MindArmour API Reference

Release r1.9

MindSpore

Jan 16, 2023

CHAPTER

ONE

MINDARMOUR

MindArmour 是 MindSpore 的工具箱，用于增强模型可信，实现隐私保护机器学习。

class mindarmour.**Attack**

所有通过创建对抗样本的攻击类的抽象基类。

对抗样本是通过向原始样本添加对抗噪声来生成的。

batch_generate(*inputs*, *labels*, *batch_size=64*)

根据输入样本及其标签来批量生成对抗样本。

参数:

- **inputs** (Union[numpy.ndarray, tuple]) - 生成对抗样本的原始样本。
- **labels** (Union[numpy.ndarray, tuple]) - 原始/目标标签。若每个输入有多个标签，将它包装在元组中。
- **batch_size** (int) - 一个批次中的样本数。默认值：64。

返回:

- **numpy.ndarray** - 生成的对抗样本。

generate(*inputs*, *labels*)

根据正常样本及其标签生成对抗样本。

参数:

- **inputs** (Union[numpy.ndarray, tuple]) - 生成对抗样本的原始样本。
- **labels** (Union[numpy.ndarray, tuple]) - 原始/目标标签。若每个输入有多个标签，将它包装在元组中。

异常:

- **NotImplementedError** - 此为抽象方法。

class mindarmour.**BlackModel**

将目标模型视为黑盒的抽象类。模型应由用户定义。

is_adversarial(*data*, *label*, *is_targeted*)

检查输入样本是否为对抗样本。

参数:

- **data** (numpy.ndarray) - 要检查的输入样本，通常是一些恶意干扰的样本。
- **label** (numpy.ndarray) - 对于目标攻击，标签是受扰动样本的预期标签。对于无目标攻击，标签是相应未扰动样本的原始标签。
- **is_targeted** (bool) - 对于有目标/无目标攻击，请选择 True/False。

返回:

- **bool** - 如果为 True，则输入样本是对抗性的。如果为 False，则输入样本不是对抗性的。

predict(*inputs*)

使用用户指定的模型进行预测。预测结果的 shape 应该是 (m,n)，其中 n 表示此模型分类的类数。

参数:

- **inputs** (numpy.ndarray) - 要预测的输入样本。

异常:

- **NotImplementedError** - 抽象方法未实现。

class mindarmour.**Detector**

所有对抗样本检测器的抽象基类。

detect(*inputs*)

从输入样本中检测对抗样本。

参数:

- **inputs** (Union[numpy.ndarray, list, tuple]) - 要检测的输入样本。

异常:

- **NotImplementedError** - 抽象方法未实现。

detect_diff(*inputs*)

计算输入样本和去噪样本之间的差值。

参数：

- **inputs** (Union[numpy.ndarray, list, tuple]) - 要检测的输入样本。

异常：

- **NotImplementedError** - 抽象方法未实现。

fit(*inputs*, *labels=None*)

拟合阈值，拒绝与去噪样本差异大于阈值的对抗样本。当应用于正常样本时，阈值由假正率决定。

参数：

- **inputs** (numpy.ndarray) - 用于计算阈值的输入样本。
- **labels** (numpy.ndarray) - 训练数据的标签。默认值：None。

异常：

- **NotImplementedError** - 抽象方法未实现。

transform(*inputs*)

过滤输入样本中的对抗性噪声。

参数：

- **inputs** (Union[numpy.ndarray, list, tuple]) - 要转换的输入样本。

异常：

- **NotImplementedError** - 抽象方法未实现。

class mindarmour.**Defense**(*network*)

所有防御类的抽象基类，用于防御对抗样本。

参数：

- **network** (Cell) - 要防御的 MindSpore 风格的深度学习模型。

batch_defense(*inputs*, *labels*, *batch_size=32*, *epochs=5*)

对输入进行批量防御操作。

参数：

- **inputs** (numpy.ndarray) - 生成对抗样本的原始样本。
- **labels** (numpy.ndarray) - 输入样本的标签。
- **batch_size** (int) - 一个批次中的样本数。默认值：32。
- **epochs** (int) - epochs 的数量。默认值：5。

返回：

- **numpy.ndarray** - *batch_defense* 操作的损失。

异常：

- **ValueError** - *batch_size* 为 0。

defense(*inputs*, *labels*)

对输入进行防御操作。

参数：

- **inputs** (numpy.ndarray) - 生成对抗样本的原始样本。
- **labels** (numpy.ndarray) - 输入样本的标签。

异常：

- **NotImplementedError** - 抽象方法未实现。

class mindarmour.**Fuzzer**(*target_model*)

深度神经网络的模糊测试框架。

参考文献：DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks。

参数：

- **target_model** (Model) - 目标模糊模型。

样例：

```
>>> from mindspore.common.initializer import TruncatedNormal
>>> from mindspore.ops import operations as P
>>> from mindspore.train import Model
>>> from mindspore.ops import TensorSummary
```

(continues on next page)

(continued from previous page)

```
>>> from mindarmour.fuzz_testing import Fuzzer
>>> from mindarmour.fuzz_testing import KMultisectionNeuronCoverage
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(1, 6, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.conv2 = nn.Conv2d(6, 16, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.fc1 = nn.Dense(16 * 5 * 5, 120, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc2 = nn.Dense(120, 84, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc3 = nn.Dense(84, 10, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.reshape = P.Reshape()
...         self.summary = TensorSummary()
...
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu(x)
...         self.summary('conv1', x)
...         x = self.max_pool2d(x)
...         x = self.conv2(x)
...         x = self.relu(x)
...         self.summary('conv2', x)
...         x = self.max_pool2d(x)
...         x = self.reshape(x, (-1, 16 * 5 * 5))
...         x = self.fc1(x)
...         x = self.relu(x)
...         self.summary('fc1', x)
...         x = self.fc2(x)
...         x = self.relu(x)
...         self.summary('fc2', x)
...         x = self.fc3(x)
...         self.summary('fc3', x)
...         return x
>>> net = Net()
>>> model = Model(net)
>>> mutate_config = [{'method': 'GaussianBlur',
...                   'params': {'ksize': [1, 2, 3, 5], 'auto_param': [True, False]}},
...                  {'method': 'MotionBlur',
...                   'params': {'degree': [1, 2, 5], 'angle': [45, 10, 100, 140, 210, 270, 300],
...                   'auto_param': [True]}},
...                  {'method': 'UniformNoise',
...                   'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
...                  {'method': 'GaussianNoise',
...                   'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
...                  {'method': 'Contrast',
...                   'params': {'alpha': [0.5, 1, 1.5], 'beta': [-10, 0, 10], 'auto_param': [False, True]}},
...                  {'method': 'Rotate',
...                   'params': {'angle': [20, 90], 'auto_param': [False, True]}},
...                  {'method': 'FGSM',
...                   'params': {'eps': [0.3, 0.2, 0.4], 'alpha': [0.1], 'bounds': [(0, 1)]}}]
>>> batch_size = 8
>>> num_classe = 10
>>> train_images = np.random.rand(32, 1, 32, 32).astype(np.float32)
>>> test_images = np.random.rand(batch_size, 1, 32, 32).astype(np.float32)
>>> test_labels = np.random.randint(num_classe, size=batch_size).astype(np.int32)
>>> test_labels = (np.eye(num_classe)[test_labels]).astype(np.float32)
>>> initial_seeds = []
>>> # make initial seeds
>>> for img, label in zip(test_images, test_labels):
...     initial_seeds.append([img, label])
>>> initial_seeds = initial_seeds[:10]
>>> nc = KMultisectionNeuronCoverage(model, train_images, segmented_num=100, incremental=True)
>>> model_fuzz_test = Fuzzer(model)
>>> samples, gt_labels, preds, strategies, metrics = model_fuzz_test.fuzzing(mutate_config, initial_seeds,
...                                                                          nc, max_iters=100)
```

fuzzing(*mutate_config*, *initial_seeds*, *coverage*, *evaluate=True*, *max_iters=10000*, *mutate_num_per_seed=20*)

深度神经网络的模糊测试。

参数：

- **mutate_config (list) - 变异方法配置。格式为：**

```
mutate_config =
    [{'method': 'GaussianBlur',
      'params': {'ksize': [1, 2, 3, 5], 'auto_param': [True, False]}},
     {'method': 'UniformNoise',
      'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
     {'method': 'GaussianNoise',
      'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
     {'method': 'Contrast',
      'params': {'alpha': [0.5, 1, 1.5], 'beta': [-10, 0, 10], 'auto_param': [False, True]}},
     {'method': 'Rotate',
      'params': {'angle': [20, 90], 'auto_param': [False, True]}},
     {'method': 'FGSM',
      'params': {'eps': [0.3, 0.2, 0.4], 'alpha': [0.1], 'bounds': [(0, 1)]}}]
    ...]
```

– 支持的方法在列表 *self._strategies* 中，每个方法的参数必须在可选参数的范围内。支持的方法分为两种类型：

– 首先，自然鲁棒性方法包括：’Translate’，‘Scale’、’Shear’、’Rotate’、’Perspective’、’Curve’、’GaussianBlur’、’MotionBlur’、’GradientBlur’、’Contrast’、’GradientLuminance’、’UniformNoise’、’GaussianNoise’、’SaltAndPepperNoise’、’NaturalNoise’。

– 其次，对抗样本攻击方式包括：’FGSM’、’PGD’和’MDIM’。’FGSM’、’PGD’和’MDIM’分别是 FastGradientSignMethod、ProjectedGradientDent 和 MomentumDiverseInputIterativeMethod 的缩写。*mutate_config* 必须包含在 [‘Contrast’，‘GradientLuminance’，‘GaussianBlur’，‘MotionBlur’，‘GradientBlur’，‘UniformNoise’，‘GaussianNoise’，‘SaltAndPepperNoise’，‘NaturalNoise’] 中的方法。

– 第一类方法的参数设置方式可以在 mindarmour/natural_robustness/transform/image 中看到。第二类方法参数配置参考 *self._attack_param_checklists* 。

- **initial_seeds** (list[list]) - 用于生成变异样本的初始种子队列。初始种子队列的格式为 [[image_data, label], […], …]，且标签必须为 one-hot。
- **coverage** (CoverageMetrics) - 神经元覆盖率指标类。
- **evaluate** (bool) - 是否返回评估报告。默认值：True。
- **max_iters** (int) - 选择要变异的种子的最大数量。默认值：10000。
- **mutate_num_per_seed** (int) - 每个种子的最大变异次数。默认值：20。

返回：

- **list** - 模糊测试生成的变异样本。
- **list** - 变异样本的 ground truth 标签。
- **list** - 预测结果。
- **list** - 变异策略。
- **dict** - Fuzzer 的指标报告。

异常：

- **ValueError** - 参数’Coverage’必须是 CoverageMetrics 的子类。
- **ValueError** - 初始种子队列为空。
- **ValueError** - 初始种子队列中的种子不是包含两个元素。

class mindarmour.**DPModel**(*micro_batches=2*, *norm_bound=1.0*, *noise_mech=None*, *clip_mech=None*, *optimizer=nn.Momentum*, ***kwargs*)

DPModel 用于构建差分隐私训练的模型。

此类重载 `mindspore.Model`。

详情请查看：应用差分隐私机制保护用户隐私。

参数：

- **micro_batches** (int) - 从原始批次拆分的小批次数。默认值：2。
- **norm_bound** (float) - 用于裁剪的约束，如果设置为 1，将返回原始数据。默认值：1.0。
- **noise_mech** (Mechanisms) - 用于生成不同类型的噪音。默认值：None。
- **clip_mech** (Mechanisms) - 用于更新自适应剪裁。默认值：None。
- **optimizer** (Cell) - 用于更新差分隐私训练过程中的模型权重值。默认值：nn.Momentum。

异常：

- **ValueError** - *optimizer* 值为 None。
- **ValueError** - *optimizer* 不是 DPOptimizer，且 *noise_mech* 为 None。
- **ValueError** - *optimizer* 是 DPOptimizer，且 *noise_mech* 非 None。
- **ValueError** - *noise_mech* 或 DPOptimizer 的 mech 方法是自适应的，而 *clip_mech* 不是 None。

class mindarmour.**MembershipInference**(*model*, *n_jobs=-1*)

成员推理是由 Shokri、Stronati、Song 和 Shmatikov 提出的一种用于推测用户隐私数据的灰盒攻击。它需要训练样本的 loss 或 logits 结果，隐私是指单个用户的一些敏感属性。

有关详细信息，请参见：使用成员推理测试模型安全性。

参考文献：Reza Shokri, Marco Stronati, Congzheng Song, Vitaly Shmatikov. Membership Inference Attacks against Machine Learning Models. 2017.。

参数：

- **model** (Model) - 目标模型。
- **n_jobs** (int) - 并行运行的任务数量。-1 表示使用所有处理器，否则 n_jobs 的值必须为正整数。

异常：

- **TypeError** - 模型的类型不是 Mindspore.Model。
- **TypeError** - *n_jobs* 的类型不是 int。
- **ValueError** - *n_jobs* 的值既不是-1，也不是正整数。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindspore.nn import Cell
>>> from mindspore import Model
>>> from mindarmour.privacy.evaluation import MembershipInference
>>> def dataset_generator():
...     batch_size = 16
...     batches = 1
...     data =  np.random.randn(batches * batch_size,1,10).astype(np.float32)
...     label =  np.random.randint(0,10, batches * batch_size).astype(np.int32)
...     for i in range(batches):
...         yield data[i*batch_size:(i+1)*batch_size], label[i*batch_size:(i+1)*batch_size]
>>> class Net(Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         return self._squeeze(out)
>>> net = Net()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> opt = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(network=net, loss_fn=loss, optimizer=opt)
>>> inference_model = MembershipInference(model, 2)
>>> config = [{
...     "method": "KNN",
...     "params": {"n_neighbors": [3, 5, 7],}
...     }]
>>> ds_train = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> ds_test = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> inference_model.train(ds_train, ds_test, config)
>>> metrics = ["precision", "accuracy", "recall"]
>>> eval_train = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> eval_test = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> result = inference_model.eval(eval_train. eval_test, metrics)
>>> print(result)
```

eval(*dataset_train*, *dataset_test*, *metrics*)

评估目标模型的不同隐私。评估指标应由 metrics 规定。

参数：

- **dataset_train** (mindspore.dataset) - 目标模型的训练数据集。
- **dataset_test** (mindspore.dataset) - 目标模型的测试数据集。
- **metrics** (Union[list, tuple]) - 评估指标。指标的值必须在 [“precision”，“accuracy”，“recall”] 中。默认值：[“precision”]。

返回：

- **list** - 每个元素都包含攻击模型的评估指标。

train(*dataset_train*, *dataset_test*, *attack_config*)

根据配置，使用输入数据集训练攻击模型。

参数：

- **dataset_train** (mindspore.dataset) - 目标模型的训练数据集。
- **dataset_test** (mindspore.dataset) - 目标模型的测试集。
- **attack_config** (Union[list, tuple]) - 攻击模型的参数设置。格式为：

```
attack_config = [
    {"method": "knn", "params": {"n_neighbors": [3, 5, 7]}},
     {"method": "lr", "params": {"C": np.logspace(-4, 2, 10)}}]
```

  - 支持的方法有 knn、lr、mlp 和 rf，每个方法的参数必须在可变参数的范围内。参数实现的提示可在下面找到：
    - KNN
    - LR
    - RF
    - MLP

异常：

- **KeyError** - *attack_config* 中的配置没有键 {“method”，“params”}。
- **NameError** - *attack_config* 中的方法（不区分大小写）不在 [“lr”，“knn”，“rf”，“mlp”] 中。

class mindarmour.**ImageInversionAttack**(*network*, *input_shape*, *input_bound*, *loss_weights=(1, 0.2, 5)*)

一种通过还原图像的深层表达来重建图像的攻击方法。

参考文献：Aravindh Mahendran, Andrea Vedaldi. Understanding Deep Image Representations by Inverting Them. 2014.。

参数：

- **network** (Cell) - 网络，用于推断图像的深层特征。
- **input_shape** (tuple) - 单个网络输入的数据形状，应与给定网络一致。形状的格式应为 (channel, image_width, image_height)。
- **input_bound** (Union[tuple, list]) - 原始图像的像素范围，应该像 [minimum_pixel, maximum_pixel] 或 (minimum_pixel, maximum_pixel)。
- **loss_weights** (Union[list, tuple]) - InversionLoss 中三个子损失的权重，可以调整以获得更好的结果。默认值：(1, 0.2, 5)。

异常：

- **TypeError** - 网络类型不是 Cell。
- **ValueError** - *input_shape* 的值有非正整数。
- **ValueError** - *loss_weights* 的值有非正数。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindspore.nn import Cell
>>> from mindarmour.privacy.evaluation.inversion_attack import ImageInversionAttack
>>> class Net(Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._reduce = P.ReduceSum()
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._reduce(out, 2)
...         return self._squeeze(out)
>>> net = Net()
>>> original_images = np.random.random((2,1,10,10)).astype(np.float32)
>>> target_features =  np.random.random((2,10)).astype(np.float32)
>>> inversion_attack = ImageInversionAttack(net,
...                                         input_shape=(1, 10, 10),
...                                         input_bound=(0, 1),
...                                         loss_weights=[1, 0.2, 5])
>>> inversion_images = inversion_attack.generate(target_features, iters=10)
>>> evaluate_result = inversion_attack.evaluate(original_images, inversion_images)
```

evaluate(*original_images*, *inversion_images*, *labels=None*, *new_network=None*)

通过三个指标评估还原图像的质量：原始图像和还原图像之间的平均 L2 距离和 SSIM 值，以及新模型对还原图像的推理结果在真实标签上的置信度平均值。

参数：

- **original_images** (numpy.ndarray) - 原始图像，其形状应为 (img_num, channels, img_width, img_height)。
- **inversion_images** (numpy.ndarray) - 还原图像，其形状应为 (img_num, channels, img_width, img_height)。
- **labels** (numpy.ndarray) - 原始图像的 ground truth 标签。默认值：None。

- **new_network** (Cell) - 其结构包含 self._network 中所有网络，但加载了不同的模型文件。默认值：None。

返回：

- **float** - l2 距离。
- **float** - 平均 ssim 值。
- **Union** [float, None] - 平均置信度。如果 labels 或 new_network 为 None，则该值为 None。

generate(*target_features*, *iters=100*)

根据 *target_features* 重建图像。

参数：

- **target_features** (numpy.ndarray) - 原始图像的深度表示。*target_features* 的第一个维度应该是 img_num。需要注意的是，如果 img_num 等于 1，则 *target_features* 的形状应该是 (1, dim2, dim3, ···)。
- **iters** (int) - 逆向攻击的迭代次数，应为正整数。默认值：100。

返回：

- **numpy.ndarray** - 重建图像，预计与原始图像相似。

异常：

- **TypeError** - target_features 的类型不是 numpy.ndarray。
- **ValueError** - *iters* 的有非正整数.

class mindarmour.**ConceptDriftCheckTimeSeries**(*window_size=100*, *rolling_window=10*, *step=10*, *threshold_index=1.5*, *need_label=False*)

概念漂移检查时间序列（ConceptDriftCheckTimeSeries）用于样本序列分布变化检测。

有关详细信息，请查看：实现时序数据概念漂移检测应用。

参数：

- **window_size** (int) - 概念窗口的大小，不小于 10。如果给定输入数据，window_size 在 [10, 1/3*len(input data)] 中。如果数据是周期性的，通常 window_size 等于 2-5 个周期，例如，对于月/周数据，30/7 天的数据量是一个周期。默认值：100。
- **rolling_window** (int) - 平滑窗口大小，在 [1, window_size] 中。默认值：10。
- **step** (int) - 滑动窗口的跳跃长度，在 [1, window_size] 中。默认值：10。
- **threshold_index** (float) - 阈值索引，$(-\infty, +\infty)$。默认值：1.5。
- **need_label** (bool) - False 或 True。如果 need_label=True，则需要概念漂移标签。默认值：False。

样例：

```
>>> from mindarmour import ConceptDriftCheckTimeSeries
>>> concept = ConceptDriftCheckTimeSeries(window_size=100, rolling_window=10,
...                                       step=10, threshold_index=1.5, need_label=False)
>>> data_example = 5*np.random.rand(1000)
>>> data_example[200: 800] = 20*np.random.rand(600)
>>> score, threshold, concept_drift_location = concept.concept_check(data_example)
```

concept_check(*data*)

在数据序列中查找概念漂移位置。

参数：

- **data** (numpy.ndarray) - 输入数据。数据的 shape 可以是 (n,1) 或 (n,m)。请注意，每列（m 列）是一个数据序列。

返回：

- **numpy.ndarray** - 样本序列的概念漂移分数。
- **float** - 判断概念漂移的阈值。
- **list** - 概念漂移的位置。

CHAPTER

TWO

MINDARMOUR.ADV_ROBUSTNESS.ATTACKS

本模块包括经典的黑盒和白盒攻击算法，以制作对抗样本。

class mindarmour.adv_robustness.attacks.**FastGradientMethod**(*network*, *eps=0.07*, *alpha=None*, *bounds=(0.0*, *1.0)*, *norm_level=2*, *is_targeted=False*, *loss_fn=None*)

基于梯度计算的单步攻击，扰动的范数包括‘L1’、’L2’和’Linf’。

参考文献：I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in ICLR, 2015.。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.07。
- **alpha** (Union[float, None]) - 单步随机扰动与数据范围的比例。默认值：None。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值, 数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **norm_level** (Union[int, str, numpy.inf]) - 范数类型。可取值：numpy.inf、1、2、’1’、’2’、’l1’、’l2’、’np.inf’、’inf’、’linf’。默认值：2。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **loss_fn** (Union[loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindarmour.adv_robustness.attacks import FastGradientMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> attack = FastGradientMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> adv_x = attack.generate(inputs, labels)
```

class mindarmour.adv_robustness.attacks.**RandomFastGradientMethod**(*network*, *eps=0.07*, *alpha=0.035*, *bounds=(0.0*, *1.0)*, *norm_level=2*, *is_targeted=False*, *loss_fn=None*)

使用随机扰动的快速梯度法（Fast Gradient Method）。基于梯度计算的单步攻击，其对抗性噪声是根据输入的梯度生成的，然后加入随机扰动，从而生成对抗样本。

参考文献：Florian Tramer, Alexey Kurakin, Nicolas Papernot, “Ensemble adversarial training: Attacks and defenses” in ICLR, 2018。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.07。
- **alpha** (float) - 单步随机扰动与数据范围的比例。默认值：0.035。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **norm_level** (Union[int, str, numpy.inf]) - 范数类型。可取值：numpy.inf、1、2、’1’、’2’、’l1’、’l2’、’np.inf’、’inf’、’linf’。默认值：2。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **loss_fn** (Union[loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

异常：

- **ValueError** - *eps* 小于 *alpha* 。

样例：

