## 1 Supported Attacks, Defences and Metrics

1.1 Setup ................................................................. 4
1.2 Examples .............................................................. 4
1.3 art.attacks .......................................................... 5
1.4 art.classifiers ....................................................... 21
1.5 art.data_generators ............................................... 38
1.6 art.defences ......................................................... 39
1.7 art.detection ........................................................ 52
1.8 art.detection.subsetscanning ................................... 57
1.9 art.poison_detection .............................................. 59
1.10 art.metrics ........................................................ 62
1.11 art.utils .......................................................... 63
1.12 art.utils_test ..................................................... 66
1.13 art.wrappers ...................................................... 67

## 2 Indices and tables

### Python Module Index

Index
This is a library dedicated to **adversarial machine learning**. Its purpose is to allow rapid crafting and analysis of attacks and defense methods for machine learning models. The Adversarial Robustness Toolbox provides an implementation for many state-of-the-art methods for attacking and defending classifiers. The code can be found on GitHub.

The library is still under development. Feedback, bug reports and extensions are highly appreciated.
SUPPORTED ATTACKS, DEFENCES AND METRICS

The Adversarial Robustness Toolbox contains implementations of the following evasion attacks:

- DeepFool (Moosavi-Dezfooli et al., 2015)
- Fast gradient method (Goodfellow et al., 2014)
- Basic iterative method (Kurakin et al., 2016)
- Projected gradient descent (Madry et al., 2017)
- Jacobian saliency map (Papernot et al., 2016)
- Universal perturbation (Moosavi-Dezfooli et al., 2016)
- Virtual adversarial method (Miyato et al., 2015)
- C&W L_2 and L_inf attacks (Carlini and Wagner, 2016)
- NewtonFool (Jang et al., 2017)
- Elastic net attack (Chen et al., 2017a)
- Spatial transformations attack (Engstrom et al., 2017)
- Query-efficient black-box attack (Ilyas et al., 2017)
- Zeroth-order optimization attack (Chen et al., 2017b)
- Decision-based attack (Brendel et al., 2018)
- Adversarial patch (Brown et al., 2017)
- HopSkipJump attack (Chen et al., 2017)

The following defense methods are also supported:

- Feature squeezing (Xu et al., 2017)
- Spatial smoothing (Xu et al., 2017)
- Label smoothing (Warde-Farley and Goodfellow, 2016)
- Adversarial training (Szegedy et al., 2013)
- Virtual adversarial training (Miyato et al., 2015)
- Gaussian data augmentation (Zantedeschi et al., 2017)
- Thermometer encoding (Buckman et al., 2018)
- Total variance minimization (Guo et al., 2018)
- JPEG compression (Dziugaite et al., 2016)
• PixelDefend (Song et al., 2017)

ART also implements detection methods of adversarial samples:
• Basic detector based on inputs
• Detector trained on the activations of a specific layer
• Detector based on Fast Generalized Subset Scan (Speakman et al., 2018)

The following detector of poisoning attacks is also supported:
• Detector based on activations analysis (Chen et al., 2018)

Robustness metrics:
• CLEVER (Weng et al., 2018)
• Empirical robustness (Moosavi-Dezfooli et al., 2015)
• Loss sensitivity (Arpit et al., 2017)

1.1 Setup

The Adversarial Robustness Toolbox is designed to run with Python 3 and Python 2.

1.1.1 Installation with pip

The library can be installed from the PyPi repository using pip:

```
pip install adversarial-robustness-toolbox
```

1.1.2 Manual installation

For the most recent version of the library, either download the source code or clone the repository in your directory of choice:

```
git clone https://github.com/IBM/adversarial-robustness-toolbox
```

To install ART, do the following in the project folder:

```
pip install .
```

The library comes with a basic set of unit tests. To check your install, you can run all the unit tests by calling in the library folder:

```
bash run_tests.sh
```

1.2 Examples

Some examples of how to use ART when writing your own code can be found in the examples folder on GitHub. See examples/README.md for more information about what each example does. To run an example, use the following command:
The notebooks folder contains Jupyter notebooks with detailed walkthroughs of some usage scenarios.

1.3 art.attacks

Module providing adversarial attacks under a common interface.

1.3.1 Adversarial Patch

class art.attacks.AdversarialPatch(classifier, target=0, rotation_max=22.5, scale_min=0.1, scale_max=1.0, learning_rate=5.0, max_iter=500, clip_patch=None, batch_size=16)


__init__(classifier, target=0, rotation_max=22.5, scale_min=0.1, scale_max=1.0, learning_rate=5.0, max_iter=500, clip_patch=None, batch_size=16)

Create an instance of the AdversarialPatch.

Parameters

• classifier (Classifier) – A trained model.

• target (int) – The target label for the created patch.

• rotation_max (float) – The maximum rotation applied to random patches. The value is expected to be in the range [0, 180].

• scale_min (float) – The minimum scaling applied to random patches. The value should be in the range [0, 1], but less than scale_max.

• scale_max (float) – The maximum scaling applied to random patches. The value should be in the range [0, 1], but larger than scale_min.

• learning_rate (float) – The learning rate of the optimization.

• max_iter (int) – The number of optimization steps.

• clip_patch ([ (float, float), (float, float), (float, float)]) – The minimum and maximum values for each channel

• batch_size (int) – The size of the training batch.

apply_patch (x, scale)

A function to apply the learned adversarial patch to images.

Parameters

• x (np.ndarray) – Instances to apply randomly transformed patch.

• scale (float) – Scale of the applied patch in relation to the classifier input shape.

Returns The patched instances.

Return type np.ndarray

generate (x, y=None, **kwargs)

Generate adversarial samples and return them in an array.

Parameters
• \textbf{x} (\textit{np.ndarray}) – An array with the original inputs. \textit{x} is expected to have spatial dimensions.

• \textbf{y} (\textit{np.ndarray}) – An array with the original labels to be predicted.

\textbf{Returns} An array holding the adversarial patch.

\textbf{Return type} \textit{np.ndarray}

\textbf{set_params (**kwargs)}

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

\textbf{Parameters}

• \textbf{target} (\textit{int}) – The target label.

• \textbf{rotation\_max} (\textit{float}) – The maximum rotation applied to random patches. The value is expected to be in the range [0, 180].

• \textbf{scale\_min} (\textit{float}) – The minimum scaling applied to random patches. The value should be in the range [0, 1], but less than \textbf{scale\_max}.

• \textbf{scale\_max} (\textit{float}) – The maximum scaling applied to random patches. The value should be in the range [0, 1], but greater than \textbf{scale\_min}.

• \textbf{learning\_rate} (\textit{float}) – The learning rate of the optimization.

• \textbf{max\_iter} (\textit{int}) – The number of optimization steps.

• \textbf{clip\_batch} – The minimum and maximum values for each channel

• \textbf{batch\_size} (\textit{int}) – The size of the training batch.

\subsection{1.3.2 Decision-Based Attack}

\textbf{class \textit{art.attacks.BoundaryAttack} (classifier, targeted=True, delta=0.01, epsilon=0.01, step\_adapt=0.667, max\_iter=5000, num\_trial=25, sample\_size=20, init\_size=100)}

Implementation of the boundary attack from Wieland Brendel et al. (2018). This is a powerful black-box attack that only requires final class prediction. Paper link: \url{https://arxiv.org/abs/1712.04248}

\textbf{\_\_init\_\_} (classifier, targeted=True, delta=0.01, epsilon=0.01, step\_adapt=0.667, max\_iter=5000, num\_trial=25, sample\_size=20, init\_size=100)

Create a boundary attack instance.

\textbf{Parameters}

• \textbf{classifier} (\textit{Classifier}) – A trained model.

• \textbf{targeted} (\textit{bool}) – Should the attack target one specific class.

• \textbf{delta} (\textit{float}) – Initial step size for the orthogonal step.

• \textbf{epsilon} (\textit{float}) – Initial step size for the step towards the target.

• \textbf{step\_adapt} (\textit{float}) – Factor by which the step sizes are multiplied or divided, must be in the range (0, 1).

• \textbf{max\_iter} (\textit{int}) – Maximum number of iterations.

• \textbf{num\_trial} (\textit{int}) – Maximum number of trials per iteration.

• \textbf{sample\_size} (\textit{int}) – Number of samples per trial.
• **init_size** (*int*) – Maximum number of trials for initial generation of adversarial examples.

**generate** (*x*, *y=None, **kwargs*)
Generate adversarial samples and return them in an array.

Parameters

• **x** (*np.ndarray*) – An array with the original inputs to be attacked.

• **y** (*np.ndarray*) – If self.targeted is true, then y represents the target labels.

• **x_adv_init** (*np.ndarray*) – Initial array to act as initial adversarial examples. Same shape as x.

Returns
An array holding the adversarial examples.

Return type
* np.ndarray

**set_params** (**kwargs**)
Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters

• **targeted** (*bool*) – Should the attack target one specific class.

• **delta** (*float*) – Initial step size for the orthogonal step.

• **epsilon** (*float*) – Initial step size for the step towards the target.

• **step_adapt** (*float*) – Factor by which the step sizes are multiplied or divided, must be in the range (0, 1).

• **max_iter** (*int*) – Maximum number of iterations.

• **num_trial** (*int*) – Maximum number of trials per iteration.

• **sample_size** (*int*) – Number of samples per trial.

• **init_size** (*int*) – Maximum number of trials for initial generation of adversarial examples.

1.3.3 Carlini and Wagner L_2 Attack

class **art.attacks.CarliniL2Method**(*classifier*, **confidence=0.0, targeted=False, learning_rate=0.01, binary_search_steps=10, max_iter=10, initial_const=0.01, max_halving=5, max_doubling=5, batch_size=1*)

The L_2 optimized attack of Carlini and Wagner (2016). This attack is among the most effective and should be used among the primary attacks to evaluate potential defences. A major difference wrt to the original implementation (https://github.com/carlini/nn_robust_attacks) is that we use line search in the optimization of the attack objective. Paper link: https://arxiv.org/pdf/1608.04644.pdf

__init__ ( * classifier, **confidence=0.0, targeted=False, learning_rate=0.01, binary_search_steps=10, max_iter=10, initial_const=0.01, max_halving=5, max_doubling=5, batch_size=1*)

Create a Carlini L_2 attack instance.

Parameters

• **classifier** (*Classifier*) – A trained model.

• **confidence** (*float*) – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.
• **targeted** *(bool)* – Should the attack target one specific class.

• **learning_rate** *(float)* – The initial learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.

• **binary_search_steps** *(int)* – number of times to adjust constant with binary search (positive value).

• **max_iter** *(int)* – The maximum number of iterations.

• **initial_const** *(float)* – The initial trade-off constant c to use to tune the relative importance of distance and confidence. If binary_search_steps is large, the initial constant is not important, as discussed in Carlini and Wagner (2016).

• **max_halving** *(int)* – Maximum number of halving steps in the line search optimization.

• **max_doubling** *(int)* – Maximum number of doubling steps in the line search optimization.

• **batch_size** *(int)* – Internal size of batches on which adversarial samples are generated.

**generate** *(x, y=None, **kwargs)*

Generate adversarial samples and return them in an array.

**Parameters**

• **x** *(np.ndarray)* – An array with the original inputs to be attacked.

• **y** *(np.ndarray)* – If self.targeted is true, then y_val represents the target labels. Otherwise, the targets are the original class labels.

**Returns** An array holding the adversarial examples.

**Return type** *np.ndarray*

**set_params** *(**kwargs)*

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

**Parameters**

• **confidence** *(float)* – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.

• **targeted** *(bool)* – Should the attack target one specific class

• **learning_rate** *(float)* – The learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.

• **binary_search_steps** *(int)* – number of times to adjust constant with binary search (positive value)

• **max_iter** *(int)* – The maximum number of iterations.

• **initial_const** *(float)* – (optional float, positive) The initial trade-off constant c to use to tune the relative importance of distance and confidence. If binary_search_steps is large, the initial constant is not important. The default value 1e-4 is suggested in Carlini and Wagner (2016).

• **max_halving** *(int)* – Maximum number of halving steps in the line search optimization.

• **max_doubling** *(int)* – Maximum number of doubling steps in the line search optimization.

