的定义需要放在最外层。
- 暂不支持对闭包语法进行解析，即类外函数的定义需要放在最外层。
- 暂不支持对 `lambda` 表达式语法进行解析。
- 暂不支持对全局变量进行解析，即需要将全局变量转换为类变量或局部变量后再使用。
- 暂不支持对父类里的方法进行解析。

对于不支持解析的语句，`rewrite` 会为对应语句生成类型为 `NodeType.Python` 的节点，以确保 `rewrite` 后的网络可以正常运行。`Python` 节点不支持对语句的输入和输出进行修改，且可能出现变量名与 `rewrite` 生成的变量名的问题，此时用户需要手动对变量名进行调整。

参数：

- **network** (Cell) - 待修改的网络实例。

返回：

`SymbolTree`，基于 `network` 创建的 `SymbolTree`。

异常：

- **TypeError** - 参数 `network` 不是 `Cell` 类型对象。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> print(type(stree))
<class 'mindspore.rewrite.api.symbol_tree.SymbolTree'>
```

erase (`node: Union[Node, str]`)

删除 `SymbolTree` 中的一个节点。

参数：

- **node** (Union[Node, str]) - 被删除的节点。可以是 Node 或者 Node 的名称。

返回:

如果 *node* 属于当前的 SymbolTree 则返回被删除节点。否则返回 None。

异常:

- **TypeError** - 参数不是 Node 或 str 类型。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> stree.erase(node)
```

get_code()

获取 SymbolTree 里的网络信息所对应的源码。如果网络已经被修改过，则返回的是修改后的源码。

返回:

str, SymbolTree 对应的源码字符串。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> codes = stree.get_code()
>>> print(codes)
```

get_network()

获取基于 SymbolTree 生成的网络对象。源码会保存到文件中，文件保存在当前目录的 *rewritten_network* 文件夹里。

说明:

- **rewrite** 模块对网络的修改基于对原有网络实例的 AST 树的修改实现，且新的网络实例会从原有网络实例里获取属性信息，因此，新网络实例和原有网络实例存在数据关联，原有网络不应该再被使用。
 - 由于新网络和原有网络实例存在数据关联，暂不支持使用 **rewrite** 生成的源码文件手动创建网络实例。
-

返回:

根据 SymbolTree 生成的网络对象。

样例:

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> new_net = stree.get_network()
```

get_node (*node_name: str*)

获取 SymbolTree 里名称为 *node_name* 的节点。

参数：

- **node_name** (str) - 节点名称。

返回：

名称为 *node_name* 的节点。如果 SymbolTree 里没有名称为 *node_name* 的节点，则返回 None。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node('conv1')
>>> print(node.get_name())
conv1
```

insert (*position, node: Node*)

在 SymbolTree 的 *position* 位置插入一个节点。*position* 通过 *before* 或 *after* 来获得。

参数：

- **position** (Position) - 插入位置。
- **node** (Node) - 要插入的节点。

返回：

Node，被插入的节点。

异常：

- **ValueError** - 如果 *position* 指定的不是该 SymbolTree 内的位置。
- **TypeError** - 如果参数 *position* 不是 Position 类型。
- **TypeError** - 如果参数 *node* 不是 Node 类型。

样例：

```
>>> from mindspore.rewrite import SymbolTree, ScopedValue
>>> import mindspore.nn as nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> position = stree.after(node)
>>> new_node = node.create_call_cell(cell=nn.ReLU(), targets=['x'],
...                                args=[ScopedValue.create_naming_value('x')], name=
↳ 'new_relu')
>>> stree.insert(position, new_node)
```

nodes (*all_nodes: bool = False*)

返回当前 SymbolTree 里节点的生成器，该接口用于遍历 SymbolTree 里的节点。

参数：

- **all_nodes** (bool) - 获取所有节点，包括在 *CallFunction* 节点、*CellContainer* 节点和子 *SymbolTree* 里面的节点。默认值：False。

返回：

SymbolTree 中节点的生成器。

异常：

- **TypeError** - 如果参数 *all_nodes* 不是 bool 类型。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> print([node.get_name() for node in stree.nodes()])
['input_x', 'conv1', 'relu', 'max_pool2d', 'conv2', 'relu_1', 'max_pool2d_1',
 'unaryop_not', 'if_node', 'flatten', 'fc1', 'relu_2', 'fc2', 'relu_3', 'fc3', 'return_1
↪']
```

print_node_tabulate (*all_nodes: bool = False*)