```
>>> from mindarmour.adv_robustness.attacks import RandomFastGradientMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> net = Net()
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> attack = RandomFastGradientMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> adv_x = attack.generate(inputs, labels)
```

class mindarmour.adv_robustness.attacks.**FastGradientSignMethod**(*network*, *eps=0.07*, *alpha=None*, *bounds=(0.0, 1.0)*, *is_targeted=False*, *loss_fn=None*)

快速梯度下降法（Fast Gradient Sign Method）攻击计算输入数据的梯度，然后使用梯度的符号创建对抗性噪声。

参考文献：Ian J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in ICLR, 2015。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.07。
- **alpha** (Union[float, None]) - 单步随机扰动与数据范围的比例。默认值：None。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindarmour.adv_robustness.attacks import FastGradientSignMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> net = Net()
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> attack = FastGradientSignMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> adv_x = attack.generate(inputs, labels)
```

class mindarmour.adv_robustness.attacks.**RandomFastGradientSignMethod**(*network*, *eps=0.07*, *alpha=0.035*, *bounds=(0.0, 1.0)*, *is_targeted=False*, *loss_fn=None*)

快速梯度下降法（Fast Gradient Sign Method）使用随机扰动。随机快速梯度符号法（Random Fast Gradient Sign Method）攻击计算输入数据的梯度，然后使用带有随机扰动的梯度符号来创建对抗性噪声。

参考文献：F. Tramer, et al., "Ensemble adversarial training: Attacks and defenses," in ICLR, 2018。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.07。
- **alpha** (float) - 单步随机扰动与数据范围的比例。默认值：0.005。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

异常：

- **ValueError** - *eps* 小于 *alpha* 。

样例：

```
>>> from mindarmour.adv_robustness.attacks import RandomFastGradientSignMethod
>>> class Net(nn.Cell):
...     def __init__(self):
```

(continues on next page)

(continued from previous page)

```
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> net = Net()
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> attack = RandomFastGradientSignMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> adv_x = attack.generate(inputs, labels)
```

class mindarmour.adv_robustness.attacks.**LeastLikelyClassMethod**(*network*, *eps=0.07*, *alpha=None*, *bounds=(0.0*, *1.0)*, *loss_fn=None*)

单步最不可能类方法（Single Step Least-Likely Class Method）是 FGSM 的变体，它以最不可能类为目标，以生成对抗样本。

参考文献：F. Tramer, et al., "Ensemble adversarial training: Attacks and defenses," in ICLR, 2018。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.07。
- **alpha** (Union[float, None]) - 单步随机扰动与数据范围的比例。默认值：None。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindarmour.adv_robustness.attacks import LeastLikelyClassMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> attack = LeastLikelyClassMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> adv_x = attack.generate(inputs, labels)
```

class mindarmour.adv_robustness.attacks.**RandomLeastLikelyClassMethod**(*network*, *eps=0.07*, *alpha=0.035*, *bounds=(0.0*, *1.0)*, *loss_fn=None*)

随机最不可能类攻击方法：以置信度最小类别对应的梯度加一个随机扰动为攻击方向。

具有随机扰动的单步最不可能类方法（Single Step Least-Likely Class Method）是随机 FGSM 的变体，它以最不可能类为目标，以生成对抗样本。

参考文献：F. Tramer, et al., "Ensemble adversarial training: Attacks and defenses," in ICLR, 2018。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.07。
- **alpha** (float) - 单步随机扰动与数据范围的比例。默认值：0.005。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

异常：

- **ValueError** - *eps* 小于 *alpha* 。

样例：

```
>>> from mindarmour.adv_robustness.attacks import RandomLeastLikelyClassMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
```

(continues on next page)

(continued from previous page)

```
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> attack = RandomLeastLikelyClassMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> adv_x = attack.generate(inputs, labels)
```

class mindarmour.adv_robustness.attacks.**IterativeGradientMethod**(*network*, *eps=0.3*, *eps_iter=0.1*, *bounds=(0.0*, *1.0)*, *nb_iter=5*, *loss_fn=None*)

所有基于迭代梯度的攻击的抽象基类。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的对抗性扰动占数据范围的比例。默认值：0.3。
- **eps_iter** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.1。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **nb_iter** (int) - 迭代次数。默认值：5。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

generate(*inputs*, *labels*)

根据输入样本和原始/目标标签生成对抗样本。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 良性输入样本，用于创建对抗样本。
- **labels** (Union[numpy.ndarray, tuple]) - 原始/目标标签。若每个输入有多个标签，将它包装在元组中。

异常：

- **NotImplementedError** - 此函数在迭代梯度方法中不可用。

class mindarmour.adv_robustness.attacks.**BasicIterativeMethod**(*network*, *eps=0.3*, *eps_iter=0.1*, *bounds=(0.0*, *1.0)*, *is_targeted=False*, *nb_iter=5*, *loss_fn=None*)

基本迭代法（Basic Iterative Method）攻击，一种生成对抗示例的迭代 FGSM 方法。

参考文献：A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” in ICLR, 2017。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的对抗性扰动占数据范围的比例。默认值：0.3。
- **eps_iter** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.1。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **nb_iter** (int) - 迭代次数。默认值：5。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindspore.ops import operations as P
>>> from mindarmour.adv_robustness.attacks import BasicIterativeMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> attack = BasicIterativeMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> adv_x = attack.generate(inputs, labels)
```

generate(*inputs*, *labels*)

使用迭代 FGSM 方法生成对抗样本。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 良性输入样本，用于创建对抗样本。
- **labels** (Union[numpy.ndarray, tuple]) - 原始/目标标签。若每个输入有多个标签，将它包装在元组中。

返回：

- **numpy.ndarray** - 生成的对抗样本。

class mindarmour.adv_robustness.attacks.**MomentumIterativeMethod**(*network*, *eps=0.3*, *eps_iter=0.1*, *bounds=(0.0, 1.0)*, *is_targeted=False*, *nb_iter=5*, *decay_factor=1.0*, *norm_level='inf'*, *loss_fn=None*)

动量迭代法（Momentum Iterative Method）攻击，通过在迭代中积累损失函数的梯度方向上的速度矢量，加速梯度下降算法，如 FGSM、FGM 和 LLCM，从而生成对抗样本。

参考文献：Y. Dong, et al., "Boosting adversarial attacks with momentum," arXiv:1710.06081, 2017。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的对抗性扰动占数据范围的比例。默认值：0.3。
- **eps_iter** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.1。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **nb_iter** (int) - 迭代次数。默认值：5。
- **decay_factor** (float) - 迭代中的衰变因子。默认值：1.0。
- **norm_level** (Union[int, str, numpy.inf]) - 范数类型。可取值：numpy.inf、1、2、'1'、'2'、'l1'、'l2'、'np.inf'、'inf'、'linf'。默认值：numpy.inf。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindspore.ops import operations as P
>>> from mindarmour.adv_robustness.attacks import MomentumIterativeMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> attack = MomentumIterativeMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> adv_x = attack.generate(inputs, labels)
```

generate(*inputs*, *labels*)

根据输入数据和原始/目标标签生成对抗样本。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 良性输入样本，用于创建对抗样本。
- **labels** (Union[numpy.ndarray, tuple]) - 原始/目标标签。若每个输入有多个标签，将它包装在元组中。

返回：

- **numpy.ndarray** - 生成的对抗样本。

class mindarmour.adv_robustness.attacks.**ProjectedGradientDescent**(*network*, *eps=0.3*, *eps_iter=0.1*, *bounds=(0.0, 1.0)*, *is_targeted=False*, *nb_iter=5*, *norm_level='inf'*, *loss_fn=None*)

投影梯度下降（Projected Gradient Descent）攻击是基本迭代法的变体，在这种方法中，每次迭代之后，扰动被投影在指定半径的 p 范数球上（除了剪切对抗样本的值，使其位于允许的数据范围内）。这是 Madry 等人提出的用于对抗性训练的攻击。

参考文献：A. Madry, et al., "Towards deep learning models resistant to adversarial attacks," in ICLR, 2018。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的对抗性扰动占数据范围的比例。默认值：0.3。
- **eps_iter** (float) - 攻击产生的单步对抗扰动占数据范围的比例。默认值：0.1。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **nb_iter** (int) - 迭代次数。默认值：5。
- **norm_level** (Union[int, str, numpy.inf]) - 范数类型。可取值：numpy.inf、1、2、'1'、'2'、'l1'、'l2'、'np.inf'、'inf'、'linf'。默认值：'numpy.inf'。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindspore.ops import operations as P
>>> from mindarmour.adv_robustness.attacks import ProjectedGradientDescent
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> attack = ProjectedGradientDescent(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> adv_x = attack.generate(inputs, labels)
```

generate(*inputs*, *labels*)

基于 BIM 方法迭代生成对抗样本。通过带有参数 norm_level 的投影方法归一化扰动。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 良性输入样本，用于创建对抗样本。
- **labels** (Union[numpy.ndarray, tuple]) - 原始/目标标签。若每个输入有多个标签，将它包装在元组中。

返回：

- **numpy.ndarray** - 生成的对抗样本。

class mindarmour.adv_robustness.attacks.**DiverseInputIterativeMethod**(*network*, *eps=0.3*, *bounds=(0.0*, *1.0)*, *is_targeted=False*, *prob=0.5*, *loss_fn=None*)

多样性输入迭代法（Diverse Input Iterative Method）攻击遵循基本迭代法，并在每次迭代时对输入数据应用随机转换。对输入数据的这种转换可以提高对抗样本的可转移性。

参考文献：Xie, Cihang and Zhang, et al., "Improving Transferability of Adversarial Examples With Input Diversity," in CVPR, 2019。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的对抗性扰动占数据范围的比例。默认值：0.3。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **prob** (float) - 对输入样本的转换概率。默认值：0.5。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindspore.ops import operations as P
>>> from mindarmour.adv_robustness.attacks import DiverseInputIterativeMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> attack = DiverseInputIterativeMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> adv_x = attack.generate(inputs, labels)
```

generate(*inputs*, *labels*)

基于多样性输入迭代法生成对抗样本。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 良性输入样本，用于创建对抗样本。
- **labels** (Union[numpy.ndarray, tuple]) - 原始/目标标签。若每个输入有多个标签，将它包装在元组中。

返回：

- **numpy.ndarray** - 生成的对抗样本。

class mindarmour.adv_robustness.attacks.**MomentumDiverseInputIterativeMethod**(*network*, *eps=0.3*, *bounds=(0.0, 1.0)*, *is_targeted=False*, *norm_level='l1'*, *prob=0.5*, *loss_fn=None*)

动量多样性输入迭代法（Momentum Diverse Input Iterative Method）攻击是一种动量迭代法，在每次迭代时对输入数据应用随机变换。对输入数据的这种转换可以提高对抗样本的可转移性。

参考文献：Xie, Cihang and Zhang, et al., "Improving Transferability of Adversarial Examples With Input Diversity," in CVPR, 2019。

参数：

- **network** (Cell) - 目标模型。
- **eps** (float) - 攻击产生的对抗性扰动占数据范围的比例。默认值：0.3。
- **bounds** (tuple) - 数据的上下界，表示数据范围。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **norm_level** (Union[int, str, numpy.inf]) - 范数类型。可取值：numpy.inf、1、2、'1'、'2'、'l1'、'l2'、'np.inf'、'inf'、'linf'。默认值：'l1'。
- **prob** (float) - 对输入样本的转换概率。默认值：0.5。
- **loss_fn** (Union[Loss, None]) - 用于优化的损失函数。如果为 None，则输入网络已配备损失函数。默认值：None。

样例：

```
>>> from mindspore.ops import operations as P
>>> from mindarmour.adv_robustness.attacks import MomentumDiverseInputIterativeMethod
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> attack = MomentumDiverseInputIterativeMethod(net, loss_fn=nn.SoftmaxCrossEntropyWithLogits(sparse=False))
>>> inputs = np.asarray([[0.1, 0.2, 0.7]], np.float32)
>>> labels = np.asarray([2],np.int32)
>>> labels = np.eye(3)[labels].astype(np.float32)
>>> net = Net()
>>> adv_x = attack.generate(inputs, labels)
```

class mindarmour.adv_robustness.attacks.**DeepFool**(*network*, *num_classes*, *model_type='classification'*, *reserve_ratio=0.3*, *max_iters=50*, *overshoot=0.02*, *norm_level=2*, *bounds=None*, *sparse=True*)

DeepFool 是一种无目标的迭代攻击，通过将良性样本移动到最近的分类边界并跨越边界来实现。

参考文献：DeepFool: a simple and accurate method to fool deep neural networks。

参数：

- **network** (Cell) - 目标模型。
- **num_classes** (int) - 模型输出的标签数，应大于零。
- **model_type** (str) - 目标模型的类型。现在支持'classification'和'detection'。默认值：'classification'。
- **reserve_ratio** (Union[int, float]) - 攻击后可检测到的对象百分比，仅当 model_type='detection'时有效。保留比率应在 (0, 1) 的范围内。默认值：0.3。
- **max_iters** (int) - 最大迭代次数，应大于零。默认值：50。
- **overshoot** (float) - 过冲参数。默认值：0.02。
- **norm_level** (Union[int, str, numpy.inf]) - 矢量范数类型。可取值：numpy.inf 或 2。默认值：2。
- **bounds** (Union[tuple, list]) - 数据范围的上下界。以 (数据最小值，数据最大值) 的形式出现。默认值：None。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindspore import Tensor
>>> from mindarmour.adv_robustness.attacks import DeepFool
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> input_shape = (1, 5)
>>> _, classes = input_shape
>>> attack = DeepFool(net, classes, max_iters=10, norm_level=2,
```

(continues on next page)

(continued from previous page)

```
...                     bounds=(0.0, 1.0))
>>> input_np = np.array([[0.1, 0.2, 0.7, 0.5, 0.4]]).astype(np.float32)
>>> input_me = Tensor(input_np)
>>> true_labels = np.argmax(net(input_me).asnumpy(), axis=1)
>>> advs = attack.generate(input_np, true_labels)
```

generate(*inputs*, *labels*)

根据输入样本和原始标签生成对抗样本。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 输入样本。
  - 如果 *model_type* =’classification’，则输入的格式应为 numpy.ndarray。输入的格式可以是 (input1, input2, …)。
  - 如果 *model_type* =’detection’，则只能是一个数组。
- **labels** (Union[numpy.ndarray, tuple]) - 目标标签或 ground-truth 标签。
  - 如果 *model_type* =’classification’，标签的格式应为 numpy.ndarray。
  - 如果 *model_type* =’detection’，标签的格式应为 (gt_boxes, gt_labels)。

返回：

- **numpy.ndarray** - 对抗样本。

异常：

- **NotImplementedError** - *norm_level* 不在 [2, numpy.inf, ‘2’，‘inf’] 中。

class mindarmour.adv_robustness.attacks.**CarliniWagnerL2Attack**(*network*, *num_classes*, *box_min=0.0*, *box_max=1.0*, *bin_search_steps=5*, *max_iterations=1000*, *confidence=0*, *learning_rate=5e-3*, *initial_const=1e-2*, *abort_early_check_ratio=5e-2*, *targeted=False*, *fast=True*, *abort_early=True*, *sparse=True*)

使用 L2 范数的 Carlini & Wagner 攻击通过分别利用两个损失生成对抗样本：“对抗损失”可使生成的示例实际上是对抗性的，“距离损失”可以控制对抗样本的质量。

参考文献：Nicholas Carlini, David Wagner: “Towards Evaluating the Robustness of Neural Networks”。

参数：

- **network** (Cell) - 目标模型。
- **num_classes** (int) - 模型输出的标签数，应大于零。
- **box_min** (float) - 目标模型输入的下界。默认值：0。
- **box_max** (float) - 目标模型输入的上界。默认值：1.0。
- **bin_search_steps** (int) - 用于查找距离和置信度之间的最优 trade-off 常数的二分查找步数。默认值：5。
- **max_iterations** (int) - 最大迭代次数，应大于零。默认值：1000。
- **confidence** (float) - 对抗样本输出的置信度。默认值：0。
- **learning_rate** (float) - 攻击算法的学习率。默认值：5e-3。
- **initial_const** (float) - 用于平衡扰动范数和置信度差异的初始 trade-off 常数。默认值：1e-2。
- **abort_early_check_ratio** (float) - 检查所有迭代中所有比率的损失进度。默认值：5e-2。
- **targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **fast** (bool) - 如果为 True，则返回第一个找到的对抗样本。如果为 False，则返回扰动较小的对抗样本。默认值：True。
- **abort_early** (bool) - 是否提前终止。
  - 如果为 True，则当损失在一段时间内没有减少，Adam 将被中止。
  - 如果为 False，Adam 将继续工作，直到到达最大迭代。默认值：True。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.attacks import CarliniWagnerL2Attack
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> input_np = np.array([[0.1, 0.2, 0.7, 0.5, 0.4]]).astype(np.float32)
```

(continues on next page)

(continued from previous page)

```
>>> num_classes = input_np.shape[1]
>>> label_np = np.array([3]).astype(np.int64)
>>> attack = CarliniWagnerL2Attack(net, num_classes, targeted=False)
>>> adv_data = attack.generate(input_np, label_np)
```

generate(*inputs*, *labels*)

根据输入数据和目标标签生成对抗样本。

参数：

- **inputs** (numpy.ndarray) - 输入样本。
- **labels** (numpy.ndarray) - 输入样本的真值标签或目标标签。

返回：

- **numpy.ndarray** - 生成的对抗样本。

class mindarmour.adv_robustness.attacks.**JSMAAttack**(*network*, *num_classes*, *box_min=0.0*, *box_max=1.0*, *theta=1.0*, *max_iteration=1000*, *max_count=3*, *increase=True*, *sparse=True*)

基于 Jacobian 的显著图攻击（Jacobian-based Saliency Map Attack）是一种基于输入特征显著图的有目标的迭代攻击。它使用每个类标签相对于输入的每个组件的损失梯度。然后，使用显著图来选择产生最大误差的维度。

参考文献：The limitations of deep learning in adversarial settings。

参数：

- **network** (Cell) - 目标模型。
- **num_classes** (int) - 模型输出的标签数，应大于零。
- **box_min** (float) - 目标模型输入的下界。默认值：0。
- **box_max** (float) - 目标模型输入的上界。默认值：1.0。
- **theta** (float) - 一个像素的变化率（相对于输入数据范围）。默认值：1.0。
- **max_iteration** (int) - 迭代的最大轮次。默认值：1000。
- **max_count** (int) - 每个像素的最大更改次数。默认值：3。
- **increase** (bool) - 如果为 True，则增加扰动。如果为 False，则减少扰动。默认值：True。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

样例：

```
>>> from mindarmour.adv_robustness.attacks import JSMAAttack
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> net = Net()
>>> input_shape = (1, 5)
>>> batch_size, classes = input_shape
>>> input_np = np.random.random(input_shape).astype(np.float32)
>>> label_np = np.random.randint(classes, size=batch_size)
>>> attack = JSMAAttack(net, classes, max_iteration=5)
>>> advs = attack.generate(input_np, label_np)
```

generate(*inputs*, *labels*)

批量生成对抗样本。

参数：

- **inputs** (numpy.ndarray) - 输入样本。
- **labels** (numpy.ndarray) - 目标标签。

返回：

- **numpy.ndarray** - 对抗样本。

class mindarmour.adv_robustness.attacks.**LBFGS**(*network*, *eps=1e-5*, *bounds=(0.0, 1.0)*, *is_targeted=True*, *nb_iter=150*, *search_iters=30*, *loss_fn=None*, *sparse=False*)

L-BFGS-B 攻击使用有限内存 BFGS 优化算法来最小化输入与对抗样本之间的距离。

参考文献：Pedro Tabacof, Eduardo Valle. “Exploring the Space of Adversarial Images” 。

参数：

- **network** (Cell) - 被攻击模型的网络。

- **eps** (float) - 攻击步长。默认值：1e-5。
- **bounds** (tuple) - 数据的上下界。默认值：(0.0, 1.0)
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：True。
- **nb_iter** (int) - lbfgs 优化器的迭代次数，应大于零。默认值：150。
- **search_iters** (int) - 步长的变更数，应大于零。默认值：30。
- **loss_fn** (Functions) - 替代模型的损失函数。默认值：None。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：False。

样例：

```
>>> from mindarmour.adv_robustness.attacks import LBFGS
>>> import mindspore.ops.operations as P
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._reduce = P.ReduceSum()
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._reduce(out, 2)
...         out = self._squeeze(out)
...         return out
>>> net = Net()
>>> classes = 10
>>> attack = LBFGS(net, is_targeted=True)
>>> input_np = np.asarray(np.random.random((1,1,32,32)), np.float32)
>>> label_np = np.array([3]).astype(np.int64)
>>> target_np = np.array([7]).astype(np.int64)
>>> target_np = np.eye(10)[target_np].astype(np.float32)
>>> adv = attack.generate(input_np, target_np)
```

generate(*inputs*, *labels*)

根据输入数据和目标标签生成对抗样本。

参数：

- **inputs** (numpy.ndarray) - 良性输入样本，用于创建对抗样本。
- **labels** (numpy.ndarray) - 原始/目标标签。

返回：

- **numpy.ndarray** - 生成的对抗样本。

class mindarmour.adv_robustness.attacks.**GeneticAttack**(*model*, *model_type='classification'*, *targeted=True*, *reserve_ratio=0.3*, *sparse=True*, *pop_size=6*, *mutation_rate=0.005*, *per_bounds=0.15*, *max_steps=1000*, *step_size=0.20*, *temp=0.3*, *bounds=(0, 1.0)*, *adaptive=False*, *c=0.1*)

遗传攻击（Genetic Attack）为基于遗传算法的黑盒攻击，属于差分进化算法。

此攻击是由 Moustafa Alzantot 等人（2018）提出的。

参考文献：Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, “GeneticAttack: Practical Black-box Attacks with Gradient-FreeOptimization” 。

参数：

- **model** (BlackModel) - 目标模型。
- **model_type** (str) - 目标模型的类型。现在支持’classification’和’detection’。默认值:’classification’。
- **targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。*model_type* =’detection’仅支持无目标攻击，默认值：True。
- **reserve_ratio** (Union[int, float]) - 攻击后可检测到的对象百分比，仅当 *model_type* =’detection’时有效。保留比率应在 (0, 1) 的范围内。默认值：0.3。
- **pop_size** (int) - 粒子的数量，应大于零。默认值：6。
- **mutation_rate** (Union[int, float]) - 突变的概率，应在（0,1）的范围内。默认值：0.005。
- **per_bounds** (Union[int, float]) - 扰动允许的最大无穷范数距离。
- **max_steps** (int) - 每个对抗样本的最大迭代轮次。默认值：1000。
- **step_size** (Union[int, float]) - 攻击步长。默认值：0.2。
- **temp** (Union[int, float]) - 用于选择的采样温度。默认值：0.3。温度越大，个体选择概率之间的差异就越大。
- **bounds** (Union[tuple, list, None]) - 数据的上下界。以 (数据最小值，数据最大值) 的形式出现。默认值：(0, 1.0)。
- **adaptive** (bool) - 为 True，则打开突变参数的动态缩放。如果为 false，则打开静态突变参数。默认值：False。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

- **c** (Union[int, float]) - 扰动损失的权重。默认值：0.1。

样例：