• **batch_size** *(int)* – Internal size of batches on which adversarial samples are generated.
1.3.4 Carlini and Wagner L\_inf Attack

class art.attacks.CarliniLInfMethod(classifier, confidence=0.0, targetted=False, learning\_rate=0.01, max\_iter=10, max\_halving=5, max\_doubling=5, eps=0.3, batch\_size=128)

This is a modified version of the L\_2 optimized attack of Carlini and Wagner (2016). It controls the L\_Inf norm, i.e. the maximum perturbation applied to each pixel.

__init__(classifier, confidence=0.0, targetted=False, learning\_rate=0.01, max\_iter=10, max\_halving=5, max\_doubling=5, eps=0.3, batch\_size=128)

Create a Carlini L\_Inf attack instance.

Parameters

- **classifier (Classifier)** – A trained model.
- **confidence (float)** – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.
- **targeted (bool)** – Should the attack target one specific class.
- **learning\_rate (float)** – The initial learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.
- **max\_iter (int)** – The maximum number of iterations.
- **max\_halving (int)** – Maximum number of halving steps in the line search optimization.
- **max\_doubling (int)** – Maximum number of doubling steps in the line search optimization.
- **eps (float)** – An upper bound for the L\_0 norm of the adversarial perturbation.
- **batch\_size (int)** – Internal size of batches on which adversarial samples are generated.
- **expectation (ExpectationOverTransformations)** – An expectation over transformations to be applied when computing classifier gradients and predictions.

generate (x, y=None, **kwargs)

Generate adversarial samples and return them in an array.

Parameters

- **x (np.ndarray)** – An array with the original inputs to be attacked.
- **y (np.ndarray)** – If self.targetted is true, then y\_val represents the target labels. Otherwise, the targets are the original class labels.

Returns An array holding the adversarial examples.

Return type np.ndarray

set\_params (**kwargs)

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters

- **confidence (float)** – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.
- **targeted (bool)** – Should the attack target one specific class
- **learning\_rate (float)** – The learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.
• \texttt{max_iter} (int) – The maximum number of iterations.
• \texttt{max_halving} (int) – Maximum number of halving steps in the line search optimization.
• \texttt{max_doubling} (int) – Maximum number of doubling steps in the line search optimization.
• \texttt{eps} (float) – An upper bound for the $L_0$ norm of the adversarial perturbation.
• \texttt{batch_size} (int) – Internal size of batches on which adversarial samples are generated.

1.3.5 DeepFool

class \texttt{art.attacks.DeepFool}(\texttt{classifier}, \texttt{max_iter}=100, \texttt{epsilon}=1e-06, \texttt{nb_grads}=10, \texttt{batch_size}=1)


__init__ (\texttt{classifier}, \texttt{max_iter}=100, \texttt{epsilon}=1e-06, \texttt{nb_grads}=10, \texttt{batch_size}=1)

Create a DeepFool attack instance.

Parameters

• \texttt{classifier}(\texttt{Classifier}) – A trained model.
• \texttt{max_iter} (int) – The maximum number of iterations.
• \texttt{epsilon} (float) – Overshoot parameter.
• \texttt{nb_grads} (int) – The number of class gradients (top \texttt{nb_grads} w.r.t. prediction) to compute. This way only the most likely classes are considered, speeding up the computation.
• \texttt{batch_size} (int) – Batch size

generate (\texttt{x}, \texttt{y}=\texttt{None}, **\texttt{kwargs})

Generate adversarial samples and return them in an array.

Parameters

• \texttt{x} (\texttt{np.ndarray}) – An array with the original inputs to be attacked.
• \texttt{y} (\texttt{np.ndarray}) – An array with the original labels to be predicted.

Returns An array holding the adversarial examples.

Return type \texttt{np.ndarray}

set_params (**\texttt{kwargs})

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters

• \texttt{max_iter} (int) – The maximum number of iterations.
• \texttt{epsilon} (float) – Overshoot parameter.
• \texttt{nb_grads} (int) – The number of class gradients (top \texttt{nb_grads} w.r.t. prediction) to compute. This way only the most likely classes are considered, speeding up the computation.
• \texttt{batch_size} (int) – Internal size of batches on which adversarial samples are generated.
1.3.6 Elastic Net Attack (EAD)

class art.attacks.ElasticNet(classifier, confidence=0.0, targeted=False, learning_rate=0.01, binary_search_steps=9, max_iter=100, beta=0.001, initial_const=0.001, batch_size=1, decision_rule='EN')


__init__(classifier, confidence=0.0, targeted=False, learning_rate=0.01, binary_search_steps=9, max_iter=100, beta=0.001, initial_const=0.001, batch_size=1, decision_rule='EN')

Create an ElasticNet attack instance.

Parameters

• classifier (Classifier) – A trained model.

• confidence (float) – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.

• targeted (bool) – Should the attack target one specific class.

• learning_rate (float) – The initial learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.

• binary_search_steps (int) – Number of times to adjust constant with binary search (positive value).

• max_iter (int) – The maximum number of iterations.

• beta (float) – Hyperparameter trading off L2 minimization for L1 minimization.

• initial_const (float) – The initial trade-off constant c to use to tune the relative importance of distance and confidence. If binary_search_steps is large, the initial constant is not important, as discussed in Carlini and Wagner (2016).

• batch_size (int) – Internal size of batches on which adversarial samples are generated.


generate (x, y=None, **kwargs)

Generate adversarial samples and return them in an array.

Parameters

• x (np.ndarray) – An array with the original inputs to be attacked.

• y (np.ndarray) – If self.targeted is true, then y represents the target labels. Otherwise, the targets are the original class labels.

Returns

An array holding the adversarial examples.

Return type

np.ndarray

set_params (**kwargs)

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters

• confidence (float) – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.

• targeted (bool) – Should the attack target one specific class.
• **learning_rate** (float) – The initial learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.

• **binary_search_steps** (int) – Number of times to adjust constant with binary search (positive value).

• **max_iter** (int) – The maximum number of iterations.

• **beta** (float) – Hyperparameter trading off L2 minimization for L1 minimization.

• **initial_const** (float) – The initial trade-off constant $c$ to use to tune the relative importance of distance and confidence. If **binary_search_steps** is large, the initial constant is not important, as discussed in Carlini and Wagner (2016).

• **batch_size** (int) – Internal size of batches on which adversarial samples are generated.


### 1.3.7 Fast Gradient Method (FGM)

class **art.attacks.FastGradientMethod**(classifier, norm=inf, eps=0.3, eps_step=0.1, targeted=False, num_random_init=0, batch_size=1, minimal=False)

This attack was originally implemented by Goodfellow et al. (2015) with the infinity norm (and is known as the “Fast Gradient Sign Method”). This implementation extends the attack to other norms, and is therefore called the Fast Gradient Method. Paper link: https://arxiv.org/abs/1412.6572

__init__ (classifier, norm=inf, eps=0.3, eps_step=0.1, targeted=False, num_random_init=0, batch_size=1, minimal=False)

Create a **FastGradientMethod** instance.

**Parameters**

• **classifier** (**Classifier**) – A trained model.

• **norm** (int) – Order of the norm. Possible values: np.inf, 1 or 2.

• **eps** (float) – Attack step size (input variation)

• **eps_step** (float) – Step size of input variation for minimal perturbation computation

• **targeted** (bool) – Should the attack target one specific class

• **num_random_init** (int) – Number of random initialisations within the epsilon ball. For random_init=0 starting at the original input.

• **batch_size** (int) – Batch size

• **minimal** (bool) – Flag to compute the minimal perturbation.

**generate** (x, y=None, **kwargs)

Generate adversarial samples and return them in an array.

**Parameters**

• **x** (np.ndarray) – An array with the original inputs.

• **y** (np.ndarray) – The labels for the data $x$. Only provide this parameter if you’d like to use true labels when crafting adversarial samples. Otherwise, model predictions are used as labels to avoid the “label leaking” effect (explained in this paper: https://arxiv.org/abs/1611.01236). Default is $None$. Labels should be one-hot-encoded.

**Returns** An array holding the adversarial examples.
Return type `np.ndarray`

**set_params(**`**kwargs`)**

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

- **param norm**: Order of the norm. Possible values: `np.inf`, `1` or `2`
- **type norm**: `int` or `float`
- **param eps**: Attack step size (input variation)
- **type eps**: `float`
- **param eps_step**: Step size of input variation for minimal perturbation computation
- **type eps_step**: `float`
- **param targeted**: Should the attack target one specific class
- **type targeted**: `bool`
- **param num_random_init**: Number of random initialisations within the epsilon ball. For random_init=0 starting at the original input.

Parameters

- **batch_size** (`int`) – Batch size
- **minimal** (`bool`) – Flag to compute the minimal perturbation.

### 1.3.8 Basic Iterative Method (BIM)

**class** `art.attacks.BasicIterativeMethod`(`classifier`, `eps=0.3`, `eps_step=0.1`, `max_iter=100`, `targeted=False`, `batch_size=1`)

The Basic Iterative Method is the iterative version of FGM and FGSM. Paper link: [https://arxiv.org/abs/1607.02533](https://arxiv.org/abs/1607.02533)

**__init__**(`classifier`, `eps=0.3`, `eps_step=0.1`, `max_iter=100`, `targeted=False`, `batch_size=1`)

Create a `ProjectedGradientDescent` instance.

Parameters

- **classifier** (`Classifier`) – A trained model.
- **eps** (`float`) – Maximum perturbation that the attacker can introduce.
- **eps_step** (`float`) – Attack step size (input variation) at each iteration.
- **max_iter** (`int`) – The maximum number of iterations.
- **targeted** (`bool`) – Should the attack target one specific class
- **batch_size** (`int`) – Batch size

### 1.3.9 Projected Gradient Descent (PGD)

**class** `art.attacks.ProjectedGradientDescent`(`classifier`, `norm=np.inf`, `eps=0.3`, `eps_step=0.1`, `max_iter=100`, `targeted=False`, `num_random_init=0`, `batch_size=1`)

The Projected Gradient Descent attack is an iterative method in which, after each iteration, the perturbation is projected on an lp-ball of specified radius (in addition to clipping the values of the adversarial sample so that it lies in the permitted data range). This is the attack proposed by Madry et al. for adversarial training. Paper link: [https://arxiv.org/abs/1706.06083](https://arxiv.org/abs/1706.06083)

**__init__**(`classifier`, `norm=np.inf`, `eps=0.3`, `eps_step=0.1`, `max_iter=100`, `targeted=False`, `num_random_init=0`, `batch_size=1`)

Create a `ProjectedGradientDescent` instance.

Parameters

- **classifier** (`Classifier`) – A trained model.
- **norm** (`int`) – Order of the norm. Possible values: `np.inf`, `1` or `2`. 

1.3. `art.attacks`
• **eps** (float) – Maximum perturbation that the attacker can introduce.

• **eps_step** (float) – Attack step size (input variation) at each iteration.

• **max_iter** (int) – The maximum number of iterations.

• **targeted** (bool) – Should the attack target one specific class

• **num_random_init** (int) – Number of random initialisations within the epsilon ball.
  For num_random_init=0 starting at the original input.

• **batch_size** (int) – Batch size

`generate` *(x, y=None, **kwargs)*
Generate adversarial samples and return them in an array.

**Parameters**

- **x** *(np.ndarray)* – An array with the original inputs.

- **y** *(np.ndarray)* – The labels for the data x. Only provide this parameter if you’d like to use true labels when crafting adversarial samples. Otherwise, model predictions are used as labels to avoid the “label leaking” effect (explained in this paper: https://arxiv.org/abs/1611.01236). Default is `None`. Labels should be one-hot-encoded.

**Returns**
An array holding the adversarial examples.

**Return type** `np.ndarray`

`set_params` (**kwargs)**
Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

**Parameters**

- **norm** (int) – Order of the norm. Possible values: np.inf, 1 or 2.

- **eps** (float) – Maximum perturbation that the attacker can introduce.

- **eps_step** (float) – Attack step size (input variation) at each iteration.

- **num_random_init** (int) – Number of random initialisations within the epsilon ball.
  For num_random_init=0 starting at the original input.

- **batch_size** (int) – Batch size

### 1.3.10 Jacobian Saliency Map Attack (JSMA)

**class art.attacks.SaliencyMapMethod** *(classifier, theta=0.1, gamma=1.0, batch_size=1)*


**__init__** *(classifier, theta=0.1, gamma=1.0, batch_size=1)*
Create a SaliencyMapMethod instance.

**Parameters**

- **classifier** *(Classifier)* – A trained model.

- **theta** (float) – Perturbation introduced to each modified feature per step (can be positive or negative).