打印 *SymbolTree* 里节点的拓扑信息，包括节点类型、节点名称、节点对应代码、节点的输入输出关系等。

信息通过 print 接口输出到屏幕上，包括以下信息：

- **node type** (str): 节点类型，具体类型参考 *mindspore.rewrite.NodeType*。
- **name** (str): 节点名称。
- **codes** (str): 节点对应的 *SymbolTree* 里的代码语句。
- **arg providers** (Dict[int, Tuple[str, int]]): 格式为 *{[idx, (n, k)]}*，代表该节点的第 *idx* 个参数是节点 *n* 的第 *k* 个输出提供的。
- **target users** (Dict[int, List[Tuple[str, int]]]): 格式为 *{[idx, [(n, k)]]}*，代表该节点的第 *idx* 个输出被用作节点 *n* 的第 *k* 个参数。

参数：

- **all_nodes** (bool) - 打印所有节点的信息，包括在 *CallFunction* 节点、*CellContainer* 节点和子 *SymbolTree* 里面的节点。默认值：False。

异常：

- **TypeError** - 如果参数 *all_nodes* 不是 bool 类型。

样例：

```
>>> from mindspore.rewrite import SymbolTree
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> stree.print_node_tabulate()
```

replace (*old_node: Node, new_nodes: List[Node]*)

使用 *new_nodes* 列表里的节点来替代旧节点 *old_node*。

该接口会将 *new_nodes* 里的节点按顺序插入到 *SymbolTree* 中，然后删除旧节点 *old_node*。

说明:

- 仅支持一对一更换或一对多替换。
- 调用者应维护好 *new_nodes* 里每个节点间的拓扑关系, 以及 *new_nodes* 里的节点与原始树中节点的拓扑关系。

参数:

- **old_node** (Node) - 被替换节点。
- **new_nodes** (List[Node]) - 要替换进 SymbolTree 的节点列表。

返回:

替换到 SymbolTree 的节点列表的根节点。

异常:

- **TypeError** - 如果参数 *new_nodes* 不是 list, 或者列表中的成员不是 Node 类型。
- **TypeError** - 如果参数 *old_node* 不是 Node 类型。

样例:

```
>>> from mindspore.rewrite import SymbolTree, ScopedValue
>>> import mindspore.nn as nn
>>> # Define the network structure of LeNet5. Refer to
>>> # https://gitee.com/mindspore/docs/blob/r2.6.0rc1/docs/mindspore/code/lenet.py
>>> net = LeNet5()
>>> stree = SymbolTree.create(net)
>>> node = stree.get_node("conv1")
>>> new_node = node.create_call_cell(cell=nn.ReLU(), targets=['x'],
...                                args=[ScopedValue.create_naming_value('x')], name=
↳ 'new_relu')
>>> stree.replace(node, [new_node])
```

unique_name (name: str = 'output')

基于给定 *name*, 返回一个 SymbolTree 内唯一的新的名称。当需要一个不冲突的变量名时, 可以使用该接口。

参数:

- **name** (str, 可选) - 名称前缀。默认值: "output"。

返回:

str, 一个 SymbolTree 内唯一的新的名称, 名称格式为 *name_n*, 其中 *n* 为数字下标。如果输入 *name* 没有名称冲突, 则没有数字下标。

异常:

- **TypeError** - 如果参数 *name* 不是 str 类型。

class mindspore.rewrite.ValueType

ValueType 表示 ScopedValue 的类型。

- NamingValue 表示对另一个变量的引用。
- CustomObjValue 表示自定义类的实例, 或类型超出 ValueType 的基本类型和容器类型范围的对象。

Hal 中封装了设备管理、流管理、事件管理与内存管理的接口。MindSpore 从不同后端抽象出对应的上述模块，允许用户在 Python 层调度硬件资源。

24.1 设备管理

<code>mindspore.hal.device_count</code>	查询目标设备的数量，此接口将在后续版本中废弃，请使用接口 <code>mindspore.device_context.cpu.device_count()</code> 、 <code>mindspore.device_context.gpu.device_count()</code> 、 <code>mindspore.device_context.ascend.device_count()</code> 代替。
<code>mindspore.hal.get_arch_list</code>	返回此 MindSpore 包支持哪些后端架构，此接口将在后续版本中废弃。
<code>mindspore.hal.get_device_capability</code>	返回指定卡号设备的设备能力，此接口将在后续版本中废弃。
<code>mindspore.hal.get_device_name</code>	返回指定卡号设备的设备名称，此接口将在后续版本中废弃。
<code>mindspore.hal.get_device_properties</code>	返回指定设备的属性信息，此接口将在后续版本中废弃。
<code>mindspore.hal.is_available</code>	查询目标设备是否可用，此接口将在后续版本中废弃，请使用接口 <code>mindspore.device_context.cpu.is_available()</code> 、 <code>mindspore.device_context.gpu.is_available()</code> 、 <code>mindspore.device_context.ascend.is_available()</code> 代替。
<code>mindspore.hal.is_initialized</code>	返回目标设备是否已被初始化，此接口将在后续版本中废弃。