```
>>> import mindspore.ops.operations as M
>>> from mindspore import Tensor
>>> from mindspore.nn import Cell
>>> from mindarmour import BlackModel
>>> from mindarmour.adv_robustness.attacks import GeneticAttack
>>> class ModelToBeAttacked(BlackModel):
...     def __init__(self, network):
...         super(ModelToBeAttacked, self).__init__()
...         self._network = network
...     def predict(self, inputs):
...         result = self._network(Tensor(inputs.astype(np.float32)))
...         return result.asnumpy()
>>> class Net(Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = M.Softmax()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         return out
>>> net = Net()
>>> model = ModelToBeAttacked(net)
>>> attack = GeneticAttack(model, sparse=False)
>>> batch_size = 6
>>> x_test = np.random.rand(batch_size, 10)
>>> y_test = np.random.randint(low=0, high=10, size=batch_size)
>>> y_test = np.eye(10)[y_test]
>>> y_test = y_test.astype(np.float32)
>>> _, adv_data, _ = attack.generate(x_test, y_test)
```

generate(*inputs*, *labels*)

根据输入数据和目标标签（或 ground_truth 标签）生成对抗样本。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 输入样本。
  - 如果 *model_type* =’classification’，则输入的格式应为 numpy.ndarray。输入的格式可以是 (input1, input2, …)。
  - 如果 *model_type* =’detection’，则只能是一个数组。
- **labels** (Union[numpy.ndarray, tuple]) - 目标标签或 ground-truth 标签。
  - 如果 *model_type* =’classification’，标签的格式应为 numpy.ndarray。
  - 如果 *model_type* =’detection’，标签的格式应为 (gt_boxes, gt_labels)。

返回：

- **numpy.ndarray** - 每个攻击结果的布尔值。
- **numpy.ndarray** - 生成的对抗样本。
- **numpy.ndarray** - 每个样本的查询次数。

class mindarmour.adv_robustness.attacks.**HopSkipJumpAttack**(*model*, *init_num_evals=100*, *max_num_evals=1000*, *stepsize_search='geometric_progression'*, *num_iterations=20*, *gamma=1.0*, *constraint='l2'*, *batch_size=32*, *clip_min=0.0*, *clip_max=1.0*, *sparse=True*)

Chen、Jordan 和 Wainwright 提出的 HopSkipJumpAttack 是一种基于决策的攻击。此攻击需要访问目标模型的输出标签。

参考文献：Chen J, Michael I. Jordan, Martin J. Wainwright. HopSkipJumpAttack: A Query-Efficient Decision-Based Attack. 2019. arXiv:1904.02144。

参数：

- **model** (BlackModel) - 目标模型。
- **init_num_evals** (int) - 梯度估计的初始评估数。默认值：100。
- **max_num_evals** (int) - 梯度估计的最大评估数。默认值：1000。
- **stepsize_search** (str) - 表示要如何搜索步长；
  - 可取值为’geometric_progression’或’grid_search’。默认值：’geometric_progression’。
- **num_iterations** (int) - 迭代次数。默认值：20。
- **gamma** (float) - 用于设置二进制搜索阈值 theta。默认值：1.0。对于 l2 攻击，二进制搜索阈值 *theta* 为 $gamma/d^{3/2}$ 。对于 linf 攻击是 $gamma/d^2$ 。默认值：1.0。
- **constraint** (str) - 要优化距离的范数。可取值为’l2’或’linf’。默认值：’l2’。
- **batch_size** (int) - 批次大小。默认值：32。

- **clip_min** (float, optional) - 最小图像组件值。默认值：0。
- **clip_max** (float, optional) - 最大图像组件值。默认值：1。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

异常：

- **ValueError** - *stepsize_search* 不在 [‘geometric_progression’,’grid_search’] 中。
- **ValueError** - *constraint* 不在 [‘l2’, ‘linf’] 中

样例：

```
>>> from mindspore import Tensor
>>> from mindarmour import BlackModel
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.attacks import HopSkipJumpAttack
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._reduce = P.ReduceSum()
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._reduce(out, 2)
...         out = self._squeeze(out)
...         return out
>>> class ModelToBeAttacked(BlackModel):
...     def __init__(self, network):
...         super(ModelToBeAttacked, self).__init__()
...         self._network = network
...     def predict(self, inputs):
...         if len(inputs.shape) == 3:
...             inputs = inputs[np.newaxis, :]
...         result = self._network(Tensor(inputs.astype(np.float32)))
...         return result.asnumpy()
>>> net = Net()
>>> model = ModelToBeAttacked(net)
>>> attack = HopSkipJumpAttack(model)
>>> n, c, h, w = 1, 1, 32, 32
>>> class_num = 3
>>> x_test = np.asarray(np.random.random((n,c,h,w)), np.float32)
>>> y_test = np.random.randint(0, class_num, size=n)
>>> _, adv_x, _= attack.generate(x_test, y_test)
```

generate(*inputs*, *labels*)

在 for 循环中生成对抗图像。

参数：

- **inputs** (numpy.ndarray) - 原始图像。
- **labels** (numpy.ndarray) - 目标标签。

返回：

- **numpy.ndarray** - 每个攻击结果的布尔值。
- **numpy.ndarray** - 生成的对抗样本。
- **numpy.ndarray** - 每个样本的查询次数。

set_target_images(*target_images*)

设置目标图像进行目标攻击。

参数：

- **target_images** (numpy.ndarray) - 目标图像。

class mindarmour.adv_robustness.attacks.**NES**(*model*, *scene*, *max_queries=10000*, *top_k=-1*, *num_class=10*, *batch_size=128*, *epsilon=0.3*, *samples_per_draw=128*, *momentum=0.9*, *learning_rate=1e-3*, *max_lr=5e-2*, *min_lr=5e-4*, *sigma=1e-3*, *plateau_length=20*, *plateau_drop=2.0*, *adv_thresh=0.25*, *zero_iters=10*, *starting_eps=1.0*, *starting_delta_eps=0.5*, *label_only_sigma=1e-3*, *conservative=2*, *sparse=True*)

该类是自然进化策略（Natural Evolutionary Strategies，NES）攻击法的实现。NES 使用自然进化策略来估计梯度，以提高查询效率。NES 包括三个设置：Query-Limited 设置、Partial-Information 置和 Label-Only 设置。

- 在’query-limit’设置中，攻击对目标模型的查询数量有限，但可以访问所有类的概率。
- 在’partial-info’设置中，攻击仅有权访问 top-k 类的概率。
- 在’label-only’设置中，攻击只能访问按其预测概率排序的 k 个推断标签列表。

在 Partial-Information 设置和 Label-Only 设置中，NES 会进行目标攻击，因此用户需要使用 set_target_images 方法来设置目标类的目标图像。

参考文献：Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In ICML, July 2018。

参数：

- **model** (BlackModel) - 要攻击的目标模型。
- **scene** (str) - 确定算法的场景，可选值为：’Label_Only’、’Partial_Info’、’Query_Limit’。
- **max_queries** (int) - 生成对抗样本的最大查询编号。默认值：10000。
- **top_k** (int) - 用于’Partial-Info’或’Label-Only’设置，表示攻击者可用的（Top-k）信息数量。对于 Query-Limited 设置，此输入应设置为-1。默认值：-1。
- **num_class** (int) - 数据集中的类数。默认值：10。
- **batch_size** (int) - 批次大小。默认值：128。
- **epsilon** (float) - 攻击中允许的最大扰动。默认值：0.3。
- **samples_per_draw** (int) - 对偶采样中绘制的样本数。默认值：128。
- **momentum** (float) - 动量。默认值：0.9。
- **learning_rate** (float) - 学习率。默认值：1e-3。
- **max_lr** (float) - 最大学习率。默认值：5e-2。
- **min_lr** (float) - 最小学习率。默认值：5e-4。
- **sigma** (float) - 随机噪声的步长。默认值：1e-3。
- **plateau_length** (int) - 退火算法中使用的平台长度。默认值：20。
- **plateau_drop** (float) - 退火算法中使用的平台 Drop。默认值：2.0。
- **adv_thresh** (float) - 对抗阈值。默认值：0.25。
- **zero_iters** (int) - 用于代理分数的点数。默认值：10。
- **starting_eps** (float) - Label-Only 设置中使用的启动 epsilon。默认值：1.0。
- **starting_delta_eps** (float) - Label-Only 设置中使用的 delta epsilon。默认值：0.5。
- **label_only_sigma** (float) - Label-Only 设置中使用的 Sigma。默认值：1e-3。
- **conservative** (int) - 用于 epsilon 衰变的守恒，如果没有收敛，它将增加。默认值：2。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

样例：

```
>>> from mindspore import Tensor
>>> from mindarmour import BlackModel
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.attacks import NES
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._reduce = P.ReduceSum()
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._reduce(out, 2)
...         out = self._squeeze(out)
...         return out
>>> class ModelToBeAttacked(BlackModel):
...     def __init__(self, network):
...         super(ModelToBeAttacked, self).__init__()
...         self._network = network
...     def predict(self, inputs):
...         if len(inputs.shape) == 1:
...             inputs = np.expand_dims(inputs, axis=0)
...         result = self._network(Tensor(inputs.astype(np.float32)))
...         return result.asnumpy()
>>> net = Net()
>>> model = ModelToBeAttacked(net)
>>> SCENE = 'Query_Limit'
>>> TOP_K = -1
>>> attack= NES(model, SCENE, top_k=TOP_K)
>>> num_class = 5
>>> x_test = np.asarray(np.random.random((1, 1, 32, 32)), np.float32)
>>> target_image  = np.asarray(np.random.random((1, 1, 32, 32)), np.float32)
```

(continues on next page)

(continued from previous page)

```
>>> orig_class = 0
>>> target_class = 2
>>> attack.set_target_images(target_image)
>>> tag, adv, queries = attack.generate(np.array(x_test), np.array([target_class]))
```

generate(*inputs*, *labels*)

根据输入数据和目标标签生成对抗样本。

参数：

- **inputs** (numpy.ndarray) - 良性输入样本。
- **labels** (numpy.ndarray) - 目标标签。

返回：

- **numpy.ndarray** - 每个攻击结果的布尔值。
- **numpy.ndarray** - 生成的对抗样本。
- **numpy.ndarray** - 每个样本的查询次数。

异常：

- **ValueError** - 在’Label-Only’或’Partial-Info’设置中 *top_k* 小于 0。
- **ValueError** - 在’Label-Only’或’Partial-Info’设置中 target_imgs 为 None。
- **ValueError** - *scene* 不在 [‘Label_Only’, ‘Partial_Info’, ‘Query_Limit’] 中

set_target_images(*target_images*)

在’Partial-Info’或’Label-Only’设置中设置目标攻击的目标样本。

参数：

- **target_images** (numpy.ndarray) - 目标攻击的目标样本。

class mindarmour.adv_robustness.attacks.**PointWiseAttack**(*model*, *max_iter=1000*, *search_iter=10*, *is_targeted=False*, *init_attack=None*, *sparse=True*)

点式攻击（Pointwise Attack）确保使用最小数量的更改像素为每个原始样本生成对抗样本。那些更改的像素将使用二进制搜索，以确保对抗样本和原始样本之间的距离尽可能接近。

参考文献：L. Schott, J. Rauber, M. Bethge, W. Brendel: “Towards the first adversarially robust neural network model on MNIST”, ICLR (2019)。

参数：

- **model** (BlackModel) - 目标模型。
- **max_iter** (int) - 生成对抗图像的最大迭代轮数。默认值：1000。
- **search_iter** (int) - 二进制搜索的最大轮数。默认值：10。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **init_attack** (Union[Attack, None]) - 用于查找起点的攻击。默认值：None。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

样例：

```
>>> from mindspore import Tensor
>>> from mindarmour import BlackModel
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.attacks import PointWiseAttack
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._reduce = P.ReduceSum()
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._reduce(out, 2)
...         out = self._squeeze(out)
...         return out
>>> class ModelToBeAttacked(BlackModel):
...     def __init__(self, network):
...         super(ModelToBeAttacked, self).__init__()
...         self._network = network
...     def predict(self, inputs):
...         result = self._network(Tensor(inputs.astype(np.float32)))
...         return result.asnumpy()
>>> net = Net()
```

(continues on next page)

(continued from previous page)

```
>>> np.random.seed(5)
>>> model = ModelToBeAttacked(net)
>>> attack = PointWiseAttack(model)
>>> x_test = np.asarray(np.random.random((1,1,32,32)), np.float32)
>>> y_test = np.random.randint(0, 3, size=1)
>>> is_adv_list, adv_list, query_times_each_adv = attack.generate(x_test, y_test)
```

generate(*inputs*, *labels*)

根据输入样本和目标标签生成对抗样本。

参数：

- **inputs** (numpy.ndarray) - 良性输入样本，用于创建对抗样本。
- **labels** (numpy.ndarray) - 对于有目标的攻击，标签是对抗性的目标标签。对于无目标攻击，标签是 ground-truth 标签。

返回：

- **numpy.ndarray** - 每个攻击结果的布尔值。
- **numpy.ndarray** - 生成的对抗样本。
- **numpy.ndarray** - 每个样本的查询次数。

class mindarmour.adv_robustness.attacks.**PSOAttack**(*model*, *model_type='classification'*, *targeted=False*, *reserve_ratio=0.3*, *sparse=True*, *step_size=0.5*, *per_bounds=0.6*, *c1=2.0*, *c2=2.0*, *c=2.0*, *pop_size=6*, *t_max=1000*, *pm=0.5*, *bounds=None*)

PSO 攻击表示基于粒子群优化（Particle Swarm Optimization）算法的黑盒攻击，属于进化算法。此攻击由 Rayan Mosli 等人（2019）提出。

参考文献：Rayan Mosli, Matthew Wright, Bo Yuan, Yin Pan, “They Might NOT Be Giants: Crafting Black-Box Adversarial Examples with Fewer Queries Using Particle Swarm Optimization”, arxiv: 1909.07490, 2019.。

参数：

- **model** (BlackModel) - 目标模型。
- **step_size** (Union[int, float]) - 攻击步长。默认值：0.5。
- **per_bounds** (Union[int, float]) - 扰动的相对变化范围。默认值：0.6。
- **c1** (Union[int, float]) - 权重系数。默认值：2。
- **c2** (Union[int, float]) - 权重系数。默认值：2。
- **c** (Union[int, float]) - 扰动损失的权重。默认值：2。
- **pop_size** (int) - 粒子的数量，应大于零。默认值：6。
- **t_max** (int) - 每个对抗样本的最大迭代轮数，应大于零。默认值：1000。
- **pm** (Union[int, float]) - 突变的概率，应在（0,1）的范围内。默认值：0.5。
- **bounds** (Union[list, tuple, None]) - 数据的上下界。以 (数据最小值，数据最大值) 的形式出现。默认值：None。
- **targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。*model_type* =’detection’仅支持无目标攻击，默认值：False。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。
- **model_type** (str) - 目标模型的类型。现在支持’classification’和’detection’。默认值:’classification’。
- **reserve_ratio** (Union[int, float]) - 攻击后可检测到的对象百分比，用于 *model_type* =’detection’模式。保留比率应在 (0, 1) 的范围内。默认值：0.3。

样例：

```
>>> import mindspore.nn as nn
>>> from mindspore import Tensor
>>> from mindspore.nn import Cell
>>> from mindarmour import BlackModel
>>> from mindarmour.adv_robustness.attacks import PSOAttack
>>> class ModelToBeAttacked(BlackModel):
...     def __init__(self, network):
...         super(ModelToBeAttacked, self).__init__()
...         self._network = network
...     def predict(self, inputs):
...         if len(inputs.shape) == 1:
...             inputs = np.expand_dims(inputs, axis=0)
...         result = self._network(Tensor(inputs.astype(np.float32)))
...         return result.asnumpy()
>>> class Net(Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
```

(continues on next page)

(continued from previous page)

```
...         return out
>>> net = Net()
>>> model = ModelToBeAttacked(net)
>>> attack = PSOAttack(model, bounds=(0.0, 1.0), pm=0.5, sparse=False)
>>> batch_size = 6
>>> x_test = np.random.rand(batch_size, 10)
>>> y_test = np.random.randint(low=0, high=10, size=batch_size)
>>> y_test = np.eye(10)[y_test]
>>> y_test = y_test.astype(np.float32)
>>> _, adv_data, _ = attack.generate(x_test, y_test)
```

generate(*inputs*, *labels*)

根据输入数据和目标标签（或 ground_truth 标签）生成对抗样本。

参数：

- **inputs** (Union[numpy.ndarray, tuple]) - 输入样本。
  - 如果 *model_type* =’classification’，则输入的格式应为 numpy.ndarray。输入的格式可以是 (input1, input2, …)。
  - 如果 *model_type* =’detection’，则只能是一个数组。
- **labels** (Union[numpy.ndarray, tuple]) - 目标标签或 ground-truth 标签。
  - 如果 *model_type* =’classification’，标签的格式应为 numpy.ndarray。
  - 如果 *model_type* =’detection’，标签的格式应为 (gt_boxes, gt_labels)。

返回：

- **numpy.ndarray** - 每个攻击结果的布尔值。
- **numpy.ndarray** - 生成的对抗样本。
- **numpy.ndarray** - 每个样本的查询次数。

class mindarmour.adv_robustness.attacks.**SaltAndPepperNoiseAttack**(*model*, *bounds=(0.0, 1.0)*, *max_iter=100*, *is_targeted=False*, *sparse=True*)

增加椒盐噪声的量以生成对抗样本。

参数：

- **model** (BlackModel) - 目标模型。
- **bounds** (tuple) - 数据的上下界。以 (数据最小值，数据最大值) 的形式出现。默认值：(0.0, 1.0)。
- **max_iter** (int) - 生成对抗样本的最大迭代。默认值：100。
- **is_targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 one-hot 编码。默认值：True。

样例：

```
>>> from mindspore import Tensor
>>> from mindarmour import BlackModel
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.attacks import SaltAndPepperNoiseAttack
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._reduce = P.ReduceSum()
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._reduce(out, 2)
...         out = self._squeeze(out)
...         return out
>>> class ModelToBeAttacked(BlackModel):
...     def __init__(self, network):
...         super(ModelToBeAttacked, self).__init__()
...         self._network = network
...     def predict(self, inputs):
...         if len(inputs.shape) == 1:
...             inputs = np.expand_dims(inputs, axis=0)
...         result = self._network(Tensor(inputs.astype(np.float32)))
...         return result.asnumpy()
>>> net = Net()
>>> model = ModelToBeAttacked(net)
>>> attack = SaltAndPepperNoiseAttack(model)
```

(continues on next page)

(continued from previous page)

```
>>> x_test = np.asarray(np.random.random((1,1,32,32)), np.float32)
>>> y_test = np.random.randint(0, 3, size=1)
>>> _, adv_list, _ = attack.generate(x_test, y_test)
```

generate(*inputs*, *labels*)

根据输入数据和目标标签生成对抗样本。

参数：

- **inputs** (numpy.ndarray) - 原始的、未受扰动的输入。
- **labels** (numpy.ndarray) - 目标标签。

返回：

- **numpy.ndarray** - 每个攻击结果的布尔值。
- **numpy.ndarray** - 生成的对抗样本。
- **numpy.ndarray** - 每个样本的查询次数。

CHAPTER

THREE

MINDARMOUR.ADV_ROBUSTNESS.DEFENSES

该模块包括经典的防御算法，用于防御对抗样本，增强模型的安全性和可信性。

class mindarmour.adv_robustness.defenses.**AdversarialDefense**(*network*, *loss_fn=None*, *optimizer=None*)

使用给定的对抗样本进行对抗训练。

参数：

- **network** (Cell) - 要防御的 MindSpore 网络。
- **loss_fn** (Union[Loss, None]) - 损失函数。默认值：None。
- **optimizer** (Cell) - 用于训练网络的优化器。默认值：None。

样例：

```
>>> from mindspore.nn.optim.momentum import Momentum
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.defenses import AdversarialDefense
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._dense = nn.Dense(10, 10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._dense(out)
...         out = self._squeeze(out)
...         return out
>>> net = Net()
>>> lr = 0.001
>>> momentum = 0.9
>>> batch_size = 16
>>> num_classes = 10
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> optimizer = Momentum(net.trainable_params(), learning_rate=lr, momentum=momentum)
>>> adv_defense = AdversarialDefense(net, loss_fn, optimizer)
>>> inputs = np.random.rand(batch_size, 1, 10).astype(np.float32)
>>> labels = np.random.randint(10, size=batch_size).astype(np.int32)
>>> labels = np.eye(num_classes)[labels].astype(np.float32)
>>> adv_defense.defense(inputs, labels)
```

defense(*inputs*, *labels*)

通过使用输入样本进行训练来增强模型。

参数：

- **inputs** (numpy.ndarray) - 输入样本。
- **labels** (numpy.ndarray) - 输入样本的标签。

返回：

- **numpy.ndarray** - 防御操作的损失。

class mindarmour.adv_robustness.defenses.**AdversarialDefenseWithAttacks**(*network*, *attacks*, *loss_fn=None*, *optimizer=None*, *bounds=(0.0, 1.0)*, *replace_ratio=0.5*)

利用特定的攻击方法和给定的对抗例子进行对抗训练，以增强模型的鲁棒性。

参数：

- **network** (Cell) - 要防御的 MindSpore 网络。
- **attacks** (list[Attack]) - 攻击方法序列。
- **loss_fn** (Union[Loss, None]) - 损失函数。默认值：None。
- **optimizer** (Cell) - 用于训练网络的优化器。默认值：None。
- **bounds** (tuple) - 数据的上下界。以 (clip_min, clip_max) 的形式出现。默认值：(0.0, 1.0)。

- **replace_ratio** (float) - 用对抗样本替换原始样本的比率，必须在 0 到 1 之间。默认值：0.5。

异常：

- **ValueError** - *replace_ratio* 不在 0 和 1 之间。

样例：

```
>>> from mindspore.nn.optim.momentum import Momentum
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.attacks import FastGradientSignMethod
>>> from mindarmour.adv_robustness.attacks import ProjectedGradientDescent
>>> from mindarmour.adv_robustness.defenses import AdversarialDefenseWithAttacks
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._dense = nn.Dense(10, 10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._dense(out)
...         out = self._squeeze(out)
...         return out
>>> net = Net()
>>> lr = 0.001
>>> momentum = 0.9
>>> batch_size = 16
>>> num_classes = 10
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> optimizer = Momentum(net.trainable_params(), learning_rate=lr, momentum=momentum)
>>> fgsm = FastGradientSignMethod(net, loss_fn=loss_fn)
>>> pgd = ProjectedGradientDescent(net, loss_fn=loss_fn)
>>> ead = AdversarialDefenseWithAttacks(net, [fgsm, pgd], loss_fn=loss_fn,
...                                     optimizer=optimizer)
>>> inputs = np.random.rand(batch_size, 1, 10).astype(np.float32)
>>> labels = np.random.randint(10, size=batch_size).astype(np.int32)
>>> labels = np.eye(num_classes)[labels].astype(np.float32)
>>> loss = ead.defense(inputs, labels)
```

defense(*inputs*, *labels*)

通过使用从输入样本生成的对抗样本进行训练来增强模型。

参数：

- **inputs** (numpy.ndarray) - 输入样本。
- **labels** (numpy.ndarray) - 输入样本的标签。

返回：

- **numpy.ndarray** - 对抗性防御操作的损失。

class mindarmour.adv_robustness.defenses.**NaturalAdversarialDefense**(*network*, *loss_fn=None*, *optimizer=None*, *bounds=(0.0*, *1.0)*, *replace_ratio=0.5*, *eps=0.1*)

基于 FGSM 的对抗性训练。

参考文献：A. Kurakin, et al., “Adversarial machine learning at scale,” in ICLR, 2017。

参数：

- **network** (Cell) - 要防御的 MindSpore 网络。
- **loss_fn** (Union[Loss, None]) - 损失函数。默认值：None。
- **optimizer** (Cell) - 用于训练网络的优化器。默认值：None。
- **bounds** (tuple) - 数据的上下界。以 (clip_min, clip_max) 的形式出现。默认值：(0.0, 1.0)。
- **replace_ratio** (float) - 用对抗样本替换原始样本的比率。默认值：0.5。
- **eps** (float) - 攻击方法（FGSM）的步长。默认值：0.1。

样例：