- **gamma** (float) – Maximum percentage of perturbed features (between 0 and 1).

- **batch_size** (int) – Batch size
generate \( (x, y=\text{None}, **\text{kwargs}) \)
Generate adversarial samples and return them in an array.

Parameters
- \( x \ (\text{np.ndarray}) \) – An array with the original inputs to be attacked.
- \( y \ (\text{np.ndarray}) \) – Target values if the attack is targeted

Returns
An array holding the adversarial examples.

Return type
\( \text{np.ndarray} \)

set_params \( (**\text{kwargs}) \)
Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters
- \( \theta \ (\text{float}) \) – Perturbation introduced to each modified feature per step (can be positive or negative)
- \( \gamma \ (\text{float}) \) – Maximum percentage of perturbed features (between 0 and 1)
- \( \text{batch size} \ (\text{int}) \) – Internal size of batches on which adversarial samples are generated.

1.3.11 NewtonFool

class art.attacks.NewtonFool \( (\text{classifier, max_iter=100, eta=0.01, batch size=1}) \)

__init__ \( (\text{classifier, max_iter=100, eta=0.01, batch size=1}) \)
Create a NewtonFool attack instance.

Parameters
- \( \text{classifier} \ (\text{Classifier}) \) – A trained model.
- \( \text{max_iter} \ (\text{int}) \) – The maximum number of iterations.
- \( \eta \ (\text{float}) \) – The eta coefficient.
- \( \text{batch size} \ (\text{int}) \) – Batch size

generate \( (x, y=\text{None}, **\text{kwargs}) \)
Generate adversarial samples and return them in a Numpy array.

Parameters
- \( x \ (\text{np.ndarray}) \) – An array with the original inputs to be attacked.
- \( y \ (\text{np.ndarray}) \) – An array with the original labels to be predicted.

Returns
An array holding the adversarial examples.

Return type
\( \text{np.ndarray} \)

set_params \( (**\text{kwargs}) \)
Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters
- \( \text{max_iter} \ (\text{int}) \) – The maximum number of iterations.
- \( \eta \ (\text{float}) \) – The eta coefficient.
- \( \text{batch size} \ (\text{int}) \) – Internal size of batches on which adversarial samples are generated.
1.3.12 Spatial Transformations Attack

```python
class art.attacks.SpatialTransformation(classifier, max_translation=0.0, num_translations=1, max_rotation=0.0, num_rotations=1)
```

Implementation of the spatial transformation attack using translation and rotation of inputs. The attack conducts black-box queries to the target model in a grid search over possible translations and rotations to find optimal attack parameters. Paper link: https://arxiv.org/abs/1712.02779

```python
__init__(classifier, max_translation=0.0, num_translations=1, max_rotation=0.0, num_rotations=1)
```

Parameters

- **classifier** (*Classifier*) – A trained model.
- **max_translation** (*float*) – The maximum translation in any direction as percentage of image size. The value is expected to be in the range $[0, 100]$. 
- **num_translations** (*int*) – The number of translations to search on grid spacing per direction.
- **max_rotation** (*float*) – The maximum rotation in either direction in degrees. The value is expected to be in the range $[0, 180]$. 
- **num_rotations** (*int*) – The number of rotations to search on grid spacing.

```python
generate(x, y=None, **kwargs)
```

Generate adversarial samples and return them in an array.

Parameters

- **x** (*np.ndarray*) – An array with the original inputs.
- **y** (*np.ndarray*) – An array with the original labels to be predicted.

Returns

An array holding the adversarial examples.

Return type

* np.ndarray

```python
set_params(**kwargs)
```

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters

- **max_translation** (*float*) – The maximum translation in any direction as percentage of image size. The value is expected to be in the range $[0, 100]$. 
- **num_translations** (*int*) – The number of translations to search on grid spacing per direction.
- **max_rotation** (*float*) – The maximum rotation in either direction in degrees. The value is expected to be in the range $[0, 180]$. 
- **num_rotations** (*int*) – The number of rotations to search on grid spacing.

1.3.13 Universal Perturbation Attack

```python
class art.attacks.UniversalPerturbation(classifier, attacker='deepfool', attacker_params=None, delta=0.2, max_iter=20, eps=10.0, norm=inf)
```

Implementation of the attack from Moosavi-Dezfooli et al. (2016). Computes a fixed perturbation to be applied
to all future inputs. To this end, it can use any adversarial attack method. Paper link: https://arxiv.org/abs/1610.08401

```python
__init__(classifier, attacker='deepfool', attacker_params=None, delta=0.2, max_iter=20, eps=10.0, norm=inf)
```

Parameters

- **classifier** (*Classifier*) – A trained model.
- **attacker_params** (*dict*) – Parameters specific to the adversarial attack.
- **delta** (*float*) – desired accuracy
- **max_iter** (*int*) – The maximum number of iterations for computing universal perturbation.
- **eps** (*float*) – Attack step size (input variation)
- **norm** (*int*) – Order of the norm. Possible values: np.inf, 2 (default is np.inf)

```python
generate(x, y=None, **kwargs)
```

Generate adversarial samples and return them in an array.

Parameters

- **x** (*np.ndarray*) – An array with the original inputs.
- **y** (*np.ndarray*) – An array with the original labels to be predicted.

Returns

An array holding the adversarial examples.

Return type

*np.ndarray*

```python
set_params(**kwargs)
```

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters

- **attacker_params** (*dict*) – Parameters specific to the adversarial attack.
- **delta** (*float*) – desired accuracy
- **max_iter** (*int*) – The maximum number of iterations for computing universal perturbation.
- **eps** (*float*) – Attack step size (input variation)
- **norm** (*int*) – Order of the norm. Possible values: np.inf, 2 (default is np.inf)

### 1.3.14 Virtual Adversarial Method

```python
class art.attacks.VirtualAdversarialMethod(classifier, max_iter=10, finite_diff=1e-06, eps=0.1, batch_size=1)
```

This attack was originally proposed by Miyato et al. (2016) and was used for virtual adversarial training. Paper link: https://arxiv.org/abs/1507.00677
__init__ (classifier, max_iter=10, finite_diff=1e-06, eps=0.1, batch_size=1)
Create a VirtualAdversarialMethod instance.

Parameters
• classifier (Classifier) – A trained model.
• eps (float) – Attack step (max input variation).
• finite_diff (float) – The finite difference parameter.
• max_iter (int) – The maximum number of iterations.
• batch_size (int) – Batch size.

generate (x, y=None, **kwargs)
Generate adversarial samples and return them in an array.

Parameters
• x (np.ndarray) – An array with the original inputs to be attacked.
• y (np.ndarray) – An array with the original labels to be predicted.

Returns An array holding the adversarial examples.
Return type np.ndarray

set_params (**kwargs)
Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

Parameters
• eps (float) – Attack step (max input variation).
• finite_diff (float) – The finite difference parameter.
• max_iter (int) – The maximum number of iterations.
• batch_size (int) – Internal size of batches on which adversarial samples are generated.

1.3.15 Zeroth-Order Optimization Attack (ZOO)

class art.attacks.ZooAttack (classifier, confidence=0.0, targeted=False, learning_rate=0.01, max_iter=10, binary_search_steps=1, initial_const=0.001, abort_early=True, use_resize=True, use_importance=True, nb_parallel=128, batch_size=1)

__init__ (classifier, confidence=0.0, targeted=False, learning_rate=0.01, max_iter=10, binary_search_steps=1, initial_const=0.001, abort_early=True, use_resize=True, use_importance=True, nb_parallel=128, batch_size=1)
Create a ZOO attack instance.

Parameters
• classifier (Classifier) – A trained model.
• confidence (float) – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.
• targeted (bool) – Should the attack target one specific class.
• **learning_rate** (*float*) – The initial learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.

• **max_iter** (*int*) – The maximum number of iterations.

• **binary_search_steps** (*int*) – Number of times to adjust constant with binary search (positive value).

• **initial_const** (*float*) – The initial trade-off constant $c$ to use to tune the relative importance of distance and confidence. If `binary_search_steps` is large, the initial constant is not important, as discussed in Carlini and Wagner (2016).

• **abort_early** (*bool*) – True if gradient descent should be abandoned when it gets stuck.

• **use_resize** (*bool*) – True if to use the resizing strategy from the paper: first, compute attack on inputs resized to 32x32, then increase size if needed to 64x64, followed by 128x128.

• **use_importance** (*bool*) – True if to use importance sampling when choosing coordinates to update.

• **nb_parallel** (*int*) – Number of coordinate updates to run in parallel. A higher value for `nb_parallel` should be preferred over a large batch size.

• **batch_size** (*int*) – Internal size of batches on which adversarial samples are generated. Small batch sizes are encouraged for ZOO, as the algorithm already runs `nb_parallel` coordinate updates in parallel for each sample. The batch size is a multiplier of `nb_parallel` in terms of memory consumption.

**generate**(*x*, *y=None, **kwargs*)

Generate adversarial samples and return them in an array.

**Parameters**

• **x** (*np.ndarray*) – An array with the original inputs to be attacked.

• **y** (*np.ndarray*) – If `self.targeted` is true, then $y$ represents the target labels. Otherwise, the targets are the original class labels.

**Returns** An array holding the adversarial examples.

**Return type** *np.ndarray*

**set_params**(**kwargs**)

Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.

**Parameters**

• **confidence** (*float*) – Confidence of adversarial examples: a higher value produces examples that are farther away, from the original input, but classified with higher confidence as the target class.

• **targeted** (*bool*) – Should the attack target one specific class.

• **learning_rate** (*float*) – The initial learning rate for the attack algorithm. Smaller values produce better results but are slower to converge.

• **max_iter** (*int*) – The maximum number of iterations.

• **binary_search_steps** (*int*) – Number of times to adjust constant with binary search (positive value).

• **initial_const** (*float*) – The initial trade-off constant $c$ to use to tune the relative importance of distance and confidence. If `binary_search_steps` is large, the initial constant is not important, as discussed in Carlini and Wagner (2016).
• **abort_early**(bool) – True if gradient descent should be abandoned when it gets stuck.

• **use_resize**(bool) – True if to use the resizing strategy from the paper: first, compute attack on inputs resized to 32x32, then increase size if needed to 64x64, followed by 128x128.

• **use_importance**(bool) – True if to use importance sampling when choosing coordinates to update.

• **nb_parallel**(int) – Number of coordinate updates to run in parallel. A higher value for `nb_parallel` should be preferred over a large batch size.

• **batch_size**(int) – Internal size of batches on which adversarial samples are generated. Small batch sizes are encouraged for ZOO, as the algorithm already runs `nb_parallel` coordinate updates in parallel for each sample. The batch size is a multiplier of `nb_parallel` in terms of memory consumption.

### 1.3.16 HopSkipJump Attack

class art.attacks.HopSkipJump(classifier, targeted=False, norm=2, max_iter=50, max_eval=10000, init_eval=100, init_size=100)

Implementation of the HopSkipJump attack from Jianbo et al. (2019). This is a powerful black-box attack that only requires final class prediction, and is an advanced version of the boundary attack. Paper link: https://arxiv.org/abs/1904.02144

__init__(classifier, targeted=False, norm=2, max_iter=50, max_eval=10000, init_eval=100, init_size=100)
Create a HopSkipJump attack instance.

Parameters

- **classifier**(Classifier) – A trained model.
- **targeted**(bool) – Should the attack target one specific class.
- **norm**(int) – Order of the norm. Possible values: np.inf or 2.
- **max_iter**(int) – Maximum number of iterations.
- **max_eval**(int) – Maximum number of evaluations for estimating gradient.
- **init_eval**(int) – Initial number of evaluations for estimating gradient.
- **init_size**(int) – Maximum number of trials for initial generation of adversarial examples.

generate(x, y=None, **kwargs)
Generate adversarial samples and return them in an array.

Parameters

- **x**(np.ndarray) – An array with the original inputs to be attacked.
- **y**(np.ndarray) – If `self.targeted` is true, then `y` represents the target labels.
- **x_adv_init**(np.ndarray) – Initial array to act as initial adversarial examples. Same shape as `x`.

Returns An array holding the adversarial examples.