24.1.1 mindspore.hal.device_count

`mindspore.hal.device_count(device_target=None)`

查询目标设备的数量，此接口将在后续版本中废弃，请使用接口 `mindspore.device_context.cpu.device_count()` 、 `mindspore.device_context.gpu.device_count()` 、 `mindspore.device_context.ascend.device_count()` 代替。

说明:

- 对于 "CPU" 设备，固定返回 1。
-

参数:

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU" , "GPU" 以及 "Ascend" 。默认 None，表示当前已经设置的设备。

返回:

int

样例:

```
>>> import mindspore
>>> device_target = mindspore.context.get_context("device_target")
>>> print(mindspore.hal.device_count(device_target))
```

24.1.2 mindspore.hal.get_arch_list

`mindspore.hal.get_arch_list(device_target=None)`

返回此 MindSpore 包支持哪些后端架构，此接口将在后续版本中废弃。

参数:

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU" , "GPU" 以及 "Ascend" 。默认 None，表示当前已经设置的设备。

返回:

GPU 设备返回 str，Ascend 以及 CPU 设备返回 None。

样例:

```
>>> import mindspore
>>> device_target = mindspore.context.get_context("device_target")
>>> print(mindspore.hal.get_arch_list(device_target))
```

24.1.3 mindspore.hal.get_device_capability

`mindspore.hal.get_device_capability(device_id, device_target=None)`

返回指定卡号设备的设备能力，此接口将在后续版本中废弃。

参数：

- **device_id** (int) - 设备 id。
- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

返回：

对于 GPU 设备，返回 tuple(param1, param2)。

- param1: int, cuda major 版本号。
- param2: int, cuda minor 版本号。

对于 CPU 以及 Ascend 设备，返回 None。

样例：

```
>>> import mindspore
>>> device_target = mindspore.context.get_context("device_target")
>>> print(mindspore.hal.get_device_capability(0, device_target))
```

24.1.4 mindspore.hal.get_device_name

`mindspore.hal.get_device_name(device_id, device_target=None)`

返回指定卡号设备的设备名称，此接口将在后续版本中废弃。

说明：

- 对于 CPU 设备，总是返回 "CPU"。

参数：

- **device_id** (int) - 设备 id。
- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

返回：

str

样例：

```
>>> import mindspore
>>> device_target = mindspore.context.get_context("device_target")
>>> print(mindspore.hal.get_device_name(0, device_target))
```

24.1.5 mindspore.hal.get_device_properties

`mindspore.hal.get_device_properties(device_id, device_target=None)`

返回指定设备的属性信息，此接口将在后续版本中废弃。

说明:

- 对于 CPU 设备，固定返回 1。
- 对于 Ascend 设备，必须等待设备初始化完成后，调用此接口才有信息返回，否则属性信息中的 *total_memory* 以及 *free_memory* 都为 0。
- *device_id* 在 Ascend 设备下会被忽略，只返回当前已占用的卡属性。

参数:

- **device_id** (int) - 设备 id。
- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

返回:

- GPU 设备，返回 *cudaDeviceProp* :

```
cudaDeviceProp {
    name(str),
    major(int),
    minor(int),
    is_multi_gpu_board(int),
    is_integrated(int),
    multi_processor_count(int),
    total_memory(int),
    warp_size(int)
}
```

- Ascend 设备，返回 *AscendDeviceProperties* :

```
AscendDeviceProperties {
    name(str),
    total_memory(int),
    free_memory(int)
}
```

- CPU 设备，返回 None。

样例:

```
>>> import mindspore
>>> device_target = mindspore.context.get_context("device_target")
>>> print(mindspore.hal.get_device_properties(0, device_target))
```

24.1.6 mindspore.hal.is_available

`mindspore.hal.is_available(device_target)`

查询目标设备是否可用，此接口将在后续版本中废弃，请使用接口 `mindspore.device_context.cpu.is_available()` 、 `mindspore.device_context.gpu.is_available()` 、 `mindspore.device_context.ascend.is_available()` 代替。

参数：

- **device_target** (str) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。

返回：

bool

样例：

```
>>> import mindspore
>>> device_target = mindspore.context.get_context("device_target")
>>> print(mindspore.hal.is_available(device_target))
True
```