```
>>> from mindspore.nn.optim.momentum import Momentum
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.defenses import NaturalAdversarialDefense
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
```

(continues on next page)

(continued from previous page)

```
...         self._dense = nn.Dense(10, 10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._dense(out)
...         out = self._squeeze(out)
...         return out
>>> net = Net()
>>> lr = 0.001
>>> momentum = 0.9
>>> batch_size = 16
>>> num_classes = 10
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> optimizer = Momentum(net.trainable_params(), learning_rate=lr, momentum=momentum)
>>> nad = NaturalAdversarialDefense(net, loss_fn=loss_fn, optimizer=optimizer)
>>> inputs = np.random.rand(batch_size, 1, 10).astype(np.float32)
>>> labels = np.random.randint(10, size=batch_size).astype(np.int32)
>>> labels = np.eye(num_classes)[labels].astype(np.float32)
>>> loss = nad.defense(inputs, labels)
```

class mindarmour.adv_robustness.defenses.**ProjectedAdversarialDefense**(*network*, *loss_fn=None*, *optimizer=None*, *bounds=(0.0, 1.0)*, *replace_ratio=0.5*, *eps=0.3*, *eps_iter=0.1*, *nb_iter=5*, *norm_level='inf'*)

基于 PGD 的对抗性训练。

参考文献：A. Madry, et al., “Towards deep learning models resistant to adversarial attacks,” in ICLR, 2018。

参数：

- **network** (Cell) - 要防御的 MindSpore 网络。
- **loss_fn** (Union[Loss, None]) - 损失函数。默认值：None。
- **optimizer** (Cell) - 用于训练网络的优化器。默认值：None。
- **bounds** (tuple) - 输入数据的上下界。以 (clip_min, clip_max) 的形式出现。默认值：(0.0, 1.0)。
- **replace_ratio** (float) - 用对抗样本替换原始样本的比率。默认值：0.5。
- **eps** (float) - PGD 攻击参数 epsilon。默认值：0.3。
- **eps_iter** (int) - PGD 攻击参数，内环 epsilon。默认值：0.1。
- **nb_iter** (int) - PGD 攻击参数，迭代次数。默认值：5。
- **norm_level** (Union[int, char, numpy.inf]) - 范数类型。可选值：1、2、np.inf、’l1’、’l2’、’np.inf’ 或 ‘inf’。默认值：’inf’。

样例：

```
>>> from mindspore.nn.optim.momentum import Momentum
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.defenses import ProjectedAdversarialDefense
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._dense = nn.Dense(10, 10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._dense(out)
...         out = self._squeeze(out)
...         return out
>>> net = Net()
>>> lr = 0.001
>>> momentum = 0.9
>>> batch_size = 16
>>> num_classes = 10
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> optimizer = Momentum(net.trainable_params(), learning_rate=lr, momentum=momentum)
>>> pad = ProjectedAdversarialDefense(net, loss_fn=loss_fn, optimizer=optimizer)
>>> inputs = np.random.rand(batch_size, 1, 10).astype(np.float32)
>>> labels = np.random.randint(10, size=batch_size).astype(np.int32)
>>> labels = np.eye(num_classes)[labels].astype(np.float32)
>>> loss = pad.defense(inputs, labels)
```

class mindarmour.adv_robustness.defenses.**EnsembleAdversarialDefense**(*network*, *attacks*, *loss_fn=None*, *optimizer=None*, *bounds=(0.0, 1.0)*, *replace_ratio=0.5*)

使用特定攻击方法列表和给定的对抗样本进行对抗训练，以增强模型的鲁棒性。

参数：

- **network** (Cell) - 要防御的 MindSpore 网络。
- **attacks** (list[Attack]) - 攻击方法序列。
- **loss_fn** (Union[Loss, None]) - 损失函数。默认值：None。
- **optimizer** (Cell) - 用于训练网络的优化器。默认值：None。
- **bounds** (tuple) - 数据的上下界。以 (clip_min, clip_max) 的形式出现。默认值：(0.0, 1.0)。
- **replace_ratio** (float) - 用对抗样本替换原始样本的比率，必须在 0 到 1 之间。默认值：0.5。

异常：

- **ValueError** - *replace_ratio* 不在 0 和 1 之间。

样例：

```
>>> from mindspore.nn.optim.momentum import Momentum
>>> import mindspore.ops.operations as P
>>> from mindarmour.adv_robustness.attacks import FastGradientSignMethod
>>> from mindarmour.adv_robustness.attacks import ProjectedGradientDescent
>>> from mindarmour.adv_robustness.defenses import EnsembleAdversarialDefense
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._dense = nn.Dense(10, 10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._dense(out)
...         out = self._squeeze(out)
...         return out
>>> net = Net()
>>> lr = 0.001
>>> momentum = 0.9
>>> batch_size = 16
>>> num_classes = 10
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=False)
>>> optimizer = Momentum(net.trainable_params(), learning_rate=lr, momentum=momentum)
>>> fgsm = FastGradientSignMethod(net, loss_fn=loss_fn)
>>> pgd = ProjectedGradientDescent(net, loss_fn=loss_fn)
>>> ead = EnsembleAdversarialDefense(net, [fgsm, pgd], loss_fn=loss_fn,
...                                  optimizer=optimizer)
>>> inputs = np.random.rand(batch_size, 1, 10).astype(np.float32)
>>> labels = np.random.randint(10, size=batch_size).astype(np.int32)
>>> labels = np.eye(num_classes)[labels].astype(np.float32)
>>> loss = ead.defense(inputs, labels)
```

CHAPTER

FOUR

MINDARMOUR.ADV_ROBUSTNESS.DETECTORS

此模块是用于区分对抗样本和良性样本的检测器方法。

class mindarmour.adv_robustness.detectors.**ErrorBasedDetector**(*auto_encoder*, *false_positive_rate=0.01*, *bounds=(0.0*, *1.0)*)

检测器重建输入样本，测量重建误差，并拒绝重建误差大的样本。

参考文献：MagNet: a Two-Pronged Defense against Adversarial Examples, by Dongyu Meng and Hao Chen, at CCS 2017.。

参数：

- **auto_encoder** (Model) - 一个（训练过的）自动编码器，对输入图片进行重构。
- **false_positive_rate** (float) - 检测器的误报率。默认值：0.01。
- **bounds** (tuple) - (clip_min, clip_max)。默认值：(0.0, 1.0)。

样例：

```
>>> from mindspore.ops.operations import Add
>>> from mindspore import Model
>>> from mindarmour.adv_robustness.detectors import ErrorBasedDetector
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.add = Add()
...     def construct(self, inputs):
...         return self.add(inputs, inputs)
>>> np.random.seed(5)
>>> ori = np.random.rand(4, 4, 4).astype(np.float32)
>>> np.random.seed(6)
>>> adv = np.random.rand(4, 4, 4).astype(np.float32)
>>> model = Model(Net())
>>> detector = ErrorBasedDetector(model)
>>> detector.fit(ori)
>>> adv_ids = detector.detect(adv)
>>> adv_trans = detector.transform(adv)
```

detect(*inputs*)

检测输入样本是否具有对抗性。

参数：

- **inputs** (numpy.ndarray) - 待判断的可疑样本。

返回：

- **list[int]** - 样本是否具有对抗性。如果 res[i]=1，则索引为 i 的输入样本是对抗性的。

detect_diff(*inputs*)

检测原始样本和重建样本之间的距离。

参数：

- **inputs** (numpy.ndarray) - 输入样本。

返回：

- **float** - 重建样本和原始样本之间的距离。

fit(*inputs*, *labels=None*)

查找给定数据集的阈值，以区分对抗样本。

参数：

- **inputs** (numpy.ndarray) - 输入样本。
- **labels** (numpy.ndarray) - 输入样本的标签。默认值：None。

返回：

- **float** - 区分对抗样本和良性样本的阈值。

set_threshold(*threshold*)

设置阈值。

参数：

- **threshold** (float) - 检测阈值。

transform(*inputs*)

重建输入样本。

参数：

- **inputs** (numpy.ndarray) - 输入样本。

返回：

- **numpy.ndarray** - 重建图像。

class mindarmour.adv_robustness.detectors.**DivergenceBasedDetector**(*auto_encoder*, *model*, *option='jsd'*, *t=1*, *bounds=(0.0, 1.0)*)

基于发散的检测器学习通过 js 发散来区分正常样本和对抗样本。

参考文献：MagNet: a Two-Pronged Defense against Adversarial Examples, by Dongyu Meng and Hao Chen, at CCS 2017.。

参数：

- **auto_encoder** (Model) - 编码器模型。
- **model** (Model) - 目标模型。
- **option** (str) - 用于计算发散的方法。默认值：’ jsd’。
- **t** (int) - 用于克服数值问题的温度。默认值：1。
- **bounds** (tuple) - 数据的上下界。以 (clip_min, clip_max) 的形式出现。默认值：(0.0, 1.0)。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindspore.nn import Cell
>>> from mindspore import Model
>>> from mindarmour.adv_robustness.detectors import DivergenceBasedDetector
>>> class PredNet(Cell):
...     def __init__(self):
...         super(PredNet, self).__init__()
...         self.shape = P.Shape()
...         self.reshape = P.Reshape()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         data = self.reshape(inputs, (self.shape(inputs)[0], -1))
...         return self._softmax(data)
>>> class Net(Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.add = P.Add()
...     def construct(self, inputs):
...         return self.add(inputs, inputs)
>>> np.random.seed(5)
>>> ori = np.random.rand(4, 4, 4).astype(np.float32)
>>> np.random.seed(6)
>>> adv = np.random.rand(4, 4, 4).astype(np.float32)
>>> encoder = Model(Net())
>>> model = Model(PredNet())
>>> detector = DivergenceBasedDetector(encoder, model)
>>> threshold = detector.fit(ori)
>>> detector.set_threshold(threshold)
>>> adv_ids = detector.detect(adv)
>>> adv_trans = detector.transform(adv)
```

detect_diff(*inputs*)

检测原始样本和重建样本之间的距离。

距离由 JSD 计算。

参数：

- **inputs** (numpy.ndarray) - 输入样本。

返回：

- **float** - 距离。

异常：

- **NotImplementedError** - 不支持参数 *option* 。

class mindarmour.adv_robustness.detectors.**RegionBasedDetector**(*model*, *number_points=10*, *initial_radius=0.0*, *max_radius=1.0*, *search_step=0.01*, *degrade_limit=0.0*, *sparse=False*)

基于区域的检测器利用对抗样本靠近分类边界的事实，并通过集成给定示例周围的信息，以检测输入是否为对抗样本。

参考文献：Mitigating evasion attacks to deep neural networks via region-based classification。

参数：

- **model** (Model) - 目标模型。
- **number_points** (int) - 从原始样本的超立方体生成的样本数。默认值：10。
- **initial_radius** (float) - 超立方体的初始半径。默认值：0.0。
- **max_radius** (float) - 超立方体的最大半径。默认值：1.0。
- **search_step** (float) - 半径搜索增量。默认值：0.01。
- **degrade_limit** (float) - 分类精度的可接受下降。默认值：0.0。
- **sparse** (bool) - 如果为 True，则输入标签为稀疏编码。如果为 False，则输入标签为 onehot 编码。默认值：False。

样例：

```
>>> from mindspore.ops.operations import Add
>>> from mindspore import Model
>>> from mindarmour.adv_robustness.detectors import RegionBasedDetector
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.add = Add()
...     def construct(self, inputs):
...         return self.add(inputs, inputs)
>>> np.random.seed(5)
>>> ori = np.random.rand(4, 4).astype(np.float32)
>>> labels = np.array([[1, 0, 0, 0], [0, 0, 1, 0], [0, 0, 1, 0],
...                    [0, 1, 0, 0]]).astype(np.int32)
>>> np.random.seed(6)
>>> adv = np.random.rand(4, 4).astype(np.float32)
>>> model = Model(Net())
>>> detector = RegionBasedDetector(model)
>>> radius = detector.fit(ori, labels)
>>> detector.set_radius(radius)
>>> adv_ids = detector.detect(adv)
```

detect(*inputs*)

判断输入样本是否具有对抗性。

参数：

- **inputs** (numpy.ndarray) - 待判断的可疑样本。

返回：

- **list[int]** - 样本是否具有对抗性。如果 res[i]=1，则索引为 i 的输入样本是对抗性的。

detect_diff(*inputs*)

返回原始预测结果和基于区域的预测结果。

参数：

- **inputs** (numpy.ndarray) - 输入样本。

返回：

- **numpy.ndarray** - 输入样本的原始预测结果和基于区域的预测结果。

fit(*inputs*, *labels=None*)

训练检测器来决定最佳半径。

参数：

- **inputs** (numpy.ndarray) - 良性样本。
- **labels** (numpy.ndarray) - 输入样本的 ground truth 标签。默认值：None。

返回：

- **float** - 最佳半径。

set_radius(*radius*)

设置半径。

参数：

- **radius** (float) - 区域的半径。

transform(*inputs*)

为输入样本生成超级立方体。

参数：

- **inputs** (numpy.ndarray) - 输入样本。

返回：

- **numpy.ndarray** - 每个样本对应的超立方体。

class mindarmour.adv_robustness.detectors.**SpatialSmoothing**(*model*, *ksize=3*, *is_local_smooth=True*, *metric='l1'*, *false_positive_ratio=0.05*)

基于空间平滑的检测方法。使用高斯滤波、中值滤波和均值滤波，模糊原始图像。当模型在样本模糊前后的预测值之间有很大的阈值差异时，将其判断为对抗样本。

参数：

- **model** (Model) - 目标模型。
- **ksize** (int) - 平滑窗口大小。默认值：3。
- **is_local_smooth** (bool) - 如果为 True，则触发局部平滑。如果为 False，则无局部平滑。默认值：True。
- **metric** (str) - 距离方法。默认值：’l1’。
- **false_positive_ratio** (float) - 良性样本上的假正率。默认值：0.05。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindspore import Model
>>> from mindarmour.adv_robustness.detectors import SpatialSmoothing
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...     def construct(self, inputs):
...         return self._softmax(inputs)
>>> input_shape = (50, 3)
>>> np.random.seed(1)
>>> input_np = np.random.randn(*input_shape).astype(np.float32)
>>> np.random.seed(2)
>>> adv_np = np.random.randn(*input_shape).astype(np.float32)
>>> model = Model(Net())
>>> detector = SpatialSmoothing(model)
>>> threshold = detector.fit(input_np)
>>> detector.set_threshold(threshold.item())
>>> detected_res = np.array(detector.detect(adv_np))
```

detect(*inputs*)

检测输入样本是否为对抗样本。

参数：

- **inputs** (numpy.ndarray) - 待判断的可疑样本。

返回：

- **list[int]** - 样本是否具有对抗性。如果 res[i]=1，则索引为 i 的输入样本是对抗样本。

detect_diff(*inputs*)

返回输入样本与其平滑对应样本之间的原始距离值（在应用阈值之前）。

参数：

- **inputs** (numpy.ndarray) - 待判断的可疑样本。

返回：

- **float** - 距离。

fit(*inputs*, *labels=None*)

训练检测器来决定阈值。适当的阈值能够确保良性样本上的实际假正率小于给定值。

参数：

- **inputs** (numpy.ndarray) - 良性样本。
- **labels** (numpy.ndarray) - 默认 None。

返回：

- **float** - 阈值，大于该距离的距离报告为正，即对抗性。

set_threshold(*threshold*)

设置阈值。

参数：

- **threshold** (float) - 检测阈值。

class mindarmour.adv_robustness.detectors.**EnsembleDetector**(*detectors*, *policy='vote'*)

集合检测器，通过检测器列表从输入样本中检测对抗样本。

参数：

- **detectors** (Union[tuple, list]) - 检测器方法列表。
- **policy** (str) - 决策策略，取值可为’vote’、’all’、’any’。默认值：’vote’

样例：

```
>>> from mindspore.ops.operations import Add
>>> from mindspore import Model
>>> from mindarmour.adv_robustness.detectors import ErrorBasedDetector
>>> from mindarmour.adv_robustness.detectors import RegionBasedDetector
>>> from mindarmour.adv_robustness.detectors import EnsembleDetector
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.add = Add()
...     def construct(self, inputs):
...         return self.add(inputs, inputs)
>>> class AutoNet(Cell):
...     def __init__(self):
...         super(AutoNet, self).__init__()
...         self.add = Add()
...     def construct(self, inputs):
...         return self.add(inputs, inputs)
>>> np.random.seed(6)
>>> adv = np.random.rand(4, 4).astype(np.float32)
>>> model = Model(Net())
>>> auto_encoder = Model(AutoNet())
>>> random_label = np.random.randint(10, size=4)
>>> labels = np.eye(10)[random_label]
>>> magnet_detector = ErrorBasedDetector(auto_encoder)
>>> region_detector = RegionBasedDetector(model)
>>> region_detector.fit(adv, labels)
>>> detectors = [magnet_detector, region_detector]
>>> detector = EnsembleDetector(detectors)
>>> adv_ids = detector.detect(adv)
```

detect(*inputs*)

从输入样本中检测对抗性示例。

参数：

- **inputs** (numpy.ndarray) - 输入样本。

返回：

- **list[int]** - 样本是否具有对抗性。如果 res[i]=1，则索引为 i 的输入样本是对抗样本。

异常：

- **ValueError** - 不支持策略。

detect_diff(*inputs*)

此方法在此类中不可用。

参数：

- **inputs** (Union[numpy.ndarray, list, tuple]) - 用于创建对抗样本。

异常：

- **NotImplementedError** - 此函数在集成中不可用。

fit(*inputs*, *labels=None*)

像机器学习模型一样拟合检测器。此方法在此类中不可用。

参数：

- **inputs** (numpy.ndarray) - 计算阈值的数据。
- **labels** (numpy.ndarray) - 数据的标签。默认值：None。

异常：

- **NotImplementedError** - 此函数在集成中不可用。

transform(*inputs*)

过滤输入样本中的对抗性噪声。此方法在此类中不可用。

参数：

- **inputs** (Union[numpy.ndarray, list, tuple]) - 用于创建对抗样本。

异常：

- **NotImplementedError** - 此函数在集成中不可用。

class mindarmour.adv_robustness.detectors.**SimilarityDetector**(*trans_model*, *max_k_neighbor=1000*, *chunk_size=1000*, *max_buffer_size=10000*, *tuning=False*, *fpr=0.001*)

检测器测量相邻查询之间的相似性，并拒绝与以前的查询非常相似的查询。

参考文献：Stateful Detection of Black-Box Adversarial Attacks by Steven Chen, Nicholas Carlini, and David Wagner. at arxiv 2019。

参数：

- **trans_model** (Model) - 一个 MindSpore 模型，将输入数据编码为低维向量。
- **max_k_neighbor** (int) - 最近邻的最大数量。默认值：1000。
- **chunk_size** (int) - 缓冲区大小。默认值：1000。
- **max_buffer_size** (int) - 最大缓冲区大小。默认值：10000。
- **tuning** (bool) - 计算 k 个最近邻的平均距离。
  - 如果’tuning’为 true，k= *max_k_neighbor* 。
  - 如果为 False，k=1,⋯, *max_k_neighbor* 。默认值：False。
- **fpr** (float) - 合法查询序列上的误报率。默认值：0.001

样例：

```
>>> from mindspore.ops.operations import Add
>>> from mindspore.nn import Cell
>>> from mindspore import Model
>>> from mindarmour.adv_robustness.detectors import SimilarityDetector
>>> class EncoderNet(Cell):
...     def __init__(self, encode_dim):
...         super(EncoderNet, self).__init__()
...         self._encode_dim = encode_dim
...         self.add = Add()
...     def construct(self, inputs):
...         return self.add(inputs, inputs)
...     def get_encode_dim(self):
...         return self._encode_dim
>>> np.random.seed(5)
>>> x_train = np.random.rand(10, 32, 32, 3).astype(np.float32)
>>> perm = np.random.permutation(x_train.shape[0])
>>> benign_queries = x_train[perm[:10], :, :, :]
>>> suspicious_queries = x_train[perm[-1], :, :, :] + np.random.normal(0, 0.05, (10,) + x_train.shape[1:])
>>> suspicious_queries = suspicious_queries.astype(np.float32)
>>> encoder = Model(EncoderNet(encode_dim=256))
>>> detector = SimilarityDetector(max_k_neighbor=3, trans_model=encoder)
>>> num_nearest_neighbors, thresholds = detector.fit(inputs=x_train)
>>> detector.set_threshold(num_nearest_neighbors[-1], thresholds[-1])
>>> detector.detect(benign_queries)
>>> detections = detector.get_detection_interval()
>>> detected_queries = detector.get_detected_queries()
```

clear_buffer()

清除缓冲区内存。

detect(*inputs*)

处理查询以检测黑盒攻击。

参数：

- **inputs** (numpy.ndarray) - 查询序列。

异常：

- **ValueError** - 阈值或 set_threshold 方法中 *num_of_neighbors* 参数不可用。

detect_diff(*inputs*)

从输入样本中检测对抗样本，如常见机器学习模型中的 predict_proba 函数。

参数：

- **inputs** (Union[numpy.ndarray, list, tuple]) - 用于创建对抗样本。

异常：

- **NotImplementedError** - 此函数在 *SimilarityDetector* 类（class）中不可用。

fit(*inputs*, *labels=None*)

处理输入训练数据以计算阈值。适当的阈值应确保假正率低于给定值。

参数：

- **inputs** (numpy.ndarray) - 用于计算阈值的训练数据。
- **labels** (numpy.ndarray) - 训练数据的标签。

返回：

- **list[int]** - 最近邻的数量。
- **list[float]** - 不同 k 的阈值。

异常：

- **ValueError** - 训练数据个数小于 *max_k_neighbor*。

get_detected_queries()

获取检测到的查询的索引。

返回：

- **list[int]** - 检测到的恶意查询的序列号。

get_detection_interval()

获取相邻检测之间的间隔。

返回：

- **list[int]** - 相邻检测之间的查询数。

set_threshold(*num_of_neighbors*, *threshold*)

设置参数 num_of_neighbors 和 threshold。

参数：

- **num_of_neighbors** (int) - 最近邻的数量。
- **threshold** (float) - 检测阈值。

transform(*inputs*)

过滤输入样本中的对抗性噪声。

参数：

- **inputs** (Union[numpy.ndarray, list, tuple]) - 用于创建对抗样本。

异常：

- **NotImplementedError** - 此函数在 *SimilarityDetector* 类（class）中不可用。

CHAPTER

FIVE

MINDARMOUR.ADV_ROBUSTNESS.EVALUATIONS

此模块包括各种指标，用于评估攻击或防御的结果。

class mindarmour.adv_robustness.evaluations.**AttackEvaluate**(*inputs*, *labels*, *adv_inputs*, *adv_preds*, *targeted=False*, *target_label=None*)

攻击方法的评估指标。

参数：

- **inputs** (numpy.ndarray) - 原始样本。
- **labels** (numpy.ndarray) - 原始样本的 one-hot 格式标签。
- **adv_inputs** (numpy.ndarray) - 从原始样本生成的对抗样本。
- **adv_preds** (numpy.ndarray) - 对对抗样本的对所有标签的预测概率。
- **targeted** (bool) - 如果为 True，则为目标攻击。如果为 False，则为无目标攻击。默认值：False。
- **target_label** (numpy.ndarray) - 对抗样本的目标标签，是大小为 adv_inputs.shape[0] 的一维。默认值：None。

异常：

- **ValueError** - 如果 *targeted* 为 True 时，*target_label* 为 None。

样例：