Return type np.ndarray

set_params(**kwargs)
Take in a dictionary of parameters and applies attack-specific checks before saving them as attributes.
Parameters

- **targeted** *(bool)* – Should the attack target one specific class.
- **norm** *(int)* – Order of the norm. Possible values: np.inf or 2.
- **max_iter** *(int)* – Maximum number of iterations.
- **max_eval** *(int)* – Maximum number of evaluations for estimating gradient.
- **init_eval** *(int)* – Initial number of evaluations for estimating gradient.
- **init_size** *(int)* – Maximum number of trials for initial generation of adversarial examples.

### 1.3.17 Base Class

class art.attacks.Attack(classifier)

Abstract base class for all attack classes.

**generate** *(x, y=None, **kwargs)*

Generate adversarial examples and return them as an array. This method should be overridden by all concrete attack implementations.

**Parameters**

- **x** *(np.ndarray)* – An array with the original inputs to be attacked.
- **y** *(np.ndarray)* – Correct labels or target labels for x, depending if the attack is targeted or not. This parameter is only used by some of the attacks.

**Returns** An array holding the adversarial examples.

**Return type** *np.ndarray*

set_params (**kwargs)**

Take in a dictionary of parameters and apply attack-specific checks before saving them as attributes.

**Parameters** **kwargs** *(dict)* – a dictionary of attack-specific parameters

**Returns** *True* when parsing was successful

### 1.4 art.classifiers

Classifier API for applying all attacks. Use the Classifier wrapper to be able to apply an attack to a preexisting model.

#### 1.4.1 Keras Wrapper

class art.classifiers.KerasClassifier(model, use_logits=False, channel_index=3, clip_values=None, defences=None, preprocessing=(0, 1), input_layer=0, output_layer=0, custom_activation=False)

Wrapper class for importing Keras models. The supported backends for Keras are TensorFlow and Theano.

**__getstate__** ()

Use to ensure KerasClassifier can be pickled.

**Returns** State dictionary with instance parameters.
Return type  dict

`__init__`(model, use_logits=False, channel_index=3, clip_values=None, defences=None, preprocessing=(0, 1), input_layer=0, output_layer=0, custom_activation=False)

Create a `Classifier` instance from a Keras model. Assumes the `model` passed as argument is compiled.

Parameters

- **model** *(keras.models.Model)* – Keras model
- **use_logits** *(bool)* – True if the output of the model are the logits.
- **channel_index** *(int)* – Index of the axis in data containing the color channels or features.
- **clip_values** *(tuple)* – Tuple of the form `(min, max)` of floats or `np.ndarray` representing the minimum and maximum values allowed for features. If floats are provided, these will be used as the range of all features. If arrays are provided, each value will be considered the bound for a feature, thus the shape of clip values needs to match the total number of features.
- **defences** *(Preprocessor or list(Preprocessor) instances)* – Defences to be activated with the classifier.
- **preprocessing** *(tuple)* – Tuple of the form `(subtractor, divider)` of floats or `np.ndarray` of values to be used for data preprocessing. The first value will be substracted from the input. The input will then be divided by the second one.
- **input_layer** *(int)* – Which layer to consider as the Input when the model has multiple input layers.
- **output_layer** *(int)* – Which layer to consider as the Output when the model has multiple output layers.
- **custom_activation** *(bool)* – True if the model uses the last activation other than softmax and requires to use the output probability rather than the logits by attacks.

`__repr__`()  
Return `repr(self)`.

`__setstate__`(state)

Use to ensure `KerasClassifier` can be unpickled.

Parameters  
**state** *(dict)* – State dictionary with instance parameters to restore.

`class_gradient`(x, label=None, logits=False, **kwargs)

Compute per-class derivatives w.r.t. `x`.

Parameters

- **x** *(np.ndarray)* – Sample input with shape as expected by the model.
- **label** *(int or list)* – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of `x`, and each value will be used as target for its corresponding sample in `x`. If `None`, then gradients for all classes will be computed for each sample.
- **logits** *(bool)* – True if the prediction should be done at the logits layer.

Returns  
Array of gradients of input features w.r.t. each class in the form `(batch_size, nb_classes, input_shape)` when computing for all classes, otherwise shape becomes `(batch_size, 1, input_shape)` when `label` parameter is specified.

Return type  np.ndarray
**fit** \((x, y, batch\_size=128, nb\_epochs=20, **kwargs)\)

Fit the classifier on the training set \((x, y)\).

**Parameters**

- \(x\) *(np.ndarray)* – Training data.
- \(y\) *(np.ndarray)* – Labels, one-vs-rest encoding.
- \(\text{batch\_size}\) *(int)* – Size of batches.
- \(\text{nb\_epochs}\) *(int)* – Number of epochs to use for training.
- \(\text{kwargs}\) *(dict)* – Dictionary of framework-specific arguments. These should be parameters supported by the \(\text{fit}\_\text{generator}\) function in Keras and will be passed to this function as such. Including the number of epochs or the number of steps per epoch as part of this argument will result in an error.

**Returns** None

**fit\_generator** *(generator, nb\_epochs=20, **kwargs)\)

Fit the classifier using the generator that yields batches as specified.

**Parameters**

- \(\text{generator}\) *(DataGenerator)* – Batch generator providing \((x, y)\) for each epoch. If the generator can be used for native training in Keras, it will.
- \(\text{nb\_epochs}\) *(int)* – Number of epochs to use for training.
- \(\text{kwargs}\) *(dict)* – Dictionary of framework-specific arguments. These should be parameters supported by the \(\text{fit}\_\text{generator}\) function in Keras and will be passed to this function as such. Including the number of epochs as part of this argument will result in an error.

**Returns** None

**get\_activations** *(x, layer, batch\_size=128)\)

Return the output of the specified layer for input \(x\). \(layer\) is specified by layer index (between 0 and \(\text{nb\_layers} - 1\)) or by name. The number of layers can be determined by counting the results returned by calling \(\text{layer\_names}\).

**Parameters**

- \(x\) *(np.ndarray)* – Input for computing the activations.
- \(\text{layer}\) *(int or str)* – Layer for computing the activations
- \(\text{batch\_size}\) *(int)* – Size of batches.

**Returns** The output of \(layer\), where the first dimension is the batch size corresponding to \(x\).

**Return type** np.ndarray

**property layer\_names**

Return the hidden layers in the model, if applicable.

**Returns** The hidden layers in the model, input and output layers excluded.

**Return type** list

**Warning:** \(\text{layer\_names}\) tries to infer the internal structure of the model. This feature comes with no guarantees on the correctness of the result. The intended order of the layers tries to match their order in the model, but this is not guaranteed either.
loss_gradient(x, y, **kwargs)
Compute the gradient of the loss function w.r.t. x.

Parameters

- **x (np.ndarray)** – Sample input with shape as expected by the model.
- **y (np.ndarray)** – Correct labels, one-vs-rest encoding.

Returns

Array of gradients of the same shape as x.

Return type

np.ndarray

predict(x, logits=False, batch_size=128, **kwargs)
Perform prediction for a batch of inputs.

Parameters

- **x (np.ndarray)** – Test set.
- **logits (bool)** – True if the prediction should be done at the logits layer.
- **batch_size (int)** – Size of batches.

Returns

Array of predictions of shape (nb_inputs, self.nb_classes).

Return type

np.ndarray

save(filename, path=None)
Save a model to file in the format specific to the backend framework. For Keras, .h5 format is used.

Parameters

- **filename (str)** – Name of the file where to store the model.
- **path (str)** – Path of the folder where to store the model. If no path is specified, the model
  will be stored in the default data location of the library DATA_PATH.

Returns

None

set_learning_phase(train)
Set the learning phase for the backend framework.

Parameters

- **train (bool)** – True to set the learning phase to training, False to set it to predic-

1.4.2 MXNet Wrapper

class art.classifiers.MXClassifier(model, input_shape, nb_classes, optimizer=None, ctx=None, channel_index=1, clip_values=None, defences=None, preprocessing=(0, 1))
Wrapper class for importing MXNet Gluon model.

__init__(model, input_shape, nb_classes, optimizer=None, ctx=None, channel_index=1, clip_values=None, defences=None, preprocessing=(0, 1))
Initialize an MXClassifier object. Assumes the model passed as parameter is a Gluon model and that the
loss function is the softmax cross-entropy.

Parameters

- **model (mxnet.gluon.Block)** – The model with logits as expected output.
- **input_shape (tuple)** – The shape of one input instance.
- **nb_classes (int)** – The number of classes of the model.
• **optimizer** *(mxnet.gluon.Trainer)* – The optimizer used to train the classifier. This parameter is not required if no training is used.

• **ctx** *(mxnet.context.Context)* – The device on which the model runs (CPU or GPU). If not provided, CPU is assumed.

• **channel_index** *(int)* – Index of the axis in data containing the color channels or features.

• **clip_values** *(tuple)* – Tuple of the form *(min, max)* of floats or *np.ndarray* representing the minimum and maximum values allowed for features. If floats are provided, these will be used as the range of all features. If arrays are provided, each value will be considered the bound for a feature, thus the shape of clip values needs to match the total number of features.

• **defences** *(str or list(str))* – Defences to be activated with the classifier.

• **preprocessing** *(tuple)* – Tuple of the form *(substracter, divider)* of floats or *np.ndarray* of values to be used for data preprocessing. The first value will be substracted from the input. The input will then be divided by the second one.

```python
__repr__(self)
```

- Return `repr(self)`.

```python
class_gradient(x, label=None, logits=False, **kwargs)
```

Compute per-class derivatives w.r.t. `x`.

**Parameters**

- **x** *(np.ndarray)* – Sample input with shape as expected by the model.

- **label** *(int or list)* – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of `x`, and each value will be used as target for its corresponding sample in `x`. If `None`, then gradients for all classes will be computed for each sample.

- **logits** *(bool)* – True if the prediction should be done at the logits layer.

**Returns** Array of gradients of input features w.r.t. each class in the form *(batch_size, nb_classes, input_shape)* when computing for all classes, otherwise shape becomes *(batch_size, 1, input_shape)* when `label` parameter is specified.

**Return type** *np.ndarray*

```python
fit(x, y, batch_size=128, nb_epochs=20, **kwargs)
```

Fit the classifier on the training set *(inputs, outputs)*.

**Parameters**

- **x** *(np.ndarray)* – Training data.

- **y** *(np.ndarray)* – Labels, one-vs-rest encoding.

- **batch_size** *(int)* – Size of batches.

- **nb_epochs** *(int)* – Number of epochs to use for training.

- **kwargs** *(dict)* – Dictionary of framework-specific arguments. This parameter is not currently supported for MXNet and providing it takes no effect.

**Returns** None

```python
fit_generator(generator, nb_epochs=20, **kwargs)
```

Fit the classifier using the generator that yields batches as specified.
Parameters

• `generator (DataGenerator)` – Batch generator providing \((x, y)\) for each epoch.

• `nb_epochs (int)` – Number of epochs to use for training.

• `kwargs (dict)` – Dictionary of framework-specific arguments. This parameter is not currently supported for MXNet and providing it takes no effect.

Returns None

`get_activations (x, layer, batch_size=128)`
Return the output of the specified layer for input \(x\). \(layer\) is specified by layer index (between 0 and \(nb\_layers - 1\)) or by name. The number of layers can be determined by counting the results returned by calling `layer_names`.

Parameters

• `x (np.ndarray)` – Input for computing the activations.

• `layer (int or str)` – Layer for computing the activations

• `batch_size (int)` – Size of batches.

Returns The output of \(layer\), where the first dimension is the batch size corresponding to \(x\).

Return type `np.ndarray`

`property layer_names`  
Return the hidden layers in the model, if applicable.

Returns The hidden layers in the model, input and output layers excluded.

Return type `list`

Warning: `layer_names` tries to infer the internal structure of the model. This feature comes with no guarantees on the correctness of the result. The intended order of the layers tries to match their order in the model, but this is not guaranteed either.

`loss_gradient (x, y, **kwargs)`
Compute the gradient of the loss function w.r.t. \(x\).

Parameters

• `x (np.ndarray)` – Sample input with shape as expected by the model.

• `y (np.ndarray)` – Correct labels, one-vs-rest encoding.

Returns Array of gradients of the same shape as \(x\).

Return type `np.ndarray`

`predict (x, logits=False, batch_size=128, **kwargs)`
Perform prediction for a batch of inputs.

Parameters

• `x (np.ndarray)` – Test set.

• `logits (bool)` – True if the prediction should be done at the logits layer.

• `batch_size (int)` – Size of batches.

Returns Array of predictions of shape \((nb\_inputs, self.nb\_classes)\).