24.1.7 mindspore.hal.is_initialized

`mindspore.hal.is_initialized(device_target)`

返回目标设备是否已被初始化，此接口将在后续版本中废弃。

说明：CPU、GPU 以及 Ascend 设备，分别为在如下场景被初始化：

- 分布式任务中，设备会在调用 `mindspore.communication.init` 后初始化。
- 单卡任务中，会在执行第一个算子或者调用创建流/事件接口后被初始化。

参数：

- **device_target** (str) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。

返回：

bool

样例：

```
>>> import mindspore as ms
>>> import numpy as np
>>> from mindspore import Tensor, ops
>>> ms.context.set_context(device_target="CPU")
>>> assert not ms.hal.is_initialized("CPU")
>>> a = Tensor(np.ones([1, 2]), ms.float32)
>>> b = Tensor(np.ones([1, 2]), ms.float32)
>>> c = ops.add(a, b).asnumpy()
>>> print(ms.hal.is_initialized("CPU"))
True
```

24.2 流管理

<code>mindspore.hal.communication_stream</code>	返回此设备上的通信流，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.communication_stream()</code> 代替。
<code>mindspore.hal.current_stream</code>	返回此设备上正在使用的流，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.current_stream()</code> 代替。
<code>mindspore.hal.default_stream</code>	返回此设备上的默认流，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.default_stream()</code> 代替。
<code>mindspore.hal.set_cur_stream</code>	设置当前流，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.set_cur_stream()</code> 代替。
<code>mindspore.hal.synchronize</code>	同步当前设备上的所有流，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.synchronize()</code> 代替。
<code>mindspore.hal.Stream</code>	基于设备流的封装器，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.Stream</code> 代替。
<code>mindspore.hal.StreamCtx</code>	上下文管理器，用于选择给定的流，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.StreamCtx</code> 代替。

24.2.1 mindspore.hal.communication_stream

`mindspore.hal.communication_stream()`
返回此设备上的通信流，此接口将在后续版本中废弃，请使用接口`mindspore.runtime.communication_stream()`代替。

返回：
Stream，此设备上的通信流。

样例：

```
>>> import mindspore
>>> mindspore.hal.communication_stream()
Stream(device_name=Ascend, device_id:0, stream id:1)
```

24.2.2 mindspore.hal.current_stream

`mindspore.hal.current_stream()`
返回此设备上正在使用的流，此接口将在后续版本中废弃，请使用接口`mindspore.runtime.current_stream()`代替。

返回：
Stream，此设备上正在使用的流。

样例：

```
>>> import mindspore
>>> cur_stream = mindspore.hal.current_stream()
>>> assert cur_stream == mindspore.hal.default_stream()
```

24.2.3 mindspore.hal.default_stream

`mindspore.hal.default_stream()`

返回此设备上的默认流，此接口将在后续版本中废弃，请使用接口`mindspore.runtime.default_stream()`代替。

返回：

Stream，此设备上的默认流。

样例：

```
>>> import mindspore
>>> cur_stream = mindspore.hal.current_stream()
>>> assert cur_stream == mindspore.hal.default_stream()
```

24.2.4 mindspore.hal.set_cur_stream

`mindspore.hal.set_cur_stream(stream)`

设置当前流，此接口将在后续版本中废弃，请使用接口`mindspore.runtime.set_cur_stream()`代替。

参数：

- **stream** (Stream) - 指定的流。如果是 None，这个上下文管理器无操作。

样例：

```
>>> import mindspore
>>> cur_stream = mindspore.hal.current_stream()
>>> assert cur_stream == mindspore.hal.default_stream()
>>> s1 = mindspore.hal.Stream()
>>> mindspore.hal.set_cur_stream(s1)
>>> assert mindspore.hal.current_stream() == s1
>>> mindspore.hal.set_cur_stream(mindspore.hal.default_stream())
```

24.2.5 mindspore.hal.synchronize

`mindspore.hal.synchronize()`

同步当前设备上的所有流，此接口将在后续版本中废弃，请使用接口`mindspore.runtime.synchronize()`代替。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1024, 2048]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2048, 4096]), mindspore.float32)
>>> s1 = mindspore.hal.Stream()
>>> with mindspore.hal.StreamCtx(s1):
...     c = mindspore.ops.matmul(a, b)
```

(下页继续)

(续上页)