```
>>> from mindarmour.adv_robustness.evaluations import AttackEvaluate
>>> x = np.random.normal(size=(3, 512, 512, 3))
>>> adv_x = np.random.normal(size=(3, 512, 512, 3))
>>> y = np.array([[0.1, 0.1, 0.2, 0.6],
...               [0.1, 0.7, 0.0, 0.2],
...               [0.8, 0.1, 0.0, 0.1]])
>>> adv_y = np.array([[0.1, 0.1, 0.2, 0.6],
...                   [0.1, 0.0, 0.8, 0.1],
...                   [0.0, 0.9, 0.1, 0.0]])
>>> attack_eval = AttackEvaluate(x, y, adv_x, adv_y)
>>> mr = attack_eval.mis_classification_rate()
>>> acac = attack_eval.avg_conf_adv_class()
>>> l_0, l_2, l_inf = attack_eval.avg_lp_distance()
>>> ass = attack_eval.avg_ssim()
>>> nte = attack_eval.nte()
>>> actc = attack_eval.avg_conf_true_class()
```

avg_conf_adv_class()

计算对抗类的平均置信度（ACAC）。

返回：

- **float** - 范围在（0,1）之间。值越高，攻击就越成功。

avg_conf_true_class()

计算真类的平均置信度（ACTC）。

返回：

- **float** - 范围在（0,1）之间。值越低，攻击就越成功。

avg_lp_distance()

计算平均 lp 距离（lp-dist）。

返回：

- **float** - 返回所有成功对抗样本的平均’l0’、’l2’或’linf’距离，返回值包括以下情况：
  - 如果返回值 >= 0，则为平均 lp 距离。值越低，攻击就越成功。
  - 如果返回值为-1，则没有成功的对抗样本。

avg_ssim()

计算平均结构相似性（ASS）。

返回：

- **float** - 平均结构相似性。

- 如果返回值在（0,1）之间，则值越高，攻击越成功。
- 如果返回值为-1，则没有成功的对抗样本。

mis_classification_rate()

计算错误分类率（MR）。

返回：

- **float** - 范围在（0,1）之间。值越高，攻击就越成功。

nte()

计算噪声容量估计（NTE）。

参考文献：Towards Imperceptible and Robust Adversarial Example Attacks against Neural Networks。

返回：

- **float** - 范围在（0,1）之间。值越高，攻击就越成功。

class mindarmour.adv_robustness.evaluations.**BlackDefenseEvaluate**(*raw_preds*, *def_preds*, *raw_query_counts*, *def_query_counts*, *raw_query_time*, *def_query_time*, *def_detection_counts*, *true_labels*, *max_queries*)

反黑盒防御方法的评估指标。

参数：

- **raw_preds** (numpy.ndarray) - 原始模型上特定样本的预测结果。
- **def_preds** (numpy.ndarray) - 原始防御模型上特定样本的预测结果。
- **raw_query_counts** (numpy.ndarray) - 在原始模型上生成对抗样本的查询数，原始模型是大小是与 raw_preds.shape[0] 的第一纬度相同。对于良性样本，查询计数必须设置为 0。
- **def_query_counts** (numpy.ndarray) - 在防御模型上生成对抗样本的查询数，原始模型是大小是与 raw_preds.shape[0] 的第一纬度相同。对于良性样本，查询计数必须设置为 0。
- **raw_query_time** (numpy.ndarray) - 在原始模型上生成对抗样本的总持续时间，该样本是大小是与 raw_preds.shape[0] 的第一纬度。
- **def_query_time** (numpy.ndarray) - 在防御模型上生成对抗样本的总持续时间，该样本是大小是与 raw_preds.shape[0] 的第一纬度。
- **def_detection_counts** (numpy.ndarray) - 每次对抗样本生成期间检测到的查询总数，大小是与 raw_preds.shape[0] 的第一纬度。对于良性样本，如果查询被识别为可疑，则将 def_detection_counts 设置为 1，否则将其设置为 0。
- **true_labels** (numpy.ndarray) - 大小是与 raw_preds.shape[0] 的第一纬度真标签。
- **max_queries** (int) - 攻击预算，最大查询数。

样例：

```
>>> from mindarmour.adv_robustness.evaluations import BlackDefenseEvaluate
>>> raw_preds = np.array([[0.1, 0.1, 0.2, 0.6],
...                       [0.1, 0.7, 0.0, 0.2],
...                       [0.8, 0.1, 0.0, 0.1]])
>>> def_preds = np.array([[0.1, 0.1, 0.1, 0.7],
...                       [0.1, 0.6, 0.2, 0.1],
...                       [0.1, 0.2, 0.1, 0.6]])
>>> raw_query_counts = np.array([0,20,10])
>>> def_query_counts = np.array([0,50,60])
>>> raw_query_time = np.array([0.1, 2, 1])
>>> def_query_time = np.array([0.2, 6, 5])
>>> def_detection_counts = np.array([1, 5, 10])
>>> true_labels = np.array([3, 1, 0])
>>> max_queries = 100
>>> def_eval = BlackDefenseEvaluate(raw_preds,
...                                 def_preds,
...                                 raw_query_counts,
...                                 def_query_counts,
...                                 raw_query_time,
...                                 def_query_time,
...                                 def_detection_counts,
...                                 true_labels,
...                                 max_queries)
>>> qcv = def_eval.qcv()
>>> asv = def_eval.asv()
>>> fpr = def_eval.fpr()
>>> qrv = def_eval.qrv()
```

asv()

计算攻击成功率方差（ASV）。

返回：

- **float** - 值越低，防守就越强。如果 num_adv_samples=0，则返回-1。

fpr()

计算基于查询的检测器的假正率（FPR）。

返回：

- **float** - 值越低，防御的可用性越高。如果 num_adv_samples=0，则返回-1。

qcv()

计算查询计数方差（QCV）。

返回：

- **float** - 值越高，防守就越强。如果 num_adv_samples=0，则返回-1。

qrv()

计算良性查询响应时间方差（QRV）。

返回：

- **float** - 值越低，防御的可用性越高。如果 num_adv_samples=0，则返回-1。

class mindarmour.adv_robustness.evaluations.**DefenseEvaluate**(*raw_preds*, *def_preds*, *true_labels*)

防御方法的评估指标。

参数：

- **raw_preds** (numpy.ndarray) - 原始模型上某些样本的预测结果。
- **def_preds** (numpy.ndarray) - 防御模型上某些样本的预测结果。
- **true_labels** (numpy.ndarray) - 样本的 ground-truth 标签，一个大小为 ground-truth 的一维数组。

样例：

```
>>> from mindarmour.adv_robustness.evaluations import DefenseEvaluate
>>> raw_preds = np.array([[0.1, 0.1, 0.2, 0.6],
...                       [0.1, 0.7, 0.0, 0.2],
...                       [0.8, 0.1, 0.0, 0.1]])
>>> def_preds = np.array([[0.1, 0.1, 0.1, 0.7],
...                       [0.1, 0.6, 0.2, 0.1],
...                       [0.1, 0.2, 0.1, 0.6]])
>>> true_labels = np.array([3, 1, 0])
>>> def_eval = DefenseEvaluate(raw_preds,
...                            def_preds,
...                            true_labels)
>>> cav = def_eval.cav()
>>> crr = def_eval.crr()
>>> csr = def_eval.csr()
>>> ccv = def_eval.ccv()
>>> cos = def_eval.cos()
```

cav()

计算分类精度方差（CAV）。

返回：

- **float** - 值越高，防守就越成功。

ccv()

计算分类置信度方差（CCV）。

返回：

- **float** - 值越低，防守就越成功。如果返回值 == -1，则说明样本数量为 0。

cos()

参考文献：Calculate classification output stability (COS)。

返回：

- **float** - 如果返回值 >=0，则是有效的防御。值越低，防守越成功。如果返回值 == -1, 则说明样本数量为 0。

crr()

计算分类校正率（CRR）。

返回：

- **float** - 值越高，防守就越成功。

csr()

计算分类牺牲比（CSR），越低越好。

返回：

- **float** - 值越低，防守就越成功。

class mindarmour.adv_robustness.evaluations.**RadarMetric**(*metrics_name*, *metrics_data*, *labels*, *title*, *scale='hide'*)

雷达图，通过多个指标显示模型的鲁棒性。

参数：

- **metrics_name** (Union[tuple, list]) - 要显示的度量名称数组。每组值对应一条雷达曲线。
- **metrics_data** (numpy.ndarray) - 多个雷达曲线的每个度量的（归一化）值，如 [[0.5, 0.8, ···], [0.2,0.6,···], ···]。
- **labels** (Union[tuple, list]) - 所有雷达曲线的图例。
- **title** (str) - 图表的标题。
- **scale** (str) - 用于调整轴刻度的标量，如’hide’、’norm’、’sparse’、’dense’。默认值:’hide’。

异常：

- **ValueError** - *scale* 值不在 [‘hide’，‘norm’，‘sparse’，‘dense’] 中。

样例：

```
>>> from mindarmour.adv_robustness.evaluations import RadarMetric
>>> metrics_name = ['MR', 'ACAC', 'ASS', 'NTE', 'ACTC']
>>> def_metrics = [0.9, 0.85, 0.6, 0.7, 0.8]
>>> raw_metrics = [0.5, 0.3, 0.55, 0.65, 0.7]
>>> metrics_data = np.array([def_metrics, raw_metrics])
>>> metrics_labels = ['before', 'after']
>>> rm = RadarMetric(metrics_name,
...                  metrics_data,
...                  metrics_labels,
...                  title='',
...                  scale='sparse')
>>> #rm.show()
```

show()

显示雷达图。

CHAPTER

SIX

MINDARMOUR.FUZZ_TESTING

该模块提供了一种基于神经元覆盖率增益的模糊测试方法来评估给定模型的鲁棒性。

class mindarmour.fuzz_testing.**Fuzzer**(*target_model*)

深度神经网络的模糊测试框架。

参考文献：DeepHunter: A Coverage-Guided Fuzz Testing Framework for Deep Neural Networks。

参数：

- **target_model** (Model) - 目标模糊模型。

样例：

```
>>> from mindspore.common.initializer import TruncatedNormal
>>> from mindspore.ops import operations as P
>>> from mindspore.train import Model
>>> from mindspore.ops import TensorSummary
>>> from mindarmour.fuzz_testing import Fuzzer
>>> from mindarmour.fuzz_testing import KMultisectionNeuronCoverage
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(1, 6, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.conv2 = nn.Conv2d(6, 16, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.fc1 = nn.Dense(16 * 5 * 5, 120, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc2 = nn.Dense(120, 84, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc3 = nn.Dense(84, 10, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.reshape = P.Reshape()
...         self.summary = TensorSummary()
...
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu(x)
...         self.summary('conv1', x)
...         x = self.max_pool2d(x)
...         x = self.conv2(x)
...         x = self.relu(x)
...         self.summary('conv2', x)
...         x = self.max_pool2d(x)
...         x = self.reshape(x, (-1, 16 * 5 * 5))
...         x = self.fc1(x)
...         x = self.relu(x)
...         self.summary('fc1', x)
...         x = self.fc2(x)
...         x = self.relu(x)
...         self.summary('fc2', x)
...         x = self.fc3(x)
...         self.summary('fc3', x)
...         return x
>>> net = Net()
>>> model = Model(net)
>>> mutate_config = [{'method': 'GaussianBlur',
...                   'params': {'ksize': [1, 2, 3, 5], 'auto_param': [True, False]}},
...                  {'method': 'MotionBlur',
...                   'params': {'degree': [1, 2, 5], 'angle': [45, 10, 100, 140, 210, 270, 300],
...                   'auto_param': [True]}},
...                  {'method': 'UniformNoise',
...                   'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
...                  {'method': 'GaussianNoise',
...                   'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
...                  {'method': 'Contrast',
```

(continues on next page)

(continued from previous page)

```
...                   'params': {'alpha': [0.5, 1, 1.5], 'beta': [-10, 0, 10], 'auto_param': [False, True]}},
...                  {'method': 'Rotate',
...                   'params': {'angle': [20, 90], 'auto_param': [False, True]}},
...                  {'method': 'FGSM',
...                   'params': {'eps': [0.3, 0.2, 0.4], 'alpha': [0.1], 'bounds': [(0, 1)]}}]
>>> batch_size = 8
>>> num_classe = 10
>>> train_images = np.random.rand(32, 1, 32, 32).astype(np.float32)
>>> test_images = np.random.rand(batch_size, 1, 32, 32).astype(np.float32)
>>> test_labels = np.random.randint(num_classe, size=batch_size).astype(np.int32)
>>> test_labels = (np.eye(num_classe)[test_labels]).astype(np.float32)
>>> initial_seeds = []
>>> # make initial seeds
>>> for img, label in zip(test_images, test_labels):
...     initial_seeds.append([img, label])
>>> initial_seeds = initial_seeds[:10]
>>> nc = KMultisectionNeuronCoverage(model, train_images, segmented_num=100, incremental=True)
>>> model_fuzz_test = Fuzzer(model)
>>> samples, gt_labels, preds, strategies, metrics = model_fuzz_test.fuzzing(mutate_config, initial_seeds,
...                                                                          nc, max_iters=100)
```

fuzzing(*mutate_config*, *initial_seeds*, *coverage*, *evaluate=True*, *max_iters=10000*, *mutate_num_per_seed=20*)

深度神经网络的模糊测试。

参数：

- **mutate_config (list) - 变异方法配置。格式为：**

```
mutate_config = [
    {'method': 'GaussianBlur',
     'params': {'ksize': [1, 2, 3, 5], 'auto_param': [True, False]}},
    {'method': 'UniformNoise',
     'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
    {'method': 'GaussianNoise',
     'params': {'factor': [0.1, 0.2, 0.3], 'auto_param': [False, True]}},
    {'method': 'Contrast',
     'params': {'alpha': [0.5, 1, 1.5], 'beta': [-10, 0, 10], 'auto_param': [False, True]}},
    {'method': 'Rotate',
     'params': {'angle': [20, 90], 'auto_param': [False, True]}},
    {'method': 'FGSM',
     'params': {'eps': [0.3, 0.2, 0.4], 'alpha': [0.1], 'bounds': [(0, 1)]}}
    ...]
```

  - 支持的方法在列表 *self._strategies* 中，每个方法的参数必须在可选参数的范围内。支持的方法分为两种类型：
  - 首先，自然鲁棒性方法包括：’ Translate’、’ Scale’、’ Shear’、’ Rotate’、’ Perspective’、’ Curve’、’ GaussianBlur’、’ MotionBlur’、’ GradientBlur’、’ Contrast’、’ GradientLuminance’、’ UniformNoise’、’ GaussianNoise’、’ SaltAndPepperNoise’、’ NaturalNoise’。
  - 其次，对抗样本攻击方式包括：’ FGSM’、’ PGD’ 和’ MDIM’。’ FGSM’、’ PGD’ 和’ MDIM’ 分别是 FastGradientSignMethod、ProjectedGradientDent 和 MomentumDiverseInputIterativeMethod 的缩写。*mutate_config* 必须包含在 [‘Contrast’，‘GradientLuminance’，‘GaussianBlur’，‘MotionBlur’，‘GradientBlur’，‘UniformNoise’，‘GaussianNoise’，‘SaltAndPepperNoise’，‘NaturalNoise’] 中的方法。
    - 第一类方法的参数设置方式可以在’mindarmour/natural_robustness/transform/image’中看到。第二类方法参数配置参考 *self._attack_param_checklists* 。

- **initial_seeds** (list[list]) - 用于生成变异样本的初始种子队列。初始种子队列的格式为 [[image_data, label], […], …]，且标签必须为 one-hot。
- **coverage** (CoverageMetrics) - 神经元覆盖率指标类。
- **evaluate** (bool) - 是否返回评估报告。默认值：True。
- **max_iters** (int) - 选择要变异的种子的最大数量。默认值：10000。
- **mutate_num_per_seed** (int) - 每个种子的最大变异次数。默认值：20。

返回：

- **list** - 模糊测试生成的变异样本。
- **list** - 变异样本的 ground truth 标签。
- **list** - 预测结果。
- **list** - 变异策略。
- **dict** - Fuzzer 的指标报告。

异常：

- **ValueError** - 参数 *coverage* 必须是 CoverageMetrics 的子类。

- **ValueError** - 初始种子队列为空。
- **ValueError** - *initial_seeds* 中的种子未包含两个元素。

class mindarmour.fuzz_testing.**CoverageMetrics**(*model*, *incremental=False*, *batch_size=32*)

计算覆盖指标的神经元覆盖类的抽象基类。

训练后网络的每个神经元输出有一个输出范围（我们称之为原始范围），测试数据集用于估计训练网络的准确性。然而，不同的测试数据集，神经元的输出分布会有所不同。因此，与传统模糊测试类似，模型模糊测试意味着测试这些神经元的输出，并评估在测试数据集上神经元输出值占原始范围的比例。

参考文献：DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems。

参数：

- **model** (Model) - 被测模型。
- **incremental** (bool) - 指标将以增量方式计算。默认值：False。
- **batch_size** (int) - 模糊测试批次中的样本数。默认值：32。

get_metrics(*dataset*)

计算给定数据集的覆盖率指标。

参数：

- **dataset** (numpy.ndarray) - 用于计算覆盖指标的数据集。

异常：

- **NotImplementedError** - 抽象方法。

class mindarmour.fuzz_testing.**NeuronCoverage**(*model*, *threshold=0.1*, *incremental=False*, *batch_size=32*)

计算神经元激活的覆盖率。当神经元的输出大于阈值时，神经元被激活。

神经元覆盖率等于网络中激活的神经元占总神经元的比例。

参数：

- **model** (Model) - 被测模型。
- **threshold** (float) - 用于确定神经元是否激活的阈值。默认值：0.1。
- **incremental** (bool) - 指标将以增量方式计算。默认值：False。
- **batch_size** (int) - 模糊测试批次中的样本数。默认值：32。

get_metrics(*dataset*)

获取神经元覆盖率的指标：激活的神经元占网络中神经元总数的比例。

参数：

- **dataset** (numpy.ndarray) - 用于计算覆盖率指标的数据集。

返回：

- **float** - ‘neuron coverage’的指标。

样例：

```
>>> from mindspore.common.initializer import TruncatedNormal
>>> from mindspore.ops import operations as P
>>> from mindspore.train import Model
>>> from mindspore.ops import TensorSummary
>>> from mindarmour.fuzz_testing import NeuronCoverage
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(1, 6, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.conv2 = nn.Conv2d(6, 16, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.fc1 = nn.Dense(16 * 5 * 5, 120, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc2 = nn.Dense(120, 84, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc3 = nn.Dense(84, 10, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.reshape = P.Reshape()
...         self.summary = TensorSummary()
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu(x)
...         self.summary('conv1', x)
...         x = self.max_pool2d(x)
...         x = self.conv2(x)
...         x = self.relu(x)
...         self.summary('conv2', x)
...         x = self.max_pool2d(x)
```

(continues on next page)

(continued from previous page)

```
...         x = self.reshape(x, (-1, 16 * 5 * 5))
...         x = self.fc1(x)
...         x = self.relu(x)
...         self.summary('fc1', x)
...         x = self.fc2(x)
...         x = self.relu(x)
...         self.summary('fc2', x)
...         x = self.fc3(x)
...         self.summary('fc3', x)
...         return x
>>> net = Net()
>>> model = Model(net)
>>> batch_size = 8
>>> num_classe = 10
>>> train_images = np.random.rand(32, 1, 32, 32).astype(np.float32)
>>> test_images = np.random.rand(batch_size, 1, 32, 32).astype(np.float32)
>>> nc = NeuronCoverage(model, threshold=0.1)
>>> nc_metrics = nc.get_metrics(test_images)
```

class mindarmour.fuzz_testing.**TopKNeuronCoverage**(*model*, *top_k=3*, *incremental=False*, *batch_size=32*)

计算前 k 个激活神经元的覆盖率。当隐藏层神经元的输出值在最大的 *top_k* 范围内，神经元就会被激活。*top_k* 神经元覆盖率等于网络中激活神经元占总神经元的比例。

参数：

- **model** (Model) - 被测模型。
- **top_k** (int) - 当隐藏层神经元的输出值在最大的 *top_k* 范围内，神经元就会被激活。默认值：3。
- **incremental** (bool) - 指标将以增量方式计算。默认值：False。
- **batch_size** (int) - 模糊测试批次中的样本数。默认值：32。

get_metrics(*dataset*)

获取 Top K 激活神经元覆盖率的指标。

参数：

- **dataset** (numpy.ndarray) - 用于计算覆盖率指标的数据集。

返回：

- **float** - ‘top k neuron coverage’的指标。

样例：

```
>>> from mindspore.common.initializer import TruncatedNormal
>>> from mindspore.ops import operations as P
>>> from mindspore.train import Model
>>> from mindspore.ops import TensorSummary
>>> from mindarmour.fuzz_testing import TopKNeuronCoverage
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(1, 6, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.conv2 = nn.Conv2d(6, 16, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.fc1 = nn.Dense(16 * 5 * 5, 120, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc2 = nn.Dense(120, 84, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc3 = nn.Dense(84, 10, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.reshape = P.Reshape()
...         self.summary = TensorSummary()
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu(x)
...         self.summary('conv1', x)
...         x = self.max_pool2d(x)
...         x = self.conv2(x)
...         x = self.relu(x)
...         self.summary('conv2', x)
...         x = self.max_pool2d(x)
...         x = self.reshape(x, (-1, 16 * 5 * 5))
...         x = self.fc1(x)
...         x = self.relu(x)
...         self.summary('fc1', x)
...         x = self.fc2(x)
```

(continues on next page)

(continued from previous page)