Return type `np.ndarray`
save (filename, path=None)
Save a model to file in the format specific to the backend framework. For Gluon, only parameters are saved in file with name <filename>.params at the specified path. To load the saved model, the original model code needs to be run before calling load_parameters on the generated Gluon model.

Parameters

- **filename** (str) – Name of the file where to store the model.
- **path** (str) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library DATA_PATH.

Returns None

set_learning_phase (train)
Set the learning phase for the backend framework.

Parameters **train** (bool) – True to set the learning phase to training, False to set it to prediction.

1.4.3 PyTorch Wrapper

class art.classifiers.PyTorchClassifier (model, loss, optimizer, input_shape, nb_classes, channel_index=1, clip_values=None, defences=None, preprocessing=(0, 1))

This class implements a classifier with the PyTorch framework.

__getstate__ ()
Use to ensure PytorchClassifier can be pickled.

Returns State dictionary with instance parameters.

Return type dict

__init__ (model, loss, optimizer, input_shape, nb_classes, channel_index=1, clip_values=None, defences=None, preprocessing=(0, 1))

Initialization specifically for the PyTorch-based implementation.

Parameters

- **model** (is instance of torch.nn.Module) – PyTorch model. The forward function of the model must return the logit output.
- **loss** (torch.nn.modules.loss._Loss) – The loss function for which to compute gradients for training. The target label must be raw categorical, i.e. not converted to one-hot encoding.
- **optimizer** (torch.optim.Optimizer) – The optimizer used to train the classifier.
- **input_shape** (tuple) – The shape of one input instance.
- **nb_classes** (int) – The number of classes of the model.
- **channel_index** (int) – Index of the axis in data containing the color channels or features.
- **clip_values** (tuple) – Tuple of the form (min, max) of floats or np.ndarray representing the minimum and maximum values allowed for features. If floats are provided, these will be used as the range of all features. If arrays are provided, each value will be considered the bound for a feature, thus the shape of clip values needs to match the total number of features.
- **defences** (str or list(str)) – Defences to be activated with the classifier.
• **preprocessing** *(tuple)* – Tuple of the form *(subtractor, divider)* of floats or *np.ndarray* of values to be used for data preprocessing. The first value will be substracted from the input. The input will then be divided by the second one.

```python
__repr__
Return repr(self).
```

```python
__setstate__(state)
Use to ensure PytorchClassifier can be unpickled.
Parameters

* state (dict) – State dictionary with instance parameters to restore.
```

```python
class_gradient(x, label=None, logits=False, **kwargs)
Compute per-class derivatives w.r.t. x.
Parameters

* x (np.ndarray) – Sample input with shape as expected by the model.
* label (int or list) – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of x, and each value will be used as target for its corresponding sample in x. If None, then gradients for all classes will be computed for each sample.
* logits (bool) – True if the prediction should be done at the logits layer.

Returns

Array of gradients of input features w.r.t. each class in the form *(batch size, nb_classes, input_shape)* when computing for all classes, otherwise shape becomes *(batch size, 1, input_shape)* when label parameter is specified.
```

```python
Return type
np.ndarray
```

```python
fit(x, y, batch_size=128, nb_epochs=10, **kwargs)
Fit the classifier on the training set (x, y).
Parameters

* x (np.ndarray) – Training data.
* y (np.ndarray) – Labels, one-vs-rest encoding.
* batch_size (int) – Size of batches.
* nb_epochs (int) – Number of epochs to use for training.
* kwargs (dict) – Dictionary of framework-specific arguments. This parameter is not currently supported for PyTorch and providing it takes no effect.

Returns

None
```

```python
fit_generator(generator, nb_epochs=20, **kwargs)
Fit the classifier using the generator that yields batches as specified.
Parameters

* generator (DataGenerator) – Batch generator providing (x, y) for each epoch.
* nb_epochs (int) – Number of epochs to use for training.
* kwargs (dict) – Dictionary of framework-specific arguments. This parameter is not currently supported for PyTorch and providing it takes no effect.

Returns

None
```

28 Chapter 1. Supported Attacks, Defences and Metrics
**get_activations** *(x, layer, batch_size=128)*

Return the output of the specified layer for input *x*. *layer* is specified by layer index (between 0 and *nb_layers - 1*) or by name. The number of layers can be determined by counting the results returned by calling *layer_names*.

**Parameters**

- **x** (*np.ndarray*) – Input for computing the activations.
- **layer** (*int* or *str*) – Layer for computing the activations
- **batch_size** (*int*) – Size of batches.

**Returns** The output of *layer*, where the first dimension is the batch size corresponding to *x*.

**Return type** *np.ndarray*

**property layer_names**

Return the hidden layers in the model, if applicable.

**Returns** The hidden layers in the model, input and output layers excluded.

**Return type** *list*

**Warning:** *layer_names* tries to infer the internal structure of the model. This feature comes with no guarantees on the correctness of the result. The intended order of the layers tries to match their order in the model, but this is not guaranteed either. In addition, the function can only infer the internal layers if the input model is of type *nn.Sequential*, otherwise, it will only return the logit layer.

**loss_gradient** *(x, y, **kwargs)*

Compute the gradient of the loss function w.r.t. *x*.

**Parameters**

- **x** (*np.ndarray*) – Sample input with shape as expected by the model.
- **y** (*np.ndarray*) – Correct labels, one-vs-rest encoding.

**Returns** Array of gradients of the same shape as *x*.

**Return type** *np.ndarray*

**predict** *(x, logits=False, batch_size=128, **kwargs)*

Perform prediction for a batch of inputs.

**Parameters**

- **logits** (*bool*) – True if the prediction should be done at the logits layer.
- **batch_size** (*int*) – Size of batches.

**Returns** Array of predictions of shape *(nb_inputs, self.nb_classes)*.

**Return type** *np.ndarray*

**save** *(filename, path=None)*

Save a model to file in the format specific to the backend framework.

**Parameters**

- **filename** (*str*) – Name of the file where to store the model.
• **path** (*str*) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library `DATA_PATH`.

Returns None

**set_learning_phase** (*train*)
Set the learning phase for the backend framework.

Parameters **train** (*bool*) – True to set the learning phase to training, False to set it to prediction.

### 1.4.4 Tensorflow Wrapper

```python
class art.classifiers.TFClassifier(input_ph, logits, output_ph=None, train=None, loss=None, learning=None, sess=None, channel_index=3, clip_values=None, defences=None, preprocessing=(0, 1))
```

This class implements a classifier with the Tensorflow framework.

**__getstate__** ()
Use to ensure `TFClassifier` can be pickled.

Returns State dictionary with instance parameters.

Return type **dict**

**__init__** (*input_ph*, *logits*, *output_ph=None*, *train=None*, *loss=None*, *learning=None*, *sess=None*, *channel_index=3*, *clip_values=None*, *defences=None*, *preprocessing=(0, 1))

Initialization specific to Tensorflow models implementation.

Parameters

• **input_ph** (*tf.Placeholder*) – The input placeholder.

• **logits** (*tf.Tensor*) – The logits layer of the model.

• **output_ph** (*tf.Tensor*) – The labels placeholder of the model. This parameter is necessary when training the model and when computing gradients w.r.t. the loss function.

• **train** (*tf.Tensor*) – The train tensor for fitting, including an optimizer. Use this parameter only when training the model.

• **loss** (*tf.Tensor*) – The loss function for which to compute gradients. This parameter is necessary when training the model and when computing gradients w.r.t. the loss function.

• **learning** (*tf.Placeholder of type bool.*) – The placeholder to indicate if the model is training.

• **sess** (*tf.Session*) – Computation session.

• **channel_index** (*int*) – Index of the axis in data containing the color channels or features.

• **clip_values** (*tuple*) – Tuple of the form *(min, max)* of floats or *np.ndarray* representing the minimum and maximum values allowed for features. If floats are provided, these will be used as the range of all features. If arrays are provided, each value will be considered the bound for a feature, thus the shape of clip values needs to match the total number of features.

• **defences** (*str or list(str]*) – Defences to be activated with the classifier.
• **preprocessing** *(tuple)* – Tuple of the form *(subtractor, divider)* of floats or *np.ndarray* of values to be used for data preprocessing. The first value will be substracted from the input. The input will then be divided by the second one.

```python
__repr__(self)
```

Return `repr(self)`.

```python
__setstate__(state)
```

Use to ensure `TFClassifier` can be unpickled.

**Parameters**

- **state** *(dict)* – State dictionary with instance parameters to restore.

```python
class_gradient(x, label=None, logits=False, **kwargs)
```

Compute per-class derivatives w.r.t. `x`.

**Parameters**

- **x** *(np.ndarray)* – Sample input with shape as expected by the model.
- **label** *(int or list)* – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of `x`, and each value will be used as target for its corresponding sample in `x`. If `None`, then gradients for all classes will be computed for each sample.
- **logits** *(bool)* – `True` if the prediction should be done at the logits layer.

**Returns**

Array of gradients of input features w.r.t. each class in the form *(batch_size, nb_classes, input_shape)* when computing for all classes, otherwise shape becomes *(batch_size, 1, input_shape)* when `label` parameter is specified.

**Return type** *np.ndarray*

```python
fit(x, y, batch_size=128, nb_epochs=10, **kwargs)
```

Fit the classifier on the training set *(x, y)*.

**Parameters**

- **x** *(np.ndarray)* – Training data.
- **y** *(np.ndarray)* – Labels, one-vs-rest encoding.
- **batch_size** *(int)* – Size of batches.
- **nb_epochs** *(int)* – Number of epochs to use for training.
- **kwargs** *(dict)* – Dictionary of framework-specific arguments. This parameter is not currently supported for TensorFlow and providing it takes no effect.

**Returns** *None*

```python
fit_generator(generator, nb_epochs=20, **kwargs)
```

Fit the classifier using the generator that yields batches as specified.

**Parameters**

- **generator** *(DataGenerator)* – Batch generator providing *(x, y)* for each epoch. If the generator can be used for native training in TensorFlow, it will.
- **nb_epochs** *(int)* – Number of epochs to use for training.
- **kwargs** *(dict)* – Dictionary of framework-specific arguments. This parameter is not currently supported for TensorFlow and providing it takes no effect.

**Returns** *None*
get_activations(x, layer, batch_size=128)
Return the output of the specified layer for input x. layer is specified by layer index (between 0 and nb_layers - 1) or by name. The number of layers can be determined by counting the results returned by calling layer_names.

Parameters
• x (np.ndarray) – Input for computing the activations.
• layer (int or str) – Layer for computing the activations
• batch_size (int) – Size of batches.

Returns The output of layer, where the first dimension is the batch size corresponding to x.

Return type np.ndarray

property layer_names
Return the hidden layers in the model, if applicable.

Returns The hidden layers in the model, input and output layers excluded.

Return type list

Warning: layer_names tries to infer the internal structure of the model. This feature comes with no guarantees on the correctness of the result. The intended order of the layers tries to match their order in the model, but this is not guaranteed either.

loss_gradient(x, y, **kwargs)
Compute the gradient of the loss function w.r.t. x.

Parameters
• x (np.ndarray) – Sample input with shape as expected by the model.
• y (np.ndarray) – Correct labels, one-vs-rest encoding.

Returns Array of gradients of the same shape as x.

Return type np.ndarray

predict(x, logits=False, batch_size=128, **kwargs)
Perform prediction for a batch of inputs.

Parameters
• x (np.ndarray) – Test set.
• logits (bool) – True if the prediction should be done at the logits layer.
• batch_size (int) – Size of batches.

Returns Array of predictions of shape (nb_inputs, self.nb_classes).

Return type np.ndarray

save(filename, path=None)
Save a model to file in the format specific to the backend framework. For TensorFlow, .ckpt is used.

Parameters
• filename (str) – Name of the file where to store the model.
• path (str) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library DATA_PATH.
Returns None

```python
set_learning_phase(train)
```
Set the learning phase for the backend framework.

**Parameters**

- `train` *(bool)* – True to set the learning phase to training, False to set it to prediction.

1.4.5 Ensemble Wrapper

```python
class art.classifiers.EnsembleClassifier(classifiers, classifier_weights=None, channel_index=3, clip_values=None, defences=None, preprocessing=(0, 1))
```
Class allowing to aggregate multiple classifiers as an ensemble. The individual classifiers are expected to be trained when the ensemble is created and no training procedures are provided through this class.

```python
__init__(classifiers, classifier_weights=None, channel_index=3, clip_values=None, defences=None, preprocessing=(0, 1))
```
Initialize a `EnsembleClassifier` object. The data range values and colour channel index have to be consistent for all the classifiers in the ensemble.