```
>>> mindspore.hal.synchronize()
>>> assert s1.query()
```

24.2.6 mindspore.hal.Stream

class mindspore.hal.**Stream** (*priority=0, **kwargs*)

基于设备流的封装器，此接口将在后续版本中废弃，请使用接口 `mindspore.runtime.Stream` 代替。

设备流是属于特定设备的线性执行序列，流之间相互独立。

参数：

- **priority** (int, 可选) - 流的优先级，较低的数字表示较高的优先级。默认情况下，流的优先级为 0。
- **kwargs** (dict) - 关键字参数字典。

query ()

检查所有提交的工作是否已完成。

返回：

bool，指示该流中的所有算子是否已执行完成。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1024, 2048]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2048, 4096]), mindspore.float32)
>>> s1 = mindspore.hal.Stream()
>>> with mindspore.hal.StreamCtx(s1):
...     c = mindspore.ops.matmul(a, b)
>>> s1.synchronize()
>>> assert s1.query()
```

record_event (*event=None*)

记录一个事件。

参数：

- **event** (Event, 可选) - 要记录的事件。如果输入为 None，将分配一个新的事件。默认值：None。

返回：

Event，记录的事件。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([3, 3]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([3, 3]), mindspore.float32)
>>> s1 = mindspore.hal.Stream()
>>> with mindspore.hal.StreamCtx(s1):
...     c = a + b
...     event = s1.record_event()
...     d = a * b
>>> cur_stream = mindspore.hal.current_stream()
>>> cur_stream.wait_event(event)
>>> e = c + 3
>>> print(e)
```

(下页继续)

(续上页)

```
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
```

synchronize()

等待此流中的所有算子执行完成。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1024, 2048]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2048, 4096]), mindspore.float32)
>>> s1 = mindspore.hal.Stream()
>>> with mindspore.hal.StreamCtx(s1):
...     c = mindspore.ops.matmul(a, b)
>>> s1.synchronize()
>>> assert s1.query()
```

wait_event(event)

使提交到流的所有未来工作等待本事件。

参数：

- **event** (Event) - 等待的事件。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([3, 3]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([3, 3]), mindspore.float32)
>>> s1 = mindspore.hal.Stream()
>>> with mindspore.hal.StreamCtx(s1):
...     c = a + b
...     event = s1.record_event()
...     d = a * b
>>> cur_stream = mindspore.hal.current_stream()
>>> cur_stream.wait_event(event)
>>> e = c + 3
>>> print(e)
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
```

wait_stream(stream)

与另一个流同步。

所有提交到本流的未来工作都将等待，直到所有算子都提交至给定流并执行完成。

参数：

- **stream** (Stream) - 需要同步的流。

样例：

```
>>> import mindspore
>>> s1 = mindspore.hal.Stream()
>>> s2 = mindspore.hal.Stream()
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2, 2]), mindspore.float32)
```

(下页继续)

(续上页)

```
>>> with mindspore.hal.StreamCtx(s1):
...     c = mindspore.ops.matmul(a, b)
>>> with mindspore.hal.StreamCtx(s2):
...     s2.wait_stream(s1)
...     d = mindspore.ops.matmul(c, b)
>>> mindspore.hal.synchronize()
>>> print(d)
[[4. 4.]]
```

24.2.7 mindspore.hal.StreamCtx

class mindspore.hal.StreamCtx(*ctx_stream*)

上下文管理器，用于选择给定的流，此接口将在后续版本中废弃，请使用接口 *mindspore.runtime.StreamCtx* 代替。

在上下文范围内，所有算子都将在指定流上执行。

参数：

- **ctx_stream** (Stream) - 指定的流。如果是 *None*，则该上下文管理器无操作。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1024, 2048]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2048, 4096]), mindspore.float32)
>>> s1 = mindspore.hal.Stream()
>>> with mindspore.hal.StreamCtx(s1):
...     c = mindspore.ops.matmul(a, b)
>>> mindspore.hal.synchronize()
>>> assert s1.query()
```

24.3 事件管理

mindspore.hal.Event

设备事件的封装器，此接口将在后续版本中废弃，请使用接口 *mindspore.runtime.Event* 代替。

24.3.1 mindspore.hal.Event

class mindspore.hal.Event(*enable_timing=False, blocking=False*)

设备事件的封装器，此接口将在后续版本中废弃，请使用接口 *mindspore.runtime.Event* 代替。

设备事件是同步标记，可用于监视设备的执行进度、准确计时和同步设备流。

当事件首次被记录时，底层设备事件才会被初始化。

参数：

- **enable_timing** (bool, 可选) - 事件是否应计时。默认 *False*。
- **blocking** (bool, 可选) - 如果为 *True*，*wait* 函数将是阻塞的。默认 *False*。