```
...         x = self.relu(x)
...         self.summary('fc2', x)
...         x = self.fc3(x)
...         self.summary('fc3', x)
...         return x
>>> net = Net()
>>> model = Model(net)
>>> batch_size = 8
>>> num_classe = 10
>>> train_images = np.random.rand(32, 1, 32, 32).astype(np.float32)
>>> test_images = np.random.rand(batch_size, 1, 32, 32).astype(np.float32)
>>> tknc = TopKNeuronCoverage(model, top_k=3)
>>> metrics = tknc.get_metrics(test_images)
```

class mindarmour.fuzz_testing.**NeuronBoundsCoverage**(*model*, *train_dataset*, *incremental=False*, *batch_size=32*)

获取’neuron boundary coverage’的指标 $NBC = (|UpperCornerNeuron| + |LowerCornerNeuron|)/(2 * |N|)$，其中 $|N|$ 是神经元的数量，NBC 是指测试数据集中神经元输出值超过训练数据集中相应神经元输出值的上下界的神经元比例。

参数：

- **model** (Model) - 等待测试的预训练模型。
- **train_dataset** (numpy.ndarray) - 用于确定神经元输出边界的训练数据集。
- **incremental** (bool) - 指标将以增量方式计算。默认值：False。
- **batch_size** (int) - 模糊测试批次中的样本数。默认值：32。

get_metrics(*dataset*)

获取’neuron boundary coverage’的指标。

参数：

- **dataset** (numpy.ndarray) - 用于计算覆盖指标的数据集。

返回：

- **float** - ‘neuron boundary coverage’的指标。

样例：

```
>>> from mindspore.common.initializer import TruncatedNormal
>>> from mindspore.ops import operations as P
>>> from mindspore.train import Model
>>> from mindspore.ops import TensorSummary
>>> from mindarmour.fuzz_testing import NeuronBoundsCoverage
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(1, 6, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.conv2 = nn.Conv2d(6, 16, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.fc1 = nn.Dense(16 * 5 * 5, 120, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc2 = nn.Dense(120, 84, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc3 = nn.Dense(84, 10, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.reshape = P.Reshape()
...         self.summary = TensorSummary()
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu(x)
...         self.summary('conv1', x)
...         x = self.max_pool2d(x)
...         x = self.conv2(x)
...         x = self.relu(x)
...         self.summary('conv2', x)
...         x = self.max_pool2d(x)
...         x = self.reshape(x, (-1, 16 * 5 * 5))
...         x = self.fc1(x)
...         x = self.relu(x)
...         self.summary('fc1', x)
...         x = self.fc2(x)
...         x = self.relu(x)
...         self.summary('fc2', x)
...         x = self.fc3(x)
...         self.summary('fc3', x)
...         return x
```

(continues on next page)

(continued from previous page)

```
>>> net = Net()
>>> model = Model(net)
>>> batch_size = 8
>>> num_classe = 10
>>> train_images = np.random.rand(32, 1, 32, 32).astype(np.float32)
>>> test_images = np.random.rand(batch_size, 1, 32, 32).astype(np.float32)
>>> nbc = NeuronBoundsCoverage(model, train_images)
>>> metrics = nbc.get_metrics(test_images)
```

class mindarmour.fuzz_testing.**SuperNeuronActivateCoverage**(*model*, *train_dataset*, *incremental=False*, *batch_size=32*)

获取超激活神经元覆盖率（’super neuron activation coverage’）的指标。$SNAC = |UpperCornerNeuron|/|N|$ 。SNAC 是指测试集中神经元输出值超过训练集中相应神经元输出值上限的神经元比例。

参数：

- **model** (Model) - 等待测试的预训练模型。
- **train_dataset** (numpy.ndarray) - 用于确定神经元输出边界的训练数据集。
- **incremental** (bool) - 指标将以增量方式计算。默认值：False。
- **batch_size** (int) - 模糊测试批次中的样本数。默认值：32。

get_metrics(*dataset*)

获取超激活神经元覆盖率（’super neuron activation coverage’）的指标。

参数：

- **dataset** (numpy.ndarray) - 用于计算覆盖指标的数据集。

返回：

- **float** - 超激活神经元覆盖率（’super neuron activation coverage’）的指标

样例：

```
>>> from mindspore.common.initializer import TruncatedNormal
>>> from mindspore.ops import operations as P
>>> from mindspore.train import Model
>>> from mindspore.ops import TensorSummary
>>> from mindarmour.fuzz_testing import SuperNeuronActivateCoverage
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(1, 6, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.conv2 = nn.Conv2d(6, 16, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.fc1 = nn.Dense(16 * 5 * 5, 120, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc2 = nn.Dense(120, 84, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc3 = nn.Dense(84, 10, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.reshape = P.Reshape()
...         self.summary = TensorSummary()
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu(x)
...         self.summary('conv1', x)
...         x = self.max_pool2d(x)
...         x = self.conv2(x)
...         x = self.relu(x)
...         self.summary('conv2', x)
...         x = self.max_pool2d(x)
...         x = self.reshape(x, (-1, 16 * 5 * 5))
...         x = self.fc1(x)
...         x = self.relu(x)
...         self.summary('fc1', x)
...         x = self.fc2(x)
...         x = self.relu(x)
...         self.summary('fc2', x)
...         x = self.fc3(x)
...         self.summary('fc3', x)
...         return x
>>> net = Net()
>>> model = Model(net)
>>> batch_size = 8
>>> num_classe = 10
>>> train_images = np.random.rand(32, 1, 32, 32).astype(np.float32)
```

(continues on next page)

(continued from previous page)

```
>>> test_images = np.random.rand(batch_size, 1, 32, 32).astype(np.float32)
>>> snac = SuperNeuronActivateCoverage(model, train_images)
>>> metrics = snac.get_metrics(test_images)
```

class mindarmour.fuzz_testing.**KMultisectionNeuronCoverage**(*model*, *train_dataset*, *segmented_num=100*, *incremental=False*, *batch_size=32*)

获取 K 分神经元覆盖率的指标。KMNC 度量测试集神经元输出落在训练集输出范围 k 等分间隔上的比例。

参数：

- **model** (Model) - 等待测试的预训练模型。
- **train_dataset** (numpy.ndarray) - 用于确定神经元输出边界的训练数据集。
- **segmented_num** (int) - 神经元输出间隔的分段部分数量。默认值：100。
- **incremental** (bool) - 指标将以增量方式计算。默认值：False。
- **batch_size** (int) - 模糊测试批次中的样本数。默认值：32。

get_metrics(*dataset*)

获取’k-multisection neuron coverage’的指标。

参数：

- **dataset** (numpy.ndarray) - 用于计算覆盖指标的数据集。

返回：

- **float** - ‘k-multisection neuron coverage’的指标。

样例：

```
>>> from mindspore.common.initializer import TruncatedNormal
>>> from mindspore.ops import operations as P
>>> from mindspore.train import Model
>>> from mindspore.ops import TensorSummary
>>> from mindarmour.fuzz_testing import KMultisectionNeuronCoverage
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self.conv1 = nn.Conv2d(1, 6, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.conv2 = nn.Conv2d(6, 16, 5, padding=0, weight_init=TruncatedNormal(0.02), pad_mode="valid")
...         self.fc1 = nn.Dense(16 * 5 * 5, 120, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc2 = nn.Dense(120, 84, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.fc3 = nn.Dense(84, 10, TruncatedNormal(0.02), TruncatedNormal(0.02))
...         self.relu = nn.ReLU()
...         self.max_pool2d = nn.MaxPool2d(kernel_size=2, stride=2)
...         self.reshape = P.Reshape()
...         self.summary = TensorSummary()
...     def construct(self, x):
...         x = self.conv1(x)
...         x = self.relu(x)
...         self.summary('conv1', x)
...         x = self.max_pool2d(x)
...         x = self.conv2(x)
...         x = self.relu(x)
...         self.summary('conv2', x)
...         x = self.max_pool2d(x)
...         x = self.reshape(x, (-1, 16 * 5 * 5))
...         x = self.fc1(x)
...         x = self.relu(x)
...         self.summary('fc1', x)
...         x = self.fc2(x)
...         x = self.relu(x)
...         self.summary('fc2', x)
...         x = self.fc3(x)
...         self.summary('fc3', x)
...         return x
>>> net = Net()
>>> model = Model(net)
>>> batch_size = 8
>>> num_classe = 10
>>> train_images = np.random.rand(32, 1, 32, 32).astype(np.float32)
>>> test_images = np.random.rand(batch_size, 1, 32, 32).astype(np.float32)
>>> kmnc = KMultisectionNeuronCoverage(model, train_images, segmented_num=100)
>>> metrics = kmnc.get_metrics(test_images)
```

CHAPTER

SEVEN

MINDARMOUR.NATURAL_ROBUSTNESS.TRANSFORM.IMAGE

This package include methods to generate natural perturbation samples.

class mindarmour.natural_robustness.transform.image.**Translate**(*x_bias=0*, *y_bias=0*, *auto_param=False*)

Translate an image.

Parameters

- **x_bias** (*Union[int, float]*) –X-direction translation, x = x + x_bias*image_width. Suggested value range in [-0.1, 0.1].
- **y_bias** (*Union[int, float]*) –Y-direction translation, y = y + y_bias*image_length. Suggested value range in [-0.1, 0.1].
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> x_bias = 0.1
>>> y_bias = 0.1
>>> trans = Translate(x_bias, y_bias)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**Scale**(*factor_x=1*, *factor_y=1*, *auto_param=False*)

Scale an image in the middle.

Parameters

- **factor_x** (*Union[float, int]*) –Rescale in X-direction, x=factor_x*x. Suggested value range in [0.5, 1] and abs(factor_y - factor_x) < 0.5.
- **factor_y** (*Union[float, int]*) –Rescale in Y-direction, y=factor_y*y. Suggested value range in [0.5, 1] and abs(factor_y - factor_x) < 0.5.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> factor_x = 0.7
>>> factor_y = 0.6
>>> trans = Scale(factor_x, factor_y)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**Shear**(*factor=0.2*, *direction="horizontal"*, *auto_param=False*)

Shear an image, the mapping between sheared image and origin image is (new_x, new_y) = (x+factor_x*y, factor_y*x+y). Then the sheared image will be rescaled to fit original size.

Parameters

- **factor** (*Union[float, int]*) –Shear rate in shear direction. Suggested value range in [0.05, 0.5].
- **direction** (*str*) –Direction of deformation. Optional value is ‘vertical’ or ‘horizontal’.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> factor = 0.2
>>> trans = Shear(factor, direction='horizontal')
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**Rotate**(*angle=20*, *auto_param=False*)

Rotate an image of counter clockwise around its center.

Parameters

- **angle** (*Union[float, int]*) –Degrees of counter clockwise. Suggested value range in [-60, 60].
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> angle = 20
>>> trans = Rotate(angle)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**Perspective**(*ori_pos*, *dst_pos*, *auto_param=False*)

Perform perspective transformation on a given picture.

Parameters

- **ori_pos** (*list*) –Four points in original image.
- **dst_pos** (*list*) –The point coordinates of the 4 points in ori_pos after perspective transformation.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> ori_pos = [[0, 0], [0, 800], [800, 0], [800, 800]]
>>> dst_pos = [[50, 0], [0, 800], [780, 0], [800, 800]]
>>> trans = Perspective(ori_pos, dst_pos)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**Curve**(*curves=3*, *depth=10*, *mode="vertical"*, *auto_param=False*)

Curve picture using sin method.

Parameters

- **curves** (*union[float, int]*) –Number of curve cycles. Suggested value range in [0.1, 5].
- **depth** (*union[float, int]*) –Amplitude of sin method. Suggested value not exceed 1/10 of the length of the picture.
- **mode** (*str*) –Direction of deformation. Optional value is ‘vertical’ or ‘horizontal’.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('x.png')
>>> curves =1
>>> depth = 10
>>> trans = Curve(curves, depth, mode='vertical')
>>> img_new = trans(img)
```

class mindarmour.natural_robustness.transform.image.**GaussianBlur**(*ksize=2*, *auto_param=False*)

Blurs the image using Gaussian blur filter.

Parameters

- **ksize** (*int*) –Size of gaussian kernel, this value must be non-negnative.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> ksize = 5
>>> trans = GaussianBlur(ksize)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**MotionBlur**(*degree=5*, *angle=45*, *auto_param=False*)

Motion blur for a given image.

Parameters

- **degree** (*int*) –Degree of blur. This value must be positive. Suggested value range in [1, 15].
- **angle** (*union[float, int]*) –Direction of motion blur. Angle=0 means up and down motion blur. Angle is counterclockwise.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> angle = 0
>>> degree = 5
>>> trans = MotionBlur(degree=degree, angle=angle)
>>> new_img = trans(img)
```

class mindarmour.natural_robustness.transform.image.**GradientBlur**(*point*, *kernel_num=3*, *center=True*, *auto_param=False*)

Gradient blur.

Parameters

- **point** (*union[tuple, list]*) –2D coordinate of the Blur center point.
- **kernel_num** (*int*) –Number of blur kernels. Suggested value range in [1, 8].
- **center** (*bool*) –Blurred or clear at the center of a specified point.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('xx.png')
>>> img = np.array(img)
>>> number = 5
>>> h, w = img.shape[:2]
>>> point = (int(h / 5), int(w / 5))
>>> center = True
>>> trans = GradientBlur(point, number,  center)
>>> new_img = trans(img)
```

class mindarmour.natural_robustness.transform.image.**Contrast**(*alpha=1*, *beta=0*, *auto_param=False*)

Contrast of an image.

Parameters

- **alpha** (*Union[float, int]*) –Control the contrast of an image. $out\_image = in\_image * alpha + beta$. Suggested value range in [0.2, 2].
- **beta** (*Union[float, int]*) –Delta added to alpha. Default: 0.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> alpha = 0.1
>>> beta = 1
>>> trans = Contrast(alpha, beta)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**GradientLuminance**(*color_start=(0, 0, 0)*, *color_end=(255, 255, 255)*, *start_point=(10, 10)*, *scope=0.5*, *pattern="light"*, *bright_rate=0.3*, *mode="circle"*, *auto_param=False*)

Gradient adjusts the luminance of picture.

Parameters

- **color_start** (*union[tuple, list]*) –Color of gradient center. Default:(0, 0, 0).
- **color_end** (*union[tuple, list]*) –Color of gradient edge. Default:(255, 255, 255).
- **start_point** (*union[tuple, list]*) –2D coordinate of gradient center.
- **scope** (*float*) –Range of the gradient. A larger value indicates a larger gradient range. Default: 0.3.
- **pattern** (*str*) –Dark or light, this value must be in [‘light’, ‘dark’].
- **bright_rate** (*float*) –Control brightness. A larger value indicates a larger gradient range. If parameter ‘pattern’ is ‘light’, Suggested value range in [0.1, 0.7], if parameter ‘pattern’ is ‘dark’, Suggested value range in [0.1, 0.9].
- **mode** (*str*) –Gradient mode, value must be in [‘circle’, ‘horizontal’, ‘vertical’].
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('x.png')
>>> height, width = img.shape[:2]
>>> point = (height // 4, width // 2)
>>> start = (255, 255, 255)
>>> end = (0, 0, 0)
>>> scope = 0.3
>>> pattern='light'
>>> bright_rate = 0.3
>>> trans = GradientLuminance(start, end, point, scope, pattern, bright_rate, mode='circle')
>>> img_new = trans(img)
```

class mindarmour.natural_robustness.transform.image.**UniformNoise**(*factor=0.1*, *auto_param=False*)

Add uniform noise of an image.

Parameters

- **factor** (*float*) –Noise density, the proportion of noise points per unit pixel area. Suggested value range in [0.001, 0.15].
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> factor = 0.1
>>> trans = UniformNoise(factor)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**GaussianNoise**(*factor=0.1*, *auto_param=False*)

Add gaussian noise of an image.

Parameters

- **factor** (*float*) –Noise density, the proportion of noise points per unit pixel area. Suggested value range in [0.001, 0.15].
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> factor = 0.1
>>> trans = GaussianNoise(factor)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**SaltAndPepperNoise**(*factor=0*, *auto_param=False*)

Add salt and pepper noise of an image.

Parameters

- **factor** (*float*) –Noise density, the proportion of noise points per unit pixel area. Suggested value range in [0.001, 0.15].
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('1.png')
>>> img = np.array(img)
>>> factor = 0.1
>>> trans = SaltAndPepperNoise(factor)
>>> dst = trans(img)
```

class mindarmour.natural_robustness.transform.image.**NaturalNoise**(*ratio=0.0002*, *k_x_range=(1*, *5)*, *k_y_range=(3*, *25)*, *auto_param=False*)

Add natural noise to an image.

Parameters

- **ratio** (*float*) –Noise density, the proportion of noise blocks per unit pixel area. Suggested value range in [0.00001, 0.001].
- **k_x_range** (*union[list, tuple]*) –Value range of the noise block length.
- **k_y_range** (*union[list, tuple]*) –Value range of the noise block width.
- **auto_param** (*bool*) –Auto selected parameters. Selected parameters will preserve semantics of image. Default: False.

Examples

```
>>> img = cv2.imread('xx.png')
>>> img = np.array(img)
>>> ratio = 0.0002
>>> k_x_range = (1, 5)
>>> k_y_range = (3, 25)
>>> trans = NaturalNoise(ratio, k_x_range, k_y_range)
>>> new_img = trans(img)
```

CHAPTER

EIGHT

MINDARMOUR.PRIVACY.DIFF_PRIVACY

本模块提供差分隐私功能，以保护用户隐私。

class mindarmour.privacy.diff_privacy.**NoiseGaussianRandom**(*norm_bound=1.0*, *initial_noise_multiplier=1.0*, *seed=0*, *decay_policy=None*)

基于 $mean = 0$ 以及 $standard\_deviation = norm\_bound * initial\_noise\_multiplier$ 的高斯分布产生噪声。

参数：

- **norm_bound** (float) - 梯度的 l2 范数的裁剪范围。默认值：1.0。
- **initial_noise_multiplier** (float) - 高斯噪声标准偏差除以 *norm_bound* 的比率，将用于计算隐私预算。默认值：1.0。
- **seed** (int) - 原始随机种子，如果 seed=0 随机正态将使用安全随机数。如果 seed!=0 随机正态将使用给定的种子生成值。默认值：0。
- **decay_policy** (str) - 衰减策略。默认值：None。

样例：

```
>>> from mindspore import Tensor
>>> from mindspore.common import dtype as mstype
>>> from mindarmour.privacy.diff_privacy import NoiseGaussianRandom
>>> gradients = Tensor([0.2, 0.9], mstype.float32)
>>> norm_bound = 0.1
>>> initial_noise_multiplier = 1.0
>>> seed = 0
>>> decay_policy = None
>>> net = NoiseGaussianRandom(norm_bound, initial_noise_multiplier, seed, decay_policy)
>>> res = net(gradients)
```

construct(*gradients*)

产生高斯噪声。

参数：

- **gradients** (Tensor) - 梯度。

返回：

- **Tensor** - 生成的 shape 与给定梯度相同的噪声。

class mindarmour.privacy.diff_privacy.**NoiseAdaGaussianRandom**(*norm_bound=1.0*, *initial_noise_multiplier=1.0*, *seed=0*, *noise_decay_rate=6e-6*, *decay_policy='Exp'*)

自适应高斯噪声产生机制。噪音会随着训练而衰减。衰减模式可以是’Time’、’Step’、’Exp’。在模型训练过程中，将更新 *self._noise_multiplier* 。

参数：

- **norm_bound** (float) - 梯度的 l2 范数的裁剪范围。默认值：1.0。
- **initial_noise_multiplier** (float) - 高斯噪声标准偏差除以 *norm_bound* 的比率，将用于计算隐私预算。默认值：1.0。
- **seed** (int) - 原始随机种子，如果 seed=0 随机正态将使用安全随机数。如果 seed!=0 随机正态将使用给定的种子生成值。默认值：0。
- **noise_decay_rate** (float) - 控制噪声衰减的超参数。默认值：6e-6。
- **decay_policy** (str) - 噪声衰减策略包括’Step’、’Time’、’Exp’。默认值:’Exp’。

样例：

```
>>> from mindspore import Tensor
>>> from mindspore.common import dtype as mstype
>>> from mindarmour.privacy.diff_privacy import NoiseAdaGaussianRandom
>>> gradients = Tensor([0.2, 0.9], mstype.float32)
>>> norm_bound = 1.0
>>> initial_noise_multiplier = 1.0
>>> seed = 0
>>> noise_decay_rate = 6e-6
>>> decay_policy = "Exp"
>>> net = NoiseAdaGaussianRandom(norm_bound, initial_noise_multiplier, seed, noise_decay_rate, decay_policy)
>>> res = net(gradients)
```

construct(*gradients*)

生成自适应高斯噪声。

参数：

- **gradients** (Tensor) - 梯度。

返回：

- **Tensor** - 生成的 shape 与给定梯度相同的噪声。

class mindarmour.privacy.diff_privacy.**AdaClippingWithGaussianRandom**(*decay_policy='Linear'*, *learning_rate=0.001*, *target_unclipped_quantile=0.9*, *fraction_stddev=0.01*, *seed=0*)

自适应剪裁。如果 *decay_policy* 是’Linear’，则更新公式为：$norm\_bound = norm\_bound - learning\_rate * (beta - target\_unclipped\_quantile)$ 。

如果 *decay_policy* 是’Geometric’，则更新公式为 $norm\_bound = norm\_bound * exp(-learning\_rate * (empirical\_fraction - target\_unclipped\_quantile))$ 。

其中，beta 是值最多为 *target_unclipped_quantile* 的样本的经验分数。

参数：

- **decay_policy** (str) - 自适应剪裁的衰变策略，*decay_policy* 必须在 [‘Linear’，‘Geometric’] 中。默认值：’Linear’。
- **learning_rate** (float) - 更新范数裁剪的学习率。默认值：0.001。
- **target_unclipped_quantile** (float) - 范数裁剪的目标分位数。默认值：0.9。
- **fraction_stddev** (float) - 高斯正态的 stddev，用于 *empirical_fraction* ，公式为 empirical_fraction + N(0, fraction_stddev)。默认值：0.01。
- **seed** (int) - 原始随机种子，如果 seed=0 随机正态将使用安全随机数。如果 seed!=0 随机正态将使用给定的种子生成值。默认值：0。

返回：

- **Tensor** - 更新后的梯度裁剪阈值。

样例：

```
>>> from mindspore import Tensor
>>> from mindspore.common import dtype as mstype
>>> from mindarmour.privacy.diff_privacy import AdaClippingWithGaussianRandom
>>> decay_policy = 'Linear'
>>> beta = Tensor(0.5, mstype.float32)
>>> norm_bound = Tensor(1.0, mstype.float32)
>>> beta_stddev = 0.01
>>> learning_rate = 0.001
>>> target_unclipped_quantile = 0.9
>>> ada_clip = AdaClippingWithGaussianRandom(decay_policy=decay_policy,
...                                          learning_rate=learning_rate,
...                                          target_unclipped_quantile=target_unclipped_quantile,
...                                          fraction_stddev=beta_stddev)
>>> next_norm_bound = ada_clip(beta, norm_bound)
```

construct(*empirical_fraction*, *norm_bound*)