**Parameters**

- `classifiers` *(list)* – List of `Classifier` instances to be ensembled together.
- `classifier_weights` *(list or np.ndarray or None)* – List of weights, one scalar per classifier, to assign to their prediction when aggregating results. If `None`, all classifiers are assigned the same weight.
- `channel_index` *(int)* – Index of the axis in data containing the color channels or features.
- `clip_values` *(tuple)* – Tuple of the form `(min, max)` of floats or `np.ndarray` representing the minimum and maximum values allowed for features. If floats are provided, these will be used as the range of all features. If arrays are provided, each value will be considered the bound for a feature, thus the shape of clip values needs to match the total number of features.
- `defences` *(str or list(str))* – Defences to be activated with the classifier.
- `preprocessing` *(tuple)* – Tuple of the form `(subtractor, divider)` of floats or `np.ndarray` of values to be used for data preprocessing. The first value will be substracted from the input. The input will then be divided by the second one.

```python
__repr__(self).
```
Return repr(self).

```python
class_gradient(x, label=None, logits=False, **kwargs)
```
Compute per-class derivatives w.r.t. `x`.

**Parameters**

- `x` *(np.ndarray)* – Sample input with shape as expected by the model.
- `label` *(int)* – Index of a specific per-class derivative. If `None`, then gradients for all classes will be computed.
- `logits` *(bool)* – True if the prediction should be done at the logits layer.
- `raw` *(bool)* – Return the individual classifier raw outputs (not aggregated).
Returns  Array of gradients of input features w.r.t. each class in the form (batch_size, nb_classes, input_shape) when computing for all classes, otherwise shape becomes (batch_size, 1, input_shape) when label parameter is specified. If raw=True, an additional dimension is added at the beginning of the array, indexing the different classifiers.

Return type  np.ndarray

fit (x, y, batch_size=128, nb_epochs=20, **kwargs)
Fit the classifier on the training set (x, y). This function is not supported for ensembles.

Parameters

• x (np.ndarray) – Training data.
• y (np.ndarray) – Labels, one-vs-rest encoding.
• batch_size (int) – Size of batches.
• nb_epochs (int) – Number of epochs to use for training.
• kwargs (dict) – Dictionary of framework-specific arguments.

Returns  None

fit_generator (generator, nb_epochs=20, **kwargs)
Fit the classifier using the generator that yields batches as specified. This function is not supported for ensembles.

Parameters

• generator (DataGenerator) – Batch generator providing (x, y) for each epoch. If the generator can be used for native training in Keras, it will.
• nb_epochs (int) – Number of epochs to use for trainings.
• kwargs (dict) – Dictionary of framework-specific argument.

Returns  None

get_activations (x, layer, batch_size=128)
Return the output of the specified layer for input x. layer is specified by layer index (between 0 and nb_layers - 1) or by name. The number of layers can be determined by counting the results returned by calling layer_names. This function is not supported for ensembles.

Parameters

• x (np.ndarray) – Input for computing the activations.
• layer (int or str) – Layer for computing the activations
• batch_size (int) – Size of batches.

Returns  The output of layer, where the first dimension is the batch size corresponding to x.

Return type  np.ndarray

property layer_names

Return the hidden layers in the model, if applicable. This function is not supported for ensembles.

Returns  The hidden layers in the model, input and output layers excluded.

Return type  list
**Warning:** *layer_names* tries to infer the internal structure of the model. This feature comes with no guarantees on the correctness of the result. The intended order of the layers tries to match their order in the model, but this is not guaranteed either.

**loss_gradient** *(x, y, **kwargs)*
Compute the gradient of the loss function w.r.t. *x*.

**Parameters**
- *x* (*np.ndarray*) – Sample input with shape as expected by the model.
- *y* (*np.ndarray*) – Correct labels, one-vs-rest encoding.
- *raw* (*bool*) – Return the individual classifier raw outputs (not aggregated).

**Returns** Array of gradients of the same shape as *x*. If *raw=True*, shape becomes [nb_classifiers, x.shape].

**Return type** *np.ndarray*

**predict** *(x, logits=False, batch_size=128, **kwargs)*
Perform prediction for a batch of inputs. Predictions from classifiers are aggregated at probabilities level, as logits are not comparable between models. If logits prediction was specified, probabilities are converted back to logits after aggregation.

**Parameters**
- *logits* (*bool*) – *True* if the prediction should be done at the logits layer.
- *raw* (*bool*) – Return the individual classifier raw outputs (not aggregated).

**Returns** Array of predictions of shape (nb_inputs, self.nb_classes), or of shape (nb_classifiers, nb_inputs, self.nb_classes) if *raw=True*.

**Return type** *np.ndarray*

**save** *(filename, path=None)*
Save a model to file in the format specific to the backend framework. This function is not supported for ensembles.

**Parameters**
- *filename* (*str*) – Name of the file where to store the model.
- *path* (*str*) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library *DATA_PATH*.

**Returns** None

**set_learning_phase** *(train)*
Set the learning phase for the backend framework.

**Parameters**
- *train* (*bool*) – *True* to set the learning phase to training, *False* to set it to prediction.

1.4.6 Base Class

**class** *art.classifiers.Classifier* *(channel_index, clip_values=None, defences=None, preprocessing=(0, 1))*
Base class for all classifiers.
property channel_index

Returns  Index of the axis in data containing the color channels or features.

Return type  int

abstract class_gradient  (x, label=None, logits=False, **kwargs)

Compute per-class derivatives w.r.t. x.

Parameters

•  x (np.ndarray) – Sample input with shape as expected by the model.

•  label (int or list) – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of x, and each value will be used as target for its corresponding sample in x. If None, then gradients for all classes will be computed for each sample.

•  logits (bool) – True if the prediction should be done at the logits layer.

Returns  Array of gradients of input features w.r.t. each class in the form (batch_size, nb_classes, input_shape) when computing for all classes, otherwise shape becomes (batch_size, 1, input_shape) when label parameter is specified.

Return type  np.ndarray

property clip_values

Returns  Tuple of the form (min, max) representing the minimum and maximum values allowed for features.

Return type  tuple

abstract fit  (x, y, batch_size=128, nb_epochs=20, **kwargs)

Fit the classifier on the training set (x, y).

Parameters

•  x (np.ndarray) – Training data.

•  y (np.ndarray) – Labels, one-vs-rest encoding.

•  batch_size (int) – Size of batches.

•  nb_epochs (int) – Number of epochs to use for training.

•  kwargs (dict) – Dictionary of framework-specific arguments.

Returns  None

fit_generator  (generator, nb_epochs=20, **kwargs)

Fit the classifier using the generator gen that yields batches as specified. Framework implementations can provide framework-specific versions of this function to speed-up computation.

Parameters

•  generator (DataGenerator) – Batch generator providing (x, y) for each epoch.

•  nb_epochs (int) – Number of epochs to use for training.

•  kwargs (dict) – Dictionary of framework-specific arguments.

Returns  None
**abstract** get_activations \((x, layer, batch\_size)\)

Return the output of the specified layer for input \(x\). \(layer\) is specified by layer index (between 0 and \(nb\_layers - 1\)) or by name. The number of layers can be determined by counting the results returned by calling layer_names.

**Parameters**

- \(x\) (*np.ndarray*) – Input for computing the activations.
- \(layer\) (*int or str*) – Layer for computing the activations
- \(batch\_size\) (*int*) – Size of batches.

**Returns** The output of \(layer\), where the first dimension is the batch size corresponding to \(x\).

**Return type** *np.ndarray*

**property** input_shape

Return the shape of one input.

**Returns** Shape of one input for the classifier.

**Return type** *tuple*

**property** layer_names

Return the hidden layers in the model, if applicable.

**Returns** The hidden layers in the model, input and output layers excluded.

**Return type** *list*

**Warning:** layer_names tries to infer the internal structure of the model. This feature comes with no guarantees on the correctness of the result. The intended order of the layers tries to match their order in the model, but this is not guaranteed either.

**property** learning_phase

Return the learning phase set by the user for the current classifier. Possible values are True for training, False for prediction and None if it has not been set through the library. In the latter case, the library does not do any explicit learning phase manipulation and the current value of the backend framework is used. If a value has been set by the user for this property, it will impact all following computations for model fitting, prediction and gradients.

**Returns** Value of the learning phase.

**Return type** *bool* or *None*

**abstract** loss_gradient \((x, y, **kwargs)\)

Compute the gradient of the loss function w.r.t. \(x\).

**Parameters**

- \(x\) (*np.ndarray*) – Sample input with shape as expected by the model.
- \(y\) (*np.ndarray*) – Correct labels, one-vs-rest encoding.

**Returns** Array of gradients of the same shape as \(x\).

**Return type** *np.ndarray*

**property** nb_classes

Return the number of output classes.

**Returns** Number of classes in the data.
abstract predict (x, logits=False, batch_size=128, **kwargs)
    Perform prediction for a batch of inputs.

    Parameters
    • x (np.ndarray) – Test set.
    • logits (bool) – True if the prediction should be done at the logits layer.
    • batch_size (int) – Size of batches.

    Returns Array of predictions of shape (nb_inputs, self.nb_classes).
    Return type np.ndarray

abstract save (filename, path=None)
    Save a model to file in the format specific to the backend framework.

    Parameters
    • filename (str) – Name of the file where to store the model.
    • path (str) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library DATA_PATH.

    Returns None

abstract set_learning_phase (train)
    Set the learning phase for the backend framework.

    Parameters train (bool) – True to set the learning phase to training, False to set it to prediction.

1.5 art.data_generators

Module defining an interface for data generators and providing concrete implementations for the supported frameworks. Their purpose is to allow for data loading and batching on the fly, as well as dynamic data augmentation. The generators can be used with the fit_generator function in the Classifier interface. Users can define their own generators following the DataGenerator interface.

1.5.1 Base Class

class art.data_generators.DataGenerator (size, batch_size)
    Base class for data generators.

    abstract get_batch ()
        Provide the next batch for training in the form of a tuple (x, y). The generator should loop over the data indefinitely.

        Returns A tuple containing a batch of data (x, y).
        Return type tuple
1.5.2 Framework-Specific Data Generators

```python
class art.data_generators.KerasDataGenerator(generator, size, batch_size)
Wrapper class on top of the Keras-native data generators. These can either be generator functions, keras.utils.Sequence or Keras-specific data generators (keras.preprocessing.image.ImageDataGenerator).

def get_batch() -> None:
    Provide the next batch for training in the form of a tuple (x, y). The generator should loop over the data indefinitely.

    Returns: A tuple containing a batch of data (x, y).
    Return type: tuple
```

```python
class art.data_generators.MXDataGenerator(data_loader, size, batch_size)
Wrapper class on top of the MXNet/Gluon native data loader mxnet.gluon.data.DataLoader.

def get_batch() -> None:
    Provide the next batch for training in the form of a tuple (x, y). The generator should loop over the data indefinitely.

    Returns: A tuple containing a batch of data (x, y).
    Return type: tuple
```

```python
class art.data_generators.PyTorchDataGenerator(data_loader, size, batch_size)
Wrapper class on top of the PyTorch native data loader torch.utils.data.DataLoader.

def get_batch() -> None:
    Provide the next batch for training in the form of a tuple (x, y). The generator should loop over the data indefinitely.

    Returns: A tuple containing a batch of data (x, y).
    Return type: tuple
```

```python
class art.data_generators.TFDataGenerator(sess, iterator, iterator_type, iterator_arg, size, batch_size)
Wrapper class on top of the TensorFlow native iterators tf.data.Iterator.

def get_batch() -> None:
    Provide the next batch for training in the form of a tuple (x, y). The generator should loop over the data indefinitely.

    Returns: A tuple containing a batch of data (x, y).
    Return type: tuple
    Raises: ValueError if the iterator has reached the end.
```

1.6 art.defences

Module implementing multiple types of defences against adversarial attacks.

1.6.1 Feature Squeezing

```python
class art.defences.FeatureSqueezing(clip_values, bit_depth=8, apply_fit=False, apply_predict=True)
```
__call__ (x, y=None)
Apply feature squeezing to sample x.

Parameters
- x (np.ndarray) – Sample to squeeze. x values are expected to be in the data range provided by clip_values.
- y (np.ndarray) – Labels of the sample x. This function does not affect them in any way.