样例:

```
>>> import mindspore
>>> start = mindspore.hal.Event(enable_timing=True)
>>> end = mindspore.hal.Event(enable_timing=True)
>>> s1 = mindspore.hal.Stream()
>>> s2 = mindspore.hal.Stream()
>>> a = mindspore.tensor(mindspore.ops.ones([2, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2, 2]), mindspore.float32)
>>> c = mindspore.tensor(mindspore.ops.ones([2, 2]), mindspore.float32)
>>> with mindspore.hal.StreamCtx(s1):
...     d = mindspore.ops.matmul(a, b)
...     start.record()
>>> c += 2
>>> end.record()
>>> with mindspore.hal.StreamCtx(s2):
...     start.synchronize()
...     end.synchronize()
...     e = c + d
>>> mindspore.hal.synchronize()
>>> print(e)
[[5. 5.]
 [5. 5.]]
>>> elapsed_time = start.elapsed_time(end)
```

elapsed_time(end_event)

返回记录事件之后到记录 end_event 之前所用的时间（以毫秒为单位）。

参数:

- end_event (Event) - 结束事件。

返回:

float, 经过的时间（以毫秒为单位）。

query()

检查事件当前捕获的所有工作是否已完成。

返回:

bool, 指示事件当前捕获的所有工作是否都已完成。

样例:

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1024, 2048]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2048, 4096]), mindspore.float32)
>>> s1 = mindspore.hal.Stream()
>>> with mindspore.hal.StreamCtx(s1):
...     c = mindspore.ops.matmul(a, b)
...     ev = s1.record_event()
>>> s1.synchronize()
>>> assert ev.query()
```

record(stream=None)

在给定的流中记录事件。

如果未指定 stream, 将使用 `mindspore.hal.current_stream()`。

参数:

- **stream** (Stream, 可选) - 需要记录的流。如果输入为 None，将使用当前流。默认 None。

synchronize()

等待事件完成。

等待直到完成当前此事件捕获的所有工作。这将阻止 CPU 线程继续进行，直到事件完成。

wait (stream=None)

使提交给给定流的所有未来工作等待此事件。

如果未指定 *stream*，将使用`mindspore.hal.current_stream()`。

参数：

- **stream** (Stream, 可选) - 需要等待的流。如果输入为 None，将使用当前流。默认 None。

样例：

```
>>> import mindspore
>>> event = mindspore.hal.Event()
>>> s1 = mindspore.hal.Stream()
>>> s2 = mindspore.hal.Stream()
>>> a = mindspore.tensor(mindspore.ops.ones([2, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([2, 2]), mindspore.float32)
>>> with mindspore.hal.StreamCtx(s1):
...     c = mindspore.ops.matmul(a, b)
...     event.record()
>>> event.wait()
>>> d = c + 2
>>> mindspore.hal.synchronize()
>>> print(d)
[[4. 4.]
 [4. 4.]]
```

24.4 内存管理

<code>mindspore.hal.contiguous_tensors_handle.combine_tensor_list_contiguous</code>	返回一个连续内存管理器，在该内存管理器中，申请了连续内存，并提供切片功能。
<code>mindspore.hal.contiguous_tensors_handle.ContiguousTensorsHandle</code>	连续内存管理器。
<code>mindspore.hal.max_memory_allocated</code>	返回从进程启动开始，内存池真实被 Tensor 占用的内存大小的峰值，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.max_memory_allocated()</code> 代替。
<code>mindspore.hal.max_memory_reserved</code>	返回从进程启动开始，内存池管理的内存总量的峰值，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.max_memory_reserved()</code> 代替。
<code>mindspore.hal.memory_allocated</code>	返回当前真实被 tensor 占用的内存大小，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.memory_allocated()</code> 代替。
<code>mindspore.hal.memory_reserved</code>	返回内存池当前管理的内存总量，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.memory_reserved()</code> 代替。

下页继续

表 4 - 续上页

<code>mindspore.hal.memory_stats</code>	返回从内存池查询到的状态信息，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.memory_stats()</code> 代替。
<code>mindspore.hal.memory_summary</code>	返回可读的内存池状态信息，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.memory_summary()</code> 代替。
<code>mindspore.hal.reset_max_memory_reserved</code>	重置被内存池所管理的内存的峰值，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.reset_max_memory_reserved()</code> 代替。
<code>mindspore.hal.reset_max_memory_allocated</code>	重置内存池真实被 <code>tensor</code> 占用的内存大小的峰值，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.reset_max_memory_allocated()</code> 代替。
<code>mindspore.hal.reset_peak_memory_stats</code>	重置内存池中的全部峰值，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.reset_peak_memory_stats()</code> 代替。
<code>mindspore.hal.empty_cache</code>	清理内存池中的内存碎片，优化内存排布，此接口将在后续版本中废弃，请使用接口 <code>mindspore.runtime.empty_cache()</code> 代替。

24.4.1 mindspore.hal.contiguous_tensors_handle.combine_tensor_list_contiguous

`mindspore.hal.contiguous_tensors_handle.combine_tensor_list_contiguous` (*tensor_list, enable_mem_align=True*)

返回一个连续内存管理器，在该内存管理器中，申请了连续内存，并提供切片功能。

参数：

- `tensor_list` (`list[Tensor]`, `Tuple[Tensor]`) - 需要申请连续内存的 `Tensor` 列表。
- `enable_mem_align` (`bool`, 可选) - 是否启用内存对齐功能。暂不支持 `False`。默认 `True`。

返回：

`ContiguousTensorsHandle`，一个连续内存管理器。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.hal.contiguous_tensors_handle import combine_tensor_list_contiguous
>>> x = Tensor(np.array([1, 2, 3]).astype(np.float32))
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> handle = combine_tensor_list_contiguous([x, y], True)
>>> print(handle[0].shape)
[1]
>>> print(handle[1: 3].asnumpy())
[2, 3]
>>> print(output.slice_by_tensor_index(0, 1).asnumpy())
[1, 2, 3]
```