更新 *norm_bound* 的值。

参数：

- **empirical_fraction** (Tensor) - 梯度裁剪的经验分位数, 最大值不超过 *target_unclipped_quantile* 。
- **norm_bound** (Tensor) - 梯度的 l2 范数的裁剪范围。

返回：

- **Tensor** - 生成的 shape 与给定梯度相同的噪声。

class mindarmour.privacy.diff_privacy.**NoiseMechanismsFactory**

噪声产生机制的工厂类。它目前支持高斯随机噪声（Gaussian Random Noise）和自适应高斯随机噪声（Adaptive Gaussian Random Noise）。

详情请查看：教程。

create(*mech_name*, *norm_bound=1.0*, *initial_noise_multiplier=1.0*, *seed=0*, *noise_decay_rate=6e-6*, *decay_policy=None*)

参数：

- **mech_name** (str) - 噪声生成策略，可以是’Gaussian’或’AdaGaussian’。噪声在’AdaGaussian’机制下衰减，而在’Gaussian’机制下则恒定。
- **norm_bound** (float) - 梯度的 l2 范数的裁剪范围。默认值：1.0。
- **initial_noise_multiplier** (float) - 高斯噪声标准偏差除以 *norm_bound* 的比率，将用于计算隐私预算。默认值：1.0。
- **seed** (int) - 原始随机种子，如果 seed=0 随机正态将使用安全随机数。如果 seed!=0 随机正态将使用给定的种子生成值。默认值：0。
- **noise_decay_rate** (float) - 控制噪声衰减的超参数。默认值：6e-6。
- **decay_policy** (str) - 衰减策略。如果 decay_policy 为 None，则不需要更新参数。默认值：None。

异常：

- **NameError** - *mech_name* 必须在 [‘Gaussian’，‘AdaGaussian’] 中。

返回：

- **Mechanisms** - 产生的噪声类别机制。

样例：

```
>>> from mindarmour.privacy.diff_privacy import NoiseMechanismsFactory
>>> norm_bound = 1.0
>>> initial_noise_multiplier = 1.0
>>> noise_mechanism = NoiseMechanismsFactory()
>>> clip = noise_mechanism.create('Gaussian',
...                               norm_bound=norm_bound,
...                               initial_noise_multiplier=initial_noise_multiplier)
```

class mindarmour.privacy.diff_privacy.**ClipMechanismsFactory**

梯度剪裁机制的工厂类。它目前支持高斯随机噪声（Gaussian Random Noise）的自适应剪裁（Adaptive Clipping）。

详情请查看：教程。

create(*mech_name*, *decay_policy='Linear'*, *learning_rate=0.001*, *target_unclipped_quantile=0.9*, *fraction_stddev=0.01*, *seed=0*)

参数：

- **mech_name** (str) - 噪声裁剪生成策略，现支持’Gaussian’。
- **decay_policy** (str) - 自适应剪裁的衰变策略，decay_policy 必须在 [‘Linear’，‘Geometric’] 中。默认值：Linear。
- **learning_rate** (float) - 更新范数裁剪的学习率。默认值：0.001。
- **target_unclipped_quantile** (float) - 范数裁剪的目标分位数。默认值：0.9。
- **fraction_stddev** (float) - 高斯正态的 stddev，用于 empirical_fraction，公式为 $empirical\_fraction + N(0, fraction\_stddev)$ 。默认值：0.01。
- **seed** (int) - 原始随机种子，如果 seed=0 随机正态将使用安全随机数。如果 seed!=0 随机正态将使用给定的种子生成值。默认值：0。

异常：

- **NameError** - *mech_name* 必须在 [‘Gaussian’] 中。

返回：

- **Mechanisms** - 产生的噪声类别机制。

样例：

```
>>> from mindspore import Tensor
>>> from mindspore.common import dtype as mstype
>>> from mindarmour.privacy.diff_privacy import ClipMechanismsFactory
>>> decay_policy = 'Linear'
>>> beta = Tensor(0.5, mstype.float32)
>>> norm_bound = Tensor(1.0, mstype.float32)
>>> beta_stddev = 0.01
>>> learning_rate = 0.001
>>> target_unclipped_quantile = 0.9
>>> clip_mechanism = ClipMechanismsFactory()
>>> ada_clip = clip_mechanism.create('Gaussian',
...                                  decay_policy=decay_policy,
...                                  learning_rate=learning_rate,
...                                  target_unclipped_quantile=target_unclipped_quantile,
...                                  fraction_stddev=beta_stddev)
>>> next_norm_bound = ada_clip(beta, norm_bound)
```

class mindarmour.privacy.diff_privacy.**PrivacyMonitorFactory**

DP 训练隐私监视器的工厂类。

详情请查看：教程。

create(*policy*, **args*, ***kwargs*)

创建隐私预算监测类。

参数：

- **policy** (str) - 监控策略，现支持’rdp’和’zcdp’。
  - 如果策略为’rdp’，监控器将根据 Renyi 差分隐私（Renyi differential privacy，RDP）理论计算 DP 训练的隐私预算；
  - 如果策略为’zcdp’，监控器将根据零集中差分隐私（zero-concentrated differential privacy，zCDP）理论计算 DP 训练的隐私预算。注意，’zcdp’不适合子采样噪声机制。
- **args** (Union[int, float, numpy.ndarray, list, str]) - 用于创建隐私监视器的参数。
- **kwargs** (Union[int, float, numpy.ndarray, list, str]) - 用于创建隐私监视器的关键字参数。

返回：

- **Callback** - 隐私监视器。

样例：

```
>>> from mindarmour.privacy.diff_privacy import PrivacyMonitorFactory
>>> rdp = PrivacyMonitorFactory.create(policy='rdp', num_samples=60000, batch_size=32)
```

class mindarmour.privacy.diff_privacy.**RDPMonitor**(*num_samples*, *batch_size*, *initial_noise_multiplier=1.5*, *max_eps=10.0*, *target_delta=1e-3*, *max_delta=None*, *target_eps=None*, *orders=None*, *noise_decay_mode='Time'*, *noise_decay_rate=6e-4*, *per_print_times=50*, *dataset_sink_mode=False*)

基于 Renyi 差分隐私（RDP）理论，计算 DP 训练的隐私预算。根据下面的参考文献，如果随机化机制被认为具有 α 阶的 ε'-Renyi 差分隐私，它也满足常规差分隐私 (ε, δ)，如下所示：

$$(\varepsilon' + \frac{log(1/\delta)}{\alpha - 1}, \delta)$$

详情请查看：教程。

参考文献：Rényi Differential Privacy of the Sampled Gaussian Mechanism。

参数：

- **num_samples** (int) - 训练数据集中的样本总数。
- **batch_size** (int) - 训练时批处理中的样本数。
- **initial_noise_multiplier** (Union[float, int]) - 高斯噪声标准偏差除以 norm_bound 的比率，将用于计算隐私预算。默认值：1.5。
- **max_eps** (Union[float, int, None]) - DP 训练的最大可接受 epsilon 预算，用于估计最大训练 epoch。'None' 表示 epsilon 预算没有限制。默认值：10.0。
- **target_delta** (Union[float, int, None]) - DP 训练的目标 delta 预算。如果 *target_delta* 设置为 δ，则隐私预算 δ 将在整个训练过程中是固定的。默认值：1e-3。
- **max_delta** (Union[float, int, None]) - DP 训练的最大可接受 delta 预算，用于估计最大训练 epoch。*max_delta* 必须小于 1，建议小于 1e-3，否则会溢出。'None' 表示 delta 预算没有限制。默认值：None。
- **target_eps** (Union[float, int, None]) - DP 训练的目标 epsilon 预算。如果 target_eps 设置为 ε，则隐私预算 ε 将在整个训练过程中是固定的。默认值：None。
- **orders** (Union[None, list[int, float]]) - 用于计算 rdp 的有限阶数，必须大于 1。不同阶的隐私预算计算结果会有所不同。为了获得更严格（更小）的隐私预算估计，可以尝试阶列表。默认值：None。
- **noise_decay_mode** (Union[None, str]) - 训练时添加噪音的衰减模式，可以是 None、'Time'、'Step'、'Exp'。默认值：'Time'。
- **noise_decay_rate** (float) - 训练时噪音的衰变率。默认值：6e-4。
- **per_print_times** (int) - 计算和打印隐私预算的间隔步数。默认值：50。
- **dataset_sink_mode** (bool) - 如果为 True，所有训练数据都将一次性传递到设备（Ascend）。如果为 False，则训练数据将在每步训练后传递到设备。默认值：False。

样例：

```
>>> from mindarmour.privacy.diff_privacy import PrivacyMonitorFactory
>>> rdp = PrivacyMonitorFactory.create(policy='rdp', num_samples=100, batch_size=32)
```

max_epoch_suggest()

估计最大训练 epoch，以满足预定义的隐私预算。

返回：

- **int** - 建议的最大训练 epoch。

step_end(*run_context*)

在每个训练步骤后计算隐私预算。

参数：

- **run_context** (RunContext) - 包含模型的一些信息。

class mindarmour.privacy.diff_privacy.**ZCDPMonitor**(*num_samples*, *batch_size*, *initial_noise_multiplier=1.5*, *max_eps=10.0*, *target_delta=1e-3*, *noise_decay_mode='Time'*, *noise_decay_rate=6e-4*, *per_print_times=50*, *dataset_sink_mode=False*)

基于零集中差分隐私（zCDP）理论，计算 DP 训练的隐私预算。根据下面的参考文献，如果随机化机制满足 ρ-zCDP 机制，它也满足传统的差分隐私（ε, δ），如下所示：

$$(\rho + 2\sqrt{\rho * log(1/\delta)}, \delta)$$

注意，ZCDPMonitor 不适合子采样噪声机制（如 NoiseAdaGaussianRandom 和 NoiseGaussianRandom）。未来将开发 zCDP 的匹配噪声机制。

详情请查看：教程。

参考文献：Concentrated Differentially Private Gradient Descent with Adaptive per-Iteration Privacy Budget。

参数：

- **num_samples** (int) - 训练数据集中的样本总数。
- **batch_size** (int) - 训练时批处理中的样本数。
- **initial_noise_multiplier** (Union[float, int]) - 高斯噪声标准偏差除以 norm_bound 的比率，将用于计算隐私预算。默认值：1.5。

- **max_eps** (Union[float, int]) - DP 训练的最大可接受 epsilon 预算，用于估计最大训练 epoch。默认值：10.0。
- **target_delta** (Union[float, int]) - DP 训练的目标 delta 预算。如果 *target_delta* 设置为 δ，则隐私预算 δ 将在整个训练过程中是固定的。默认值：1e-3。
- **noise_decay_mode** (Union[None, str]) - 训练时添加噪音的衰减模式，可以是 None、’Time’、’Step’、’Exp’。默认值：’Time’。
- **noise_decay_rate** (float) - 训练时噪音的衰变率。默认值：6e-4。
- **per_print_times** (int) - 计算和打印隐私预算的间隔步数。默认值：50。
- **dataset_sink_mode** (bool) - 如果为 True，所有训练数据都将一次性传递到设备（Ascend）。如果为 False，则训练数据将在每步训练后传递到设备。默认值：False。

样例：

```
>>> from mindarmour.privacy.diff_privacy import PrivacyMonitorFactory
>>> zcdp = PrivacyMonitorFactory.create(policy='zcdp',
...                                     num_samples=100,
...                                     batch_size=32,
...                                     initial_noise_multiplier=1.5)
```

max_epoch_suggest()

估计最大训练 epoch，以满足预定义的隐私预算。

返回：

- **int** - 建议的最大训练 epoch。

step_end(*run_context*)

在每个训练步骤后计算隐私预算。

参数：

- **run_context** (RunContext) - 包含模型的一些信息。

class mindarmour.privacy.diff_privacy.**DPOptimizerClassFactory**(*micro_batches=2*)

优化器的工厂类。

参数：

- **micro_batches** (int) - 从原始批次拆分的小批次中的样本数量。默认值：2。

返回：

- **Optimizer** - 优化器类。

样例：

```
>>> from mindarmour.privacy.diff_privacy import DPOptimizerClassFactory
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._relu = nn.ReLU()
...     def construct(self, inputs):
...         out = self._relu(inputs)
...         return out
>>> network = Net()
>>> GaussianSGD = DPOptimizerClassFactory(micro_batches=2)
>>> GaussianSGD.set_mechanisms('Gaussian', norm_bound=1.0, initial_noise_multiplier=1.5)
>>> net_opt = GaussianSGD.create('Momentum')(params=network.trainable_params(),
...                                          learning_rate=0.001,
...                                          momentum=0.9)
```

create(*policy*)

创建 DP 优化器。策略可以是’sgd’、’momentum’、’adam’。

参数：

- **policy** (str) - 选择原始优化器类型。

返回：

- **Optimizer** - 一个带有差分加噪的优化器。

set_mechanisms(*policy*, **args*, ***kwargs*)

获取噪音机制对象。策略可以是’Gaussian’或’AdaGaussian’。候选的 args 和 kwargs 可以在 mechanisms.py 的 `NoiseMechanismsFactory` 类中看到。

参数：

- **policy** (str) - 选择机制类型。

class mindarmour.privacy.diff_privacy.**DPModel**(*micro_batches=2*, *norm_bound=1.0*, *noise_mech=None*, *clip_mech=None*, *optimizer=nn.Momentum*, ***kwargs*)

DPModel 用于构建差分隐私训练的模型。

这个类重载自 `mindspore.Model` 。

详情请查看：教程。

参数：

- **micro_batches** (int) - 从原始批次拆分的小批次数。默认值：2。
- **norm_bound** (float) - 用于裁剪范围，如果设置为 1，将返回原始数据。默认值：1.0。
- **noise_mech** (Mechanisms) - 用于生成不同类型的噪音。默认值：None。
- **clip_mech** (Mechanisms) - 用于更新自适应剪裁。默认值：None。
- **optimizer** (Cell) - 用于更新差分隐私训练过程中的模型权重值。默认值：nn.Momentum。

异常：

- **ValueError** - optimizer 值为 None。
- **ValueError** - optimizer 不是 DPOptimizer，且 noise_mech 为 None。
- **ValueError** - optimizer 是 DPOptimizer，且 noise_mech 非 None。
- **ValueError** - noise_mech 或 DPOptimizer 的 mech 方法是自适应的，而 clip_mech 不是 None。

CHAPTER

NINE

MINDARMOUR.PRIVACY.EVALUATION

本模块提供了一些评估给定模型隐私泄露风险的方法。

class mindarmour.privacy.evaluation.**MembershipInference**(*model*, *n_jobs=-1*)

成员推理是由 Shokri、Stronati、Song 和 Shmatikov 提出的一种用于推断用户隐私数据的灰盒攻击。它需要训练样本的 loss 或 logits 结果，隐私是指单个用户的一些敏感属性。

有关详细信息，请参见：教程。

参考文献：Reza Shokri, Marco Stronati, Congzheng Song, Vitaly Shmatikov. Membership Inference Attacks against Machine Learning Models. 2017.。

参数：

- **model** (Model) - 目标模型。
- **n_jobs** (int) - 并行运行的任务数量。-1 表示使用所有处理器，否则 n_jobs 的值必须为正整数。

异常：

- **TypeError** - 模型的类型不是 mindspore.Model 。
- **TypeError** - *n_jobs* 的类型不是 int。
- **ValueError** - *n_jobs* 的值既不是-1，也不是正整数。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindspore.nn import Cell
>>> from mindspore import Model
>>> from mindarmour.privacy.evaluation import MembershipInference
>>> def dataset_generator():
...     batch_size = 16
...     batches = 1
...     data =  np.random.randn(batches * batch_size,1,10).astype(np.float32)
...     label =  np.random.randint(0,10, batches * batch_size).astype(np.int32)
...     for i in range(batches):
...         yield data[i*batch_size:(i+1)*batch_size], label[i*batch_size:(i+1)*batch_size]
>>> class Net(Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         return self._squeeze(out)
>>> net = Net()
>>> loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
>>> opt = nn.Momentum(params=net.trainable_params(), learning_rate=0.1, momentum=0.9)
>>> model = Model(network=net, loss_fn=loss, optimizer=opt)
>>> inference_model = MembershipInference(model, 2)
>>> config = [{
...     "method": "KNN",
...     "params": {"n_neighbors": [3, 5, 7],}
...     }]
>>> ds_train = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> ds_test = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> inference_model.train(ds_train, ds_test, config)
>>> metrics = ["precision", "accuracy", "recall"]
>>> eval_train = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> eval_test = ds.GeneratorDataset(dataset_generator, ["image", "label"])
>>> result = inference_model.eval(eval_train. eval_test, metrics)
>>> print(result)
```

eval(*dataset_train*, *dataset_test*, *metrics*)

评估目标模型的隐私泄露程度。评估指标应由 metrics 规定。

参数：

- **dataset_train** (mindspore.dataset) - 目标模型的训练数据集。
- **dataset_test** (mindspore.dataset) - 目标模型的测试数据集。
- **metrics** (Union[list, tuple]) - 评估指标。指标的值必须在 [“precision”，“accuracy”，“recall”] 中。默认值：[“precision”]。

返回：

- **list** - 每个元素都包含攻击模型的评估指标。

train(*dataset_train*, *dataset_test*, *attack_config*)

根据配置，使用输入数据集训练攻击模型。

参数：

- **dataset_train** (mindspore.dataset) - 目标模型的训练数据集。
- **dataset_test** (mindspore.dataset) - 目标模型的测试集。
- **attack_config (Union[list, tuple]) - 攻击模型的参数设置。格式为**

```
attack_config =
    [{"method": "knn", "params": {"n_neighbors": [3, 5, 7]}},
     {"method": "lr", "params": {"C": np.logspace(-4, 2, 10)}}]
```

  - 支持的方法有 knn、lr、mlp 和 rf，每个方法的参数必须在可变参数的范围内。参数实现的提示可在下面找到：
    - KNN
    - LR
    - RF
    - MLP

异常：

- **KeyError** - *attack_config* 中的配置没有键 {“method”，“params”}。
- **NameError** - *attack_config* 中的方法（不区分大小写）不在 [“lr”，“knn”，“rf”，“mlp”] 中。

class mindarmour.privacy.evaluation.**ImageInversionAttack**(*network*, *input_shape*, *input_bound*, *loss_weights=(1*, *0.2*, *5)*)

一种通过还原图像的深层表达来重建图像的攻击方法。

参考文献：Aravindh Mahendran, Andrea Vedaldi. Understanding Deep Image Representations by Inverting Them. 2014.。

参数：

- **network** (Cell) - 网络，用于推断图像的深层特征。
- **input_shape** (tuple) - 单个网络输入的数据形状，应与给定网络一致。形状的格式应为 (channel, image_width, image_height)。
- **input_bound** (Union[tuple, list]) - 原始图像的像素范围，应该像 [minimum_pixel, maximum_pixel] 或 (minimum_pixel, maximum_pixel)。
- **loss_weights** (Union[list, tuple]) - InversionLoss 中三个子损失的权重，可以调整以获得更好的结果。默认值：(1, 0.2, 5)。

异常：

- **TypeError** - 网络类型不是 Cell。
- **ValueError** - *input_shape* 的值有非正整数。
- **ValueError** - *loss_weights* 的值有非正数。

样例：

```
>>> import mindspore.ops.operations as P
>>> from mindspore.nn import Cell
>>> from mindarmour.privacy.evaluation.inversion_attack import ImageInversionAttack
>>> class Net(Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._reduce = P.ReduceSum()
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._reduce(out, 2)
...         return self._squeeze(out)
>>> net = Net()
>>> original_images = np.random.random((2,1,10,10)).astype(np.float32)
```

(continues on next page)

(continued from previous page)

```
>>> target_features =  np.random.random((2,10)).astype(np.float32)
>>> inversion_attack = ImageInversionAttack(net,
...                                         input_shape=(1, 10, 10),
...                                         input_bound=(0, 1),
...                                         loss_weights=[1, 0.2, 5])
>>> inversion_images = inversion_attack.generate(target_features, iters=10)
>>> evaluate_result = inversion_attack.evaluate(original_images, inversion_images)
```

evaluate(*original_images*, *inversion_images*, *labels=None*, *new_network=None*)

通过三个指标评估还原图像的质量：原始图像和还原图像之间的平均 L2 距离和 SSIM 值，以及新模型对还原图像的推理结果在真实标签上的置信度平均值。

参数：

- **original_images** (numpy.ndarray) - 原始图像，其形状应为 (img_num, channels, img_width, img_height)。
- **inversion_images** (numpy.ndarray) - 还原图像，其形状应为 (img_num, channels, img_width, img_height)。
- **labels** (numpy.ndarray) - 原始图像的 ground truth 标签。默认值：None。
- **new_network** (Cell) - 其结构包含 self._network 中所有网络，但加载了不同的模型文件。默认值：None。

返回：

- **float** - l2 距离。
- **float** - 平均 ssim 值。
- **Union[float, None]** - 平均置信度。如果 *labels* 或 *new_network* 为 None，则该值为 None。

generate(*target_features*, *iters=100*)

根据 *target_features* 重建图像。

参数：

- **target_features** (numpy.ndarray) - 原始图像的深度表示。*target_features* 的第一个维度应该是 img_num。需要注意的是，如果 img_num 等于 1，则 *target_features* 的形状应该是 (1, dim2, dim3, ⋯)。
- **iters** (int) - 逆向攻击的迭代次数，应为正整数。默认值：100。

返回：

- **numpy.ndarray** - 重建图像，预计与原始图像相似。

异常：

- **TypeError** - target_features 的类型不是 numpy.ndarray。
- **ValueError** - *iters* 的值都不是正整数.

CHAPTER

TEN

MINDARMOUR.PRIVACY.SUP_PRIVACY

本模块提供抑制隐私功能，以保护用户隐私。

class mindarmour.privacy.sup_privacy.**SuppressMasker**(*model*, *suppress_ctrl*)

周期性检查抑制隐私功能状态和切换（启动/关闭）抑制操作。

详情请查看：应用抑制隐私机制保护用户隐私。

参数：

- **model** (SuppressModel) - SuppressModel 实例。
- **suppress_ctrl** (SuppressCtrl) - SuppressCtrl 实例。

样例：