Returns Squeezed sample.
Return type np.ndarray

__init__ (clip_values, bit_depth=8, apply_fit=False, apply_predict=True)
Create an instance of feature squeezing.

Parameters
- clip_values (tuple) – Tuple of the form (min, max) representing the minimum and maximum values allowed for features.
- bit_depth (int) – The number of bits per channel for encoding the data.
- apply_fit (bool) – True if applied during fitting/training.
- apply_predict (bool) – True if applied during predicting.

property apply_fit
Property of the defence indicating if it should be applied at training time.

Returns True if the defence should be applied when fitting a model, False otherwise.
Return type bool

property apply_predict
Property of the defence indicating if it should be applied at test time.

Returns True if the defence should be applied at prediction time, False otherwise.
Return type bool

estimate_gradient (x, grad)
Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differentiable, this is an estimate of the gradient, most often replacing the computation performed by the defence with the identity function.

Parameters
- x (np.ndarray) – Input data for which the gradient is estimated. First dimension is the batch size.
- grad (np.ndarray) – Gradient value so far.

Returns The gradient (estimate) of the defence.
Return type np.ndarray

fit (x, y=None, **kwargs)
No parameters to learn for this method; do nothing.

set_params (**kwargs)
Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

Parameters
• **clip_values** *(tuple)* – Tuple of the form *(min, max)* representing the minimum and maximum values allowed for features.

• **bit_depth** *(int)* – The number of bits per channel for encoding the data.

### 1.6.2 Spatial Smoothing

**class** art.defences.SpatialSmoothing*(window_size=3, channel_index=3, clip_values=None, apply_fit=False, apply_predict=True)*


**__call__**(x, y=None)

Apply local spatial smoothing to sample x.

**Parameters**

- **x** *(np.ndarray)* – Sample to smooth with shape *(batch_size, width, height, depth).*
- **y** *(np.ndarray)* – Labels of the sample x. This function does not affect them in any way.

**Returns** Smoothed sample

**Return type** np.ndarray

**__init__**(window_size=3, channel_index=3, clip_values=None, apply_fit=False, apply_predict=True)

Create an instance of local spatial smoothing.

**Parameters**

- **channel_index** *(int)* – Index of the axis in data containing the color channels or features.
- **window_size** *(int)* – The size of the sliding window.
- **clip_values** *(tuple)* – Tuple of the form *(min, max)* representing the minimum and maximum values allowed for features.
- **apply_fit** *(bool)* – True if applied during fitting/training.
- **apply_predict** *(bool)* – True if applied during predicting.

**property apply_fit**

Property of the defence indicating if it should be applied at training time.

**Returns** True if the defence should be applied when fitting a model, False otherwise.

**Return type** bool

**property apply_predict**

Property of the defence indicating if it should be applied at test time.

**Returns** True if the defence should be applied at prediction time, False otherwise.

**Return type** bool

**estimate_gradient**(x, grad)

Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differentiable, this is an estimate of the gradient, most often replacing the computation performed by the defence with the identity function.

**Parameters**
• \textbf{x} (np.ndarray) – Input data for which the gradient is estimated. First dimension is the batch size.

• \textbf{grad} (np.ndarray) – Gradient value so far.

\textbf{Returns} The gradient (estimate) of the defence.

\textbf{Return type} np.ndarray

\textbf{fit}(x, y=None, **kwargs)

No parameters to learn for this method; do nothing.

\textbf{set_params}(**kwargs)

Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

\textbf{Parameters}

• \textbf{window_size} (int) – The size of the sliding window.

• \textbf{channel_index} (int) – Index of the axis in data containing the color channels or features.

• \textbf{clip_values} (tuple) – Tuple of the form \((\text{min, max})\) representing the minimum and maximum values allowed for features.

\subsection{1.6.3 Label Smoothing}

\texttt{class} \texttt{art.defences.LabelSmoothing}(\texttt{max_value=0.9, apply_fit=True, apply_predict=False})

Computes a vector of smooth labels from a vector of hard ones. The hard labels have to contain ones for the correct classes and zeros for all the others. The remaining probability mass between \(\text{max\_value}\) and 1 is distributed uniformly between the incorrect classes for each instance.

\texttt{__call__}(x, y=None)

Apply label smoothing.

\textbf{Parameters}

• \textbf{x} (np.ndarray) – Input data, will not be modified by this method

• \textbf{y} (np.ndarray) – Original vector of label probabilities (one-vs-rest)

\textbf{Returns} Unmodified input data and the vector of smooth probabilities as correct labels

\textbf{Return type} (np.ndarray, np.ndarray)

\texttt{__init__}(\texttt{max_value=0.9, apply_fit=True, apply_predict=False})

Create an instance of label smoothing.

\textbf{Parameters}

• \textbf{max_value} (float) – Value to affect to correct label

• \textbf{apply_fit} (bool) – True if applied during fitting/training.

• \textbf{apply_predict} (bool) – True if applied during predicting.

\textbf{property apply_fit}

Property of the defence indicating if it should be applied at training time.

\textbf{Returns} \textit{True} if the defence should be applied when fitting a model, \textit{False} otherwise.

\textbf{Return type} bool

\textbf{property apply_predict}

Property of the defence indicating if it should be applied at test time.
Returns True if the defence should be applied at prediction time, False otherwise.

Return type bool

estimate_gradient (x, grad)

Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differentiable, this is an estimate of the gradient, most often replacing the computation performed by the defence with the identity function.

Parameters

- x (np.ndarray) – Input data for which the gradient is estimated. First dimension is the batch size.
- grad (np.ndarray) – Gradient value so far.

Returns The gradient (estimate) of the defence.

Return type np.ndarray

fit (x, y=None, **kwargs)

No parameters to learn for this method; do nothing.

set_params (**kwargs)

Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

Defense-specific parameters: :param max_value: Value to affect to correct label :type max_value: float

1.6.4 Adversarial Training

class art.defences.AdversarialTrainer (classifier, attacks, ratio=0.5)

Class performing adversarial training based on a model architecture and one or multiple attack methods.

Incorporates original adversarial training, ensemble adversarial training (https://arxiv.org/abs/1705.07204), training on all adversarial data and other common setups. If multiple attacks are specified, they are rotated for each batch. If the specified attacks have as target a different model, then the attack is transferred. The ratio determines how many of the clean samples in each batch are replaced with their adversarial counterpart.

Warning: Both successful and unsuccessful adversarial samples are used for training. In the case of unbounded attacks (e.g., DeepFool), this can result in invalid (very noisy) samples being included.

__init__ (classifier, attacks, ratio=0.5)

Create an AdversarialTrainer instance.

Parameters

- classifier (Classifier) – Model to train adversarially.
- attacks (Attack or list(Attack)) – attacks to use for data augmentation in adversarial training
- ratio (float) – The proportion of samples in each batch to be replaced with their adversarial counterparts. Setting this value to 1 allows to train only on adversarial samples.

__weakref__

list of weak references to the object (if defined)

fit (x, y, batch_size=128, nb_epochs=20, **kwargs)

Train a model adversarially. See class documentation for more information on the exact procedure.
Parameters

- **batch_size** (*int*) – Size of batches.
- **nb_epochs** (*int*) – Number of epochs to use for trainings.
- **kwargs** (*dict*) – Dictionary of framework-specific arguments. These will be passed as such to the `fit` function of the target classifier.

Returns *None*

**fit_generator** (*generator, nb_epochs=20, **kwargs*)

Train a model adversarially using a data generator. See class documentation for more information on the exact procedure.

Parameters

- **generator** (*DataGenerator*) – Data generator.
- **nb_epochs** (*int*) – Number of epochs to use for trainings.
- **kwargs** (*dict*) – Dictionary of framework-specific arguments. These will be passed as such to the `fit` function of the target classifier.

Returns *None*

**predict**(x, **kwargs)**

Perform prediction using the adversarially trained classifier.

Parameters

- **kwargs** (*dict*) – Other parameters to be passed on to the `predict` function of the classifier.

Returns Predictions for test set.

Return type *np.ndarray*

### 1.6.5 Gaussian Data Augmentation

**class** *art.defences.GaussianAugmentation* (*sigma=1.0, augmentation=True, ratio=1.0, clip_values=None, apply_fit=True, apply_predict=False*)

Add Gaussian noise to a dataset in one of two ways: either add noise to each sample (keeping the size of the original dataset) or perform augmentation by keeping all original samples and adding noisy counterparts. When used as part of a `Classifier` instance, the defense will be applied automatically only when training if `augmentation` is true, and only when performing prediction otherwise.

**__call__**(x, y=None)

Augment the sample (x, y) with Gaussian noise. The result is either an extended dataset containing the original sample, as well as the newly created noisy samples (augmentation=True) or just the noisy counterparts to the original samples.

Parameters

- **x** (*np.ndarray*) – Sample to augment with shape (batch_size, width, height, depth).
- **y** (*np.ndarray*) – Labels for the sample. If this argument is provided, it will be augmented with the corresponded original labels of each sample point.
Returns The augmented dataset and (if provided) corresponding labels.

Return type

__init__ (sigma=1.0, augmentation=True, ratio=1.0, clip_values=None, apply_fit=True, apply_predict=False)

Initialize a Gaussian augmentation object.

Parameters

• sigma (float) – Standard deviation of Gaussian noise to be added.

• augmentation (bool) – If true, perform dataset augmentation using ratio, otherwise replace samples with noisy counterparts.

• ratio (float) – Percentage of data augmentation. E.g. for a rate of 1, the size of the dataset will double. If augmentation is false, ratio value is ignored.

• clip_values (tuple) – Tuple of the form (min, max) representing the minimum and maximum values allowed for features.

• apply_fit (bool) – True if applied during fitting/training.

• apply_predict (bool) – True if applied during predicting.

property apply_fit

Property of the defence indicating if it should be applied at training time.

Returns True if the defence should be applied when fitting a model, False otherwise.

Return type bool

property apply_predict

Property of the defence indicating if it should be applied at test time.

Returns True if the defence should be applied at prediction time, False otherwise.

Return type bool

estimate_gradient (x, grad)

Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differentiable, this is an estimate of the gradient, most often replacing the computation performed by the defence with the identity function.

Parameters

• x (np.ndarray) – Input data for which the gradient is estimated. First dimension is the batch size.

• grad (np.ndarray) – Gradient value so far.

Returns The gradient (estimate) of the defence.

Return type np.ndarray

fit (x, y=None, **kwargs)

No parameters to learn for this method; do nothing.

set_params (**kwargs)

Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

Parameters

• sigma (float) – Standard deviation of Gaussian noise to be added.

• augmentation (bool) – If true, perform dataset augmentation using ratio, otherwise replace samples with noisy counterparts.
• **ratio (float)** – Percentage of data augmentation. E.g. for a ratio of 1, the size of the dataset will double.

• **clip_values (tuple)** – Tuple of the form `(min, max)` representing the minimum and maximum values allowed for features.

### 1.6.6 PixelDefend

class art.defences.PixelDefend (clip_values=(0, 1), eps=16, pixel_cnn=None, apply_fit=False, apply_predict=True)

Implement the pixel defence approach. Defense based on PixelCNN that projects samples back to the data manifold. Paper link: https://arxiv.org/abs/1710.10766

`__call__(x, y=None)`

Apply pixel defence to sample `x`.

**Parameters**

• **x (np.ndarray)** – Sample to defense with shape `(batch_size, width, height, depth)`. `x` values are expected to be in the data range `[0, 1]`.

• **y (np.ndarray)** – Labels of the sample `x`. This function does not affect them in any way.

**Returns** Purified sample.

**Return type** np.ndarray

`__init__(clip_values=(0, 1), eps=16, pixel_cnn=None, apply_fit=False, apply_predict=True)`

Create an instance of pixel defence.

**Parameters**

• **clip_values (tuple)** – Tuple of the form `(min, max)` representing the minimum and maximum values allowed for features.

• **eps (int)** – Defense parameter 0-255.

• **pixel_cnn (Classifier)** – Pre-trained PixelCNN model.

**property apply_fit**

Property of the defence indicating if it should be applied at training time.

**Returns** *True* if the defence should be applied when fitting a model, *False* otherwise.

**Return type** bool

**property apply_predict**

Property of the defence indicating if it should be applied at test time.