24.4.2 mindspore.hal.contiguous_tensors_handle.ContiguousTensorsHandle

class mindspore.hal.contiguous_tensors_handle.**ContiguousTensorsHandle** (*tensor_list*,
enable_mem_align=True)

连续内存管理器。

参数：

- **tensor_list** (list[Tensor], Tuple[Tensor]) - 需要申请连续内存的 Tensor 列表。
- **enable_mem_align** (bool, 可选) - 是否启用内存对齐功能。暂不支持 False。默认 True。

返回：

ContiguousTensorsHandle，一个连续内存管理器。

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.hal.contiguous_tensors_handle import ContiguousTensorsHandle
>>> x = Tensor(np.array([1, 2, 3]).astype(np.float32))
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> handle = ContiguousTensorsHandle([x, y], True)
>>> print(handle[0].shape)
[1]
>>> print(handle[1: 3].asnumpy())
[2, 3]
```

slice_by_tensor_index (*start=None*, *end=None*)

返回根据 tensor 列表的 index 切片出的连续内存。

参数：

- **start** (int, None) - 起始位置。默认 None。
- **end** (int, None) - 结束位置。默认 None。

返回：

Tensor

样例：

```
>>> import numpy as np
>>> import mindspore as ms
>>> from mindspore import Tensor
>>> from mindspore.hal.contiguous_tensors_handle import ContiguousTensorsHandle
>>> x = Tensor(np.array([1, 2, 3]).astype(np.float32))
>>> y = Tensor(np.array([4, 5, 6]).astype(np.float32))
>>> handle = ContiguousTensorsHandle([x, y], True)
>>> print(output.slice_by_tensor_index(0, 1).asnumpy())
[1, 2, 3]
```

24.4.3 mindspore.hal.max_memory_allocated

`mindspore.hal.max_memory_allocated(device_target=None)`

返回从进程启动开始，内存池真实被 Tensor 占用的内存大小的峰值，此接口将在后续版本中废弃，请使用接口 `mindspore.runtime.max_memory_allocated()` 代替。

说明:

- 对于 CPU 设备，固定返回 0。

参数:

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

返回:

int，单位为 Byte。

样例:

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
>>> print(mindspore.hal.max_memory_allocated())
1536
```

24.4.4 mindspore.hal.max_memory_reserved

`mindspore.hal.max_memory_reserved(device_target=None)`

返回从进程启动开始，内存池管理的内存总量的峰值，此接口将在后续版本中废弃，请使用接口 `mindspore.runtime.max_memory_reserved()` 代替。

说明:

- 对于 CPU 设备，固定返回 0。

参数:

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

返回:

int，单位为 Byte。

样例:

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
```

(下页继续)

(续上页)

```
>>> print(mindspore.hal.max_memory_reserved())
1073741824
```

24.4.5 mindspore.hal.memory_allocated

`mindspore.hal.memory_allocated(device_target=None)`

返回当前真实被 `tensor` 占用的内存大小，此接口将在后续版本中废弃，请使用接口 `mindspore.runtime.memory_allocated()` 代替。

说明：

- 对于 *CPU* 设备，固定返回 0。

参数：

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 `None`，表示当前已经设置的设备。

返回：

int，单位为 Byte。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
>>> print(mindspore.hal.memory_allocated())
1024
```