```
>>> import mindspore.nn as nn
>>> import mindspore as ms
>>> from mindspore import set_context
>>> from mindspore.nn import Accuracy
>>> from mindarmour.privacy.sup_privacy import SuppressModel
>>> from mindarmour.privacy.sup_privacy import SuppressMasker
>>> from mindarmour.privacy.sup_privacy import SuppressPrivacyFactory
>>> from mindarmour.privacy.sup_privacy import MaskLayerDes
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = ops.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = ops.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         return self._squeeze(out)
>>> set_context(mode=ms.PYNATIVE_MODE, device_target="GPU")
>>> network = Net()
>>> masklayers = []
>>> masklayers.append(MaskLayerDes("_Dense.weight", 0, False, True, 10))
>>> suppress_ctrl_instance = SuppressPrivacyFactory().create(networks=network,
...                                                          mask_layers=masklayers,
...                                                          policy="local_train",
...                                                          end_epoch=10,
...                                                          batch_num=1,
...                                                          start_epoch=3,
...                                                          mask_times=10,
...                                                          lr=0.05,
...                                                          sparse_end=0.95,
...                                                          sparse_start=0.0)
>>> net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
>>> net_opt = nn.SGD(network.trainable_params(), 0.05)
>>> model_instance = SuppressModel(network=network,
...                                loss_fn=net_loss,
...                                optimizer=net_opt,
...                                metrics={"Accuracy": Accuracy()})
>>> model_instance.link_suppress_ctrl(suppress_ctrl_instance)
>>> masker_instance = SuppressMasker(model_instance, suppress_ctrl_instance)
```

step_end(*run_context*)

更新用于抑制模型实例的掩码矩阵张量。

参数：

- **run_context** (RunContext) - 包含模型的一些信息。

class mindarmour.privacy.sup_privacy.**SuppressModel**(*network*, *loss_fn*, *optimizer*, ***kwargs*)

抑制隐私训练器，重载自 mindspore.Model 。

有关详细信息，请查看：应用抑制隐私机制保护用户隐私。

参数：

- **network** (Cell) - 要训练的神经网络模型。
- **loss_fn** (Cell) - 优化器的损失函数。
- **optimizer** (Optimizer) - 优化器实例。
- **kwargs** - 创建抑制模型时使用的关键字参数。

link_suppress_ctrl(*suppress_pri_ctrl*)

SuppressCtrl 实例关联到 SuppressModel 实例。

参数：

- **suppress_pri_ctrl** (SuppressCtrl) - SuppressCtrl 实例。

class mindarmour.privacy.sup_privacy.**SuppressPrivacyFactory**

SuppressCtrl 机制的工厂类。

详情请查看：应用抑制隐私机制保护用户隐私。

create(*networks*, *mask_layers*, *policy='local_train'*, *end_epoch=10*, *batch_num=20*, *start_epoch=3*, *mask_times=1000*, *lr=0.05*, *sparse_end=0.90*, *sparse_start=0.0*)

参数：

- **networks** (Cell) - 要训练的神经网络模型。此网络参数应与 SuppressModel() 的’network’参数相同。
- **mask_layers** (list) - 需要抑制的训练网络层的描述。
- **policy** (str) - 抑制隐私训练的训练策略。默认值：”local_train”，表示本地训练。
- **end_epoch** (int) - 最后一次抑制操作对应的 epoch 序号，0<start_epoch<=end_epoch<=100。默认值：10。此 end_epoch 参数应与 mindspore.train.model.train() 的’epoch’参数相同。
- **batch_num** (int) - 一个 epoch 中批次的数量，应等于 num_samples/batch_size。默认值：20。
- **start_epoch** (int) - 第一个抑制操作对应的 epoch 序号，0<start_epoch<=end_epoch<=100。默认值：3。
- **mask_times** (int) - 抑制操作的数量。默认值：1000。
- **lr** (Union[float, int]) - 学习率，在训练期间应保持不变。0<lr<=0.50. 默认值：0.05。此 lr 参数应与 mindspore.nn.SGD() 的’learning_rate’参数相同。
- **sparse_end** (float) - 要到达的稀疏性，0.0<=sparse_start<sparse_end<1.0。默认值：0.90。
- **sparse_start** (Union[float, int]) - 抑制操作启动时对应的稀疏性，0.0<=sparse_start<sparse_end<1.0。默认值：0.0。

返回：

- **SuppressCtrl** - 抑制隐私机制的类。

样例：

```
>>> import mindspore.nn as nn
>>> import mindspore as ms
>>> from mindspore import set_context, ops
>>> from mindspore.nn import Accuracy
>>> from mindarmour.privacy.sup_privacy import SuppressPrivacyFactory
>>> from mindarmour.privacy.sup_privacy import MaskLayerDes
>>> from mindarmour.privacy.sup_privacy import SuppressModel
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = ops.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = ops.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         return self._squeeze(out)
>>> set_context(mode=ms.PYNATIVE_MODE, device_target="CPU")
>>> network = Net()
>>> masklayers = []
>>> masklayers.append(MaskLayerDes("_Dense.weight", 0, False, True, 10))
>>> suppress_ctrl_instance = SuppressPrivacyFactory().create(networks=network,
...                                                          mask_layers=masklayers,
...                                                          policy="local_train",
...                                                          end_epoch=10,
...                                                          batch_num=1,
...                                                          start_epoch=3,
...                                                          mask_times=10,
```

(continues on next page)

(continued from previous page)

```
...                                                    lr=0.05,
...                                                    sparse_end=0.95,
...                                                    sparse_start=0.0)
>>> net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction="mean")
>>> net_opt = nn.SGD(network.trainable_params(), 0.05)
>>> model_instance = SuppressModel(network=network,
...                                loss_fn=net_loss,
...                                optimizer=net_opt,
...                                metrics={"Accuracy": Accuracy()})
>>> model_instance.link_suppress_ctrl(suppress_ctrl_instance)
```

class mindarmour.privacy.sup_privacy.**SuppressCtrl**(*networks*, *mask_layers*, *end_epoch*, *batch_num*, *start_epoch*, *mask_times*, *lr*, *sparse_end*, *sparse_start*)

完成抑制隐私操作，包括计算抑制比例，找到应该抑制的参数，并永久抑制这些参数。

详情请查看：应用抑制隐私机制保护用户隐私。

参数：

- **networks** (Cell) - 要训练的神经网络模型。
- **mask_layers** (list) - 需要抑制的层的描述。
- **end_epoch** (int) - 最后一次抑制操作对应的 epoch 序号。
- **batch_num** (int) - 一个 epoch 中的 batch 数量。
- **start_epoch** (int) - 第一个抑制操作对应的 epoch 序号。
- **mask_times** (int) - 抑制操作的数量。
- **lr** (Union[float, int]) - 学习率。
- **sparse_end** (float) - 要到达的稀疏性。
- **sparse_start** (Union[float, int]) - 要启动的稀疏性。

calc_actual_sparse_for_conv(*networks*)

计算 con1 层和 con2 层的网络稀疏性。

参数：

- **networks** (Cell) - 要训练的神经网络模型。

calc_actual_sparse_for_fc1(*networks*)

计算全连接 1 层的实际稀疏

参数：

- **networks** (Cell) - 要训练的神经网络模型。

calc_actual_sparse_for_layer(*networks*, *layer_name*)

计算一个网络层的实际稀疏性

参数：

- **networks** (Cell) - 要训练的神经网络模型。
- **layer_name** (str) - 目标层的名称。

calc_theoretical_sparse_for_conv()

计算卷积层的掩码矩阵的实际稀疏性。

print_paras()

显示参数信息

reset_zeros()

将用于加法运算的掩码数组设置为 0。

update_mask(*networks*, *cur_step*, *target_sparse=0.0*)

对整个模型的用于加法运算和乘法运算的掩码数组进行更新。

参数：

- **networks** (Cell) - 训练网络。
- **cur_step** (int) - 整个训练过程的当前 epoch。
- **target_sparse** (float) - 要到达的稀疏性。默认值：0.0。

update_mask_layer(*weight_array_flat*, *sparse_weight_thd*, *sparse_stop_pos*, *weight_abs_max*, *layer_index*)

对单层的用于加法运算和乘法运算的掩码数组进行更新。

参数：

- **weight_array_flat** (numpy.ndarray) - 层参数权重数组。
- **sparse_weight_thd** (float) - 绝对值小于该阈值的权重会被抑制。

- **sparse_stop_pos** (int) - 要抑制的最大元素数。
- **weight_abs_max** (float) - 权重的最大绝对值。
- **layer_index** (int) - 目标层的索引。

update_mask_layer_approximity(*weight_array_flat*, *weight_array_flat_abs*, *actual_stop_pos*, *layer_index*)

对单层的用于加法运算和乘法运算的掩码数组进行更新。

禁用 clipping lower、clipping、adding noise 操作。

参数：

- **weight_array_flat** (numpy.ndarray) - 层参数权重数组。
- **weight_array_flat_abs** (numpy.ndarray) - 层参数权重的绝对值的数组。
- **actual_stop_pos** (int) - 应隐藏实际参数编号。
- **layer_index** (int) - 目标层的索引。

update_status(*cur_epoch*, *cur_step*, *cur_step_in_epoch*)

更新抑制操作状态。

参数：

- **cur_epoch** (int) - 整个训练过程的当前 epoch。
- **cur_step** (int) - 整个训练过程的当前步骤。
- **cur_step_in_epoch** (int) - 当前 epoch 的当前步骤。

class mindarmour.privacy.sup_privacy.**MaskLayerDes**(*layer_name*, *grad_idx*, *is_add_noise*, *is_lower_clip*, *min_num*, *upper_bound=1.20*)

对抑制目标层的描述。

参数：

- **layer_name (str) - 层名称，如下获取一个层的名称：**

```
for layer in networks.get_parameters(expand=True):
    if layer.name == "conv": ...
```

- **grad_idx** (int) - 掩码层在梯度元组中的索引。可参考 model.py 中 TrainOneStepCell 的构造函数，在 PYNATIVE_MODE 模式下打印某些层的索引值。
- **is_add_noise** (bool) - 如果为 True，则此层的权重可以添加噪声。如果为 False，则此层的权重不能添加噪声。如果参数 num 大于 100000，则 *is_add_noise* 无效。
- **is_lower_clip** (bool) - 如果为 True，则此层的权重将被剪裁到大于下限值。如果为 False，此层的权重不会被要求大于下限制。如果参数 num 大于 100000，则 is_lower_clip 无效。
- **min_num** (int) - 未抑制的剩余权重数。如果 min_num 小于（参数总数量 * *SupperssCtrl.sparse_end* ），则 min_num 无效。
- **upper_bound** (Union[float, int]) - 此层权重的最大 abs 值，默认值：1.20。如果参数 num 大于 100000，则 upper_bound 无效。

样例：

```
>>> from mindarmour.privacy.sup_privacy import MaskLayerDes
>>> masklayers = []
>>> masklayers.append(MaskLayerDes("conv1.weight", 0, False, True, 10))
```

CHAPTER

ELEVEN

MINDARMOUR.RELIABILITY

MindArmour 的可靠性方法。

class mindarmour.reliability.**FaultInjector**(*model*, *fi_type=None*, *fi_mode=None*, *fi_size=None*)

故障注入模块模拟深度神经网络的各种故障场景，并评估模型的性能和可靠性。

详情请查看 实现模型故障注入评估模型容错性。

参数：

- **model** (Model) - 需要评估模型。
- **fi_type** (list) - 故障注入的类型，包括’bitflips_random’（随机翻转）、’bitflips_designated’（翻转关键位）、’random’、’zeros’、’nan’、’inf’、’anti_activation’、’precision_loss’等。
- **fi_mode** (list) - 故障注入的模式。可选值：’single_layer’, ’all_layer’。
- **fi_size** (list) - 故障注入的次数，表示需要注入多少值。

样例：

```
>>> from mindspore import Model
>>> import mindspore.ops.operations as P
>>> from mindarmour.reliability.model_fault_injection.fault_injection import FaultInjector
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         return self._squeeze(out)
>>> def dataset_generator():
...     batch_size = 16
...     batches = 1
...     data =  np.random.randn(batches * batch_size,1,10).astype(np.float32)
...     label =  np.random.randint(0,10, batches * batch_size).astype(np.int32)
...     for i in range(batches):
...         yield data[i*batch_size:(i+1)*batch_size], label[i*batch_size:(i+1)*batch_size]
>>> net = Net()
>>> model = Model(net)
>>> ds_eval = ds.GeneratorDataset(dataset_generator, ['image', 'label'])
>>> fi_type = ['bitflips_random', 'bitflips_designated', 'random', 'zeros',
...            'nan', 'inf', 'anti_activation', 'precision_loss']
>>> fi_mode = ['single_layer', 'all_layer']
>>> fi_size = [1]
>>> fi = FaultInjector(model, ds_eval, fi_type, fi_mode, fi_size)
>>> fi.kick_off()
>>> fi.metrics()
```

kick_off(*ds_data*, *ds_label*, *iter_times=100*)

启动故障注入并返回最终结果。

参数：

- **ds_data** (np.ndarray) - 输入测试数据。评估基于这些数据。
- **ds_label** (np.ndarray) - 数据的标签，对应于数据。
- **iter_times** (int) - 评估数，这将决定批处理大小。

返回：

- **list** - 故障注入的结果。

metrics()

最终结果的指标。

返回：

- **list** - 结果总结。

class mindarmour.reliability.**ConceptDriftCheckTimeSeries**(*window_size=100*, *rolling_window=10*, *step=10*, *threshold_index=1.5*, *need_label=False*)

概念漂移检查时间序列（ConceptDriftCheckTimeSeries）用于样本序列分布变化检测。

有关详细信息，请查看 实现时序数据概念漂移检测应用。

参数：

- **window_size** (int) - 概念窗口的大小，不小于 10。如果给定输入数据，*window_size* 在 [10, 1/3*len(*data*)] 中。如果数据是周期性的，通常 *window_size* 等于 2-5 个周期。例如，对于月/周数据，30/7 天的数据量是一个周期。默认值：100。
- **rolling_window** (int) - 平滑窗口大小，在 [1, *window_size*] 中。默认值：10。
- **step** (int) - 滑动窗口的跳跃长度，在 [1, *window_size*] 中。默认值：10。
- **threshold_index** (float) - 阈值索引，$(-\infty, +\infty)$。默认值：1.5。
- **need_label** (bool) - False 或 True。如果 *need_label* =True，则需要概念漂移标签。默认值：False。

样例：

```
>>> from mindarmour import ConceptDriftCheckTimeSeries
>>> concept = ConceptDriftCheckTimeSeries(window_size=100, rolling_window=10,
...                                       step=10, threshold_index=1.5, need_label=False)
>>> data_example = 5*np.random.rand(1000)
>>> data_example[200: 800] = 20*np.random.rand(600)
>>> score, threshold, concept_drift_location = concept.concept_check(data_example)
```

concept_check(*data*)

在数据序列中查找概念漂移位置。

参数：

- **data** (numpy.ndarray) - 输入数据。数据的 shape 可以是 (n,1) 或 (n,m)。请注意，每列（m 列）是一个数据序列。

返回：

- **numpy.ndarray** - 样本序列的概念漂移分数。
- **float** - 判断概念漂移的阈值。
- **list** - 概念漂移的位置。

class mindarmour.reliability.**OodDetector**(*model*, *ds_train*)

分布外检测器的抽象类。

参数：

- **model** (Model) - 训练模型。
- **ds_train** (numpy.ndarray) - 训练数据集。

get_optimal_threshold(*label*, *ds_eval*)

获取最佳阈值。尝试找到一个最佳阈值来检测 OOD 样本。最佳阈值由标记的数据集 *ds_eval* 计算。

参数：

- **label** (numpy.ndarray) - 区分图像是否为分布内或分布外的标签。
- **ds_eval** (numpy.ndarray) - 帮助查找阈值的测试数据集。

返回：

- **float** - 最佳阈值。

ood_predict(*threshold*, *ds_test*)

分布外（out-of-distribution，OOD）检测。此函数的目的是检测被视为 *ds_test* 的图像是否为 OOD 样本。如果一张图像的预测分数大于 *threshold* ，则该图像为分布外。

参数：

- **threshold** (float) - 判断 ood 数据的阈值。可以根据经验设置值，也可以使用函数 get_optimal_threshold。
- **ds_test** (numpy.ndarray) - 测试数据集。

返回：

- **numpy.ndarray** - 检测结果。0 表示数据不是 ood，1 表示数据是 ood。

class mindarmour.reliability.**OodDetectorFeatureCluster**(*model*, *ds_train*, *n_cluster*, *layer*)

训练 OOD 检测器。提取训练数据特征，得到聚类中心。测试数据特征与聚类中心之间的距离确定图像是否为分布外（OOD）图像。

有关详细信息，请查看 实现图像数据概念漂移检测应用。

参数：

- **model** (Model) - 训练模型。

- **ds_train** (numpy.ndarray) - 训练数据集。
- **n_cluster** (int) - 聚类数量。取值属于 [2,100]。通常，n_cluster 等于训练数据集的类号。如果 OOD 检测器在测试数据集中性能较差，我们可以适当增加 n_cluster 的值。
- **layer** (str) - 特征层的名称。layer (str) 由’name[:Tensor]’表示，其中’name’由用户在训练模型时给出。请查看有关如何在’README.md’中命名模型层的更多详细信息。

样例：

```
>>> from mindspore import Model
>>> from mindspore.ops import TensorSummary
>>> import mindspore.ops.operations as P
>>> from mindarmour.reliability.concept_drift.concept_drift_check_images import OodDetectorFeatureCluster
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = P.Squeeze(1)
...         self._summary = TensorSummary()
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         self._summary('output', out)
...         return self._squeeze(out)
>>> net = Net()
>>> model = Model(net)
>>> batch_size = 16
>>> batches = 1
>>> ds_train = np.random.randn(batches * batch_size, 1, 10).astype(np.float32)
>>> ds_eval = np.random.randn(batches * batch_size, 1, 10).astype(np.float32)
>>> detector = OodDetectorFeatureCluster(model, ds_train, n_cluster=10, layer='output[:Tensor]')
>>> num = int(len(ds_eval) / 2)
>>> ood_label = np.concatenate((np.zeros(num), np.ones(num)), axis=0)
>>> optimal_threshold = detector.get_optimal_threshold(ood_label, ds_eval)
```

get_optimal_threshold(*label*, *ds_eval*)

获取最佳阈值。尝试找到一个最佳阈值来检测 OOD 样本。最佳阈值由标记的数据集 *ds_eval* 计算。

参数：

- **label** (numpy.ndarray) - 区分图像是否为分布内或分布外的标签。
- **ds_eval** (numpy.ndarray) - 帮助查找阈值的测试数据集。

返回：

- **float** - 最佳阈值。

ood_predict(*threshold*, *ds_test*)

分布外（out-of-distribution，OOD）检测。此函数的目的是检测 *ds_test* 中的图像是否为 OOD 样本。如果一张图像的预测分数大于 *threshold* ，则该图像为分布外。

参数：

- **threshold** (float) - 判断 ood 数据的阈值。可以根据经验设置值，也可以使用函数 get_optimal_threshold。
- **ds_test** (numpy.ndarray) - 测试数据集。

返回：

- **numpy.ndarray** - 检测结果。0 表示数据不是 ood，1 表示数据是 ood。

CHAPTER

TWELVE

MINDARMOUR.UTILS

MindArmour 的工具方法。

class mindarmour.utils.**LogUtil**

日志记录模块。

在长期运行的脚本中记录随时间推移的日志统计信息。

异常：

- **SyntaxError** - 创建此类异常。

add_handler(*handler*)

添加日志模块支持的其他处理程序。

参数：

- **handler** (logging.Handler) - 日志模块支持的其他处理程序。

异常：

- **ValueError** - 输入 handler 不是 logging.Handler 的实例。

debug(*tag*, *msg*, **args*)

记录’[tag] msg % args’，严重性为’DEBUG’。

参数：

- **tag** (str) - Logger 标记。
- **msg** (str) - Logger 消息。
- **args** (Any) - 辅助值。

error(*tag*, *msg*, **args*)

记录’[tag] msg % args’，严重性为’ERROR’。

参数：

- **tag** (str) - Logger 标记。
- **msg** (str) - Logger 消息。
- **args** (Any) - 辅助值。

get_instance()

获取类 *LogUtil* 的实例。

返回：

- **Object** - 类 *LogUtil* 的实例。

info(*tag*, *msg*, **args*)

记录’[tag] msg % args’，严重性为’INFO’。

参数：

- **tag** (str) - Logger 标记。
- **msg** (str) - Logger 消息。
- **args** (Any) - 辅助值。

set_level(*level*)

设置此 logger 的日志级别，级别必须是整数或字符串。支持的级别为‘NOTSET’(integer: 0)、’ERROR’(integer: 1-40)、’WARNING’(‘WARN’，integer: 1-30)、’INFO’(integer: 1-20) 以及’DEBUG’(integer: 1-10)

例如，如果 logger.set_level(‘WARNING’) 或 logger.set_level(21)，则在运行时将打印脚本中的 logger.warn() 和 logger.error()，而 logger.info() 或 logger.debug() 将不会打印。

参数：

- **level** (Union[int, str]) - logger 的级别。

warn(*tag*, *msg*, **args*)

记录’[tag] msg % args’，严重性为’WARNING’。

参数：

- **tag** (str) - Logger 标记。
- **msg** (str) - Logger 消息。
- **args** (Any) - 辅助值。

class mindarmour.utils.**GradWrapWithLoss**(*network*)

构造一个网络来计算输入空间中损失函数的梯度，并由 *weight* 加权。

参数：

- **network** (Cell) - 要包装的目标网络。

样例：

```
>>> from mindspore import Tensor
>>> from mindarmour.utils import GradWrapWithLoss
>>> from mindarmour.utils.util import WithLossCell
>>> import mindspore.ops.operations as P
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         return self._squeeze(out)
>>> data = Tensor(np.ones([2, 1, 10]).astype(np.float32)*0.01)
>>> labels = Tensor(np.ones([2, 10]).astype(np.float32))
>>> net = Net()
>>> loss_fn = nn.SoftmaxCrossEntropyWithLogits()
>>> loss_net = WithLossCell(net, loss_fn)
>>> grad_all = GradWrapWithLoss(loss_net)
>>> out_grad = grad_all(data, labels)
```

construct(*inputs*, *labels*)

使用标签和权重计算 *inputs* 的梯度。

参数：

- **inputs** (Tensor) - 网络的输入。
- **labels** (Tensor) - 输入的标签。

返回：

- **Tensor** - 梯度矩阵。

class mindarmour.utils.**GradWrap**(*network*)

构建一个网络，以计算输入空间中网络输出的梯度，并由 *weight* 加权，表示为雅可比矩阵。

参数：

- **network** (Cell) - 要包装的目标网络。

样例：

```
>>> from mindspore import Tensor
>>> from mindarmour.utils import GradWrap
>>> from mindarmour.utils.util import WithLossCell
>>> import mindspore.ops.operations as P
>>> class Net(nn.Cell):
...     def __init__(self):
...         super(Net, self).__init__()
...         self._softmax = P.Softmax()
...         self._Dense = nn.Dense(10,10)
...         self._squeeze = P.Squeeze(1)
...     def construct(self, inputs):
...         out = self._softmax(inputs)
...         out = self._Dense(out)
...         return self._squeeze(out)
>>> net = Net()
>>> data = Tensor(np.ones([2, 1, 10]).astype(np.float32)*0.01)
>>> labels = Tensor(np.ones([2, 10]).astype(np.float32))
>>> num_classes = 10
>>> sens = np.zeros((data.shape[0], num_classes)).astype(np.float32)
```

(continues on next page)

(continued from previous page)

```
>>> sens[:, 1] = 1.0
>>> wrap_net = GradWrap(net)
>>> wrap_net(data, Tensor(sens))
```

construct(**data*)

计算雅可比矩阵（jacobian matrix）。

参数：

- **data** (Tensor) - 数据由输入和权重组成。
  - inputs: 网络的输入。
  - weight: 每个梯度的权重，’weight’与’labels’的 shape 相同。

返回：

- **Tensor** - 雅可比矩阵。