24.4.6 mindspore.hal.memory_reserved

`mindspore.hal.memory_reserved(device_target=None)`

返回内存池当前管理的内存总量，此接口将在后续版本中废弃，请使用接口 `mindspore.runtime.memory_reserved()` 代替。

说明：

- 对于 *CPU* 设备，固定返回 0。

参数：

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 `None`，表示当前已经设置的设备。

返回：

int，单位为 Byte。

样例：


```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
>>> print(mindspore.hal.memory_reserved())
1073741824
```

24.4.7 mindspore.hal.memory_stats

`mindspore.hal.memory_stats(device_target=None)`

返回从内存池查询到的状态信息，此接口将在后续版本中废弃，请使用接口 `mindspore.runtime.memory_stats()` 代替。

说明:

- 对于 *CPU* 设备，固定返回数据为空的字典。

参数:

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

返回:

dict, 查询到的内存信息。

样例:

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
>>> print(mindspore.hal.memory_stats())
{'total_reserved_memory': 1073741824, 'total_allocated_memory': 1024, 'total_idle_memory': 1073740800,
'total_eager_free_memory': 0, 'max_reserved_memory': 1073741824, 'max_allocated_memory': 1536,
'common_mem_pool_stats': {'block_unit_size': 1073741824, 'block_counts': 1, 'blocks_info':
{<capsule object NULL at 0x7f7e8c27b030>: {'block_stream_id': 0, 'block_memory_size': 1073741824}}},
'persistent_mem_pool_stats': {'block_unit_size': 1073741824, 'block_counts': 0, 'blocks_info': {}}}
```

24.4.8 mindspore.hal.memory_summary

`mindspore.hal.memory_summary(device_target=None)`

返回可读的内存池状态信息，此接口将在后续版本中废弃，请使用接口 `mindspore.runtime.memory_summary()` 代替。

参数:

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

返回:

str, 表格形式的可读内存池状态信息。

24.4.9 mindspore.hal.reset_max_memory_reserved

`mindspore.hal.reset_max_memory_reserved(device_target=None)`

重置被内存池所管理的内存的峰值, 此接口将在后续版本中废弃, 请使用接口`mindspore.runtime.reset_max_memory_reserved()`代替。

参数:

- **device_target** (str, 可选) - 目标设备, 可选值为 "CPU", "GPU" 以及 "Ascend"。默认 None, 表示当前已经设置的设备。

样例:

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
>>> print(mindspore.hal.max_memory_reserved())
1073741824
>>> mindspore.hal.reset_max_memory_reserved()
>>> print(mindspore.hal.max_memory_reserved())
0
```

24.4.10 mindspore.hal.reset_max_memory_allocated

`mindspore.hal.reset_max_memory_allocated(device_target=None)`

重置内存池真实被 tensor 占用的内存大小的峰值, 此接口将在后续版本中废弃, 请使用接口`mindspore.runtime.reset_max_memory_allocated()`代替。

参数:

- **device_target** (str, 可选) - 目标设备, 可选值为 "CPU", "GPU" 以及 "Ascend"。默认 None, 表示当前已经设置的设备。

样例:

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
>>> print(mindspore.hal.max_memory_allocated())
1536
>>> mindspore.hal.reset_max_memory_allocated()
>>> print(mindspore.hal.max_memory_allocated())
0
```

24.4.11 mindspore.hal.reset_peak_memory_stats

`mindspore.hal.reset_peak_memory_stats(device_target=None)`

重置内存池中的全部峰值，此接口将在后续版本中废弃，请使用接口`mindspore.runtime.reset_peak_memory_stats()`代替。

参数：

- **device_target** (str, 可选) - 目标设备，可选值为 "CPU"，"GPU" 以及 "Ascend"。默认 None，表示当前已经设置的设备。

样例：

```
>>> import mindspore
>>> a = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> b = mindspore.tensor(mindspore.ops.ones([1, 2]), mindspore.float32)
>>> c = mindspore.ops.add(a, b).asnumpy()
>>> print(mindspore.hal.max_memory_reserved())
1073741824
>>> print(mindspore.hal.max_memory_allocated())
1536
>>> mindspore.hal.reset_peak_memory_stats()
>>> print(mindspore.hal.max_memory_reserved())
0
>>> print(mindspore.hal.max_memory_allocated())
0
```

24.4.12 mindspore.hal.empty_cache

`mindspore.hal.empty_cache()`

清理内存池中的内存碎片，优化内存排布，此接口将在后续版本中废弃，请使用接口`mindspore.runtime.empty_cache()`代替。

说明：

- 目前 MindSpore 内存池没有清空内存碎片的功能，此为预留接口，实现为空方法。使用时，会通过日志方式提示。