**Returns** *True* if the defence should be applied at prediction time, *False* otherwise.

**Return type** bool

**estimate_gradient (x, grad)**

Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differentiable, this is an estimate of the gradient, most often replacing the computation performed by the defence with the identity function.

**Parameters**

• **x (np.ndarray)** – Input data for which the gradient is estimated. First dimension is the batch size.

• **grad (np.ndarray)** – Gradient value so far.
Returns  The gradient (estimate) of the defence.

Return type  np.ndarray

```
fit (x, y=None, **kwargs)
```

No parameters to learn for this method; do nothing.

```
set_params (**kwargs)
```

Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

Defense-specific parameters:

- **clip_values**: Tuple of the form (min, max) representing the minimum and maximum values allowed for features.

### Parameters

- **eps (int)** – Defense parameter 0-255.
- **pixel_cnn (Classifier)** – Pre-trained PixelCNN model.

#### 1.6.7 JPEG Compression

```
class art.defences.JpegCompression (clip_values, quality=50, channel_index=3, apply_fit=True, apply_predict=False)
```


```
__call__ (x, y=None)
```

Apply JPEG compression to sample x.

**Parameters**

- **x (np.ndarray)** – Sample to compress with shape (batch_size, width, height, depth). x values are expected to be in the data range [0, 1].
- **y (np.ndarray)** – Labels of the sample x. This function does not affect them in any way.

**Returns**  compressed sample.

**Return type**  np.ndarray

```
__init__ (clip_values, quality=50, channel_index=3, apply_fit=True, apply_predict=False)
```

Create an instance of jpeg compression.

**Parameters**

- **clip_values (tuple)** – Tuple of the form (min, max) representing the minimum and maximum values allowed for features.
- **quality (int)** – The image quality, on a scale from 1 (worst) to 95 (best). Values above 95 should be avoided.
- **channel_index (int)** – Index of the axis in data containing the color channels or features.
- **apply_fit (bool)** – True if applied during fitting/training.
- **apply_predict (bool)** – True if applied during predicting.

**property apply_fit**

Property of the defence indicating if it should be applied at training time.

**Returns**  True if the defence should be applied when fitting a model, False otherwise.
Return type bool

property apply_predict
    Property of the defence indicating if it should be applied at test time.

Returns True if the defence should be applied at prediction time, False otherwise.

Return type bool

estimate_gradient (x, grad)
    Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differen-
    tiable, this is an estimate of the gradient, most often replacing the computation performed by the defence
    with the identity function.

Parameters
    • x (np.ndarray) – Input data for which the gradient is estimated. First dimension is the
      batch size.
    • grad (np.ndarray) – Gradient value so far.

Returns The gradient (estimate) of the defence.

Return type np.ndarray

fit (x, y=None, **kwargs)
    No parameters to learn for this method; do nothing.

set_params (**kwargs)
    Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

Parameters
    • clip_values (tuple) – Tuple of the form (min, max) representing the minimum and
      maximum values allowed for features.
    • quality (int) – The image quality, on a scale from 1 (worst) to 95 (best). Values above
      95 should be avoided.
    • channel_index (int) – Index of the axis in data containing the color channels or fea-

1.6.8 Thermometer Encoding

class art.defences.ThermometerEncoding (clip_values, num_space=10, channel_index=3, apply_fit=True, apply_predict=True)
    Implement the thermometer encoding defence approach. Defence method from
    https://openreview.net/forum?id=S18Su–CW.

    __call__ (x, y=None)
        Apply thermometer encoding to sample x. The new axis with the encoding is added as last dimension.

Parameters
    • x (np.ndarray) – Sample to encode with shape (batch_size, width, height, depth).
    • y (np.ndarray) – Labels of the sample x. This function does not affect them in any way.

Returns Encoded sample with shape (batch_size, width, height, depth x num_space).

Return type np.ndarray

    __init__ (clip_values, num_space=10, channel_index=3, apply_fit=True, apply_predict=True)
        Create an instance of thermometer encoding.
Parameters

- **clip_values** *(tuple)* – Tuple of the form *(min, max)* representing the minimum and maximum values allowed for features.
- **num_space** *(int)* – Number of evenly spaced levels within [0, 1].
- **channel_index** *(int)* – Index of the axis in data containing the color channels or features.
- **apply_fit** *(bool)* – True if applied during fitting/training.
- **apply_predict** *(bool)* – True if applied during predicting.

**property apply_fit**

Property of the defence indicating if it should be applied at training time.

**Returns** *True* if the defence should be applied when fitting a model, *False* otherwise.

**Return type** *bool*

**property apply_predict**

Property of the defence indicating if it should be applied at test time.

**Returns** *True* if the defence should be applied at prediction time, *False* otherwise.

**Return type** *bool*

**estimate_gradient** *(x, grad)*

Provide an estimate of the gradients of the defence for the backward pass. For thermometer encoding, the gradient estimate is the one used in [https://arxiv.org/abs/1802.00420](https://arxiv.org/abs/1802.00420), where the thermometer encoding is replaced with a differentiable approximation: \( g(x_{i,j,c})_k = \min(\max(x_{i,j,c} - k / self.num_space, 0), 1) \).

**Parameters**

- **x** *(np.ndarray)* – Input data for which the gradient is estimated. First dimension is the batch size.
- **grad** *(np.ndarray)* – Gradient value so far.

**Returns** The gradient (estimate) of the defence.

**Return type** *np.ndarray*

**fit** *(x, y=None, **kwargs)*

No parameters to learn for this method; do nothing.

**set_params** (**kwargs)**

Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

**Parameters**

- **clip_values** *(tuple)* – Tuple of the form *(min, max)* representing the minimum and maximum values allowed for features.
- **num_space** *(int)* – Number of evenly spaced levels within [0, 1].
- **channel_index** *(int)* – Index of the axis in data containing the color channels or features.
1.6.9 Total Variance Minimization

```python
class art.defences.TotalVarMin(prob=0.3, norm=2, lamb=0.5, solver='L-BFGS-B',
                               max_iter=10, clip_values=None, apply_fit=False, apply_predict=True)
```

Implement the total variance minimization defence approach. Defence method from [Guo et al., 2018]. Paper link: https://openreview.net/forum?id=SyJ7CIWCb

___call__(x, y=None)

Apply total variance minimization to sample `x`.

Parameters
- `x` (np.ndarray) – Sample to compress with shape (batch_size, width, height, depth).
- `y` (np.ndarray) – Labels of the sample `x`. This function does not affect them in any way.

Returns
Similar samples.

Return type
np.ndarray

___init__ (prob=0.3, norm=2, lamb=0.5, solver='L-BFGS-B', max_iter=10, clip_values=None, apply_fit=False, apply_predict=True)

Create an instance of total variance minimization.

Parameters
- `prob` (float) – Probability of the Bernoulli distribution.
- `norm` (int) – The norm (positive integer).
- `lamb` (float) – The lambda parameter in the objective function.
- `solver` (str) – Current support: L-BFGS-B, CG, Newton-CG.
- `max_iter` (int) – Maximum number of iterations when performing optimization.
- `clip_values` (tuple) – Tuple of the form (min, max) representing the minimum and maximum values allowed for features.
- `apply_fit` (bool) – True if applied during fitting/training.
- `apply_predict` (bool) – True if applied during predicting.

property apply_fit

Property of the defence indicating if it should be applied at training time.

Returns
`True` if the defence should be applied when fitting a model, `False` otherwise.

Return type
bool

property apply_predict

Property of the defence indicating if it should be applied at test time.

Returns
`True` if the defence should be applied at prediction time, `False` otherwise.

Return type
bool

estimate_gradient (x, grad)

Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differentiable, this is an estimate of the gradient, most often replacing the computation performed by the defence with the identity function.

Parameters
- `x` (np.ndarray) – Input data for which the gradient is estimated. First dimension is the batch size.
• `grad` *(np.ndarray)* – Gradient value so far.

**Returns** The gradient (estimate) of the defence.

**Return type** *np.ndarray*

`.fit`(x, y=None, **kwargs)

No parameters to learn for this method; do nothing.

`.set_params`(**kwargs)

Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes.

**Parameters**

• `prob` *(float)* – Probability of the Bernoulli distribution.

• `norm` *(int)* – The norm (positive integer).

• `lamb` *(float)* – The lambda parameter in the objective function.

• `solver` *(str)* – Current support: *L-BFGS-B, CG, Newton-CG*.

• `max_iter` *(int)* – Maximum number of iterations when performing optimization.

• `clip_values` *(tuple)* – Tuple of the form *(min, max)* representing the minimum and maximum values allowed for features.

### 1.6.10 Base Preprocessor Class

`class` *art.defences.Preprocessor*

Abstract base class for defences performing model hardening by preprocessing data.

`.abstract __call__`(x, y=None)

Perform data preprocessing and return preprocessed data as tuple.

**Parameters**

• `x` *(np.ndarray)* – Dataset to be preprocessed.

• `y` *(np.ndarray)* – Labels to be preprocessed.

**Returns** Preprocessed data

`.abstract __init__`()

Create a preprocessing object

`.abstract property apply_fit`

Property of the defence indicating if it should be applied at training time.

**Returns** *True* if the defence should be applied when fitting a model, *False* otherwise.

**Return type** *bool*

`.abstract property apply_predict`

Property of the defence indicating if it should be applied at test time.

**Returns** *True* if the defence should be applied at prediction time, *False* otherwise.

**Return type** *bool*

`.abstract estimate_gradient`(x, grad)

Provide an estimate of the gradients of the defence for the backward pass. If the defence is not differentiable, this is an estimate of the gradient, most often replacing the computation performed by the defence with the identity function.

**Parameters**
• **x** (*np.ndarray*) – Input data for which the gradient is estimated. First dimension is the batch size.

• **grad** (*np.ndarray*) – Gradient value so far.

**Returns**
The gradient (estimate) of the defence.

**Return type** *np.ndarray*

**abstract fit**(*x*, *y=None, **kwargs*)
Fit the parameters of the data preprocessor if it has any.

**Parameters**

• **x** (*np.ndarray*) – Training set to fit the preprocessor.

• **y** (*np.ndarray*) – Labels for the training set.

• **kwargs** (*dict*) – Other parameters.

**Returns**
None

**property is_fitted**
Return the state of the preprocessing object.

**Returns**
*True* if the preprocessing model has been fitted (if this applies).

**Return type** *bool*

**set_params**(**kwargs**)
Take in a dictionary of parameters and apply checks before saving them as attributes.

**Returns**
*True* when parsing was successful

### 1.7 art.detection

Module providing methods for detecting adversarial samples under a common interface.

#### 1.7.1 Binary Input Detector

**class art.detection.BinaryInputDetector**(detector)
Binary detector of adversarial samples coming from evasion attacks. The detector uses an architecture provided by the user and trains it on data labeled as clean (label 0) or adversarial (label 1).

**property channel_index**
Returns Index of the axis in data containing the color channels or features.

**Return type** *int*

**class_gradient**(*x*, *label=None, logits=False, **kwargs*)
Compute per-class derivatives w.r.t. *x*.

**Parameters**

• **x** (*np.ndarray*) – Sample input with shape as expected by the model.

• **label** (*int or list*) – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of *x*, and each value will be used as target for its corresponding sample in *x*. If *None*, then gradients for all classes will be computed for each sample.
• logits (bool) – True if the prediction should be done at the logits layer.

Returns  Array of gradients of input features w.r.t. each class in the form \((\text{batch} \_\text{size}, \text{nb} \_\text{classes}, \text{input} \_\text{shape})\) when computing for all classes, otherwise shape becomes \((\text{batch} \_\text{size}, 1, \text{input} \_\text{shape})\) when label parameter is specified.

Return type np.ndarray

property clip_values

Returns  Tuple of the form \((\text{min}, \text{max})\) representing the minimum and maximum values allowed for features.

Return type tuple

fit \((x, y, \text{batch} \_\text{size}=128, \text{nb} \_\text{epochs}=20, **\text{kwargs})\)

Fit the detector using clean and adversarial samples.

Parameters

• x (np.ndarray) – Training set to fit the detector.
• y (np.ndarray) – Labels for the training set.
• batch_size (int) – Size of batches.
• nb_epochs (int) – Number of epochs to use for training.
• kwargs (dict) – Other parameters.

Returns  None

fit_generator \((\text{generator}, \text{nb} \_\text{epochs}=20, **\text{kwargs})\)

Fit the classifier using the generator gen that yields batches as specified. This function is not supported for this detector.

Raises  NotImplementedError

get_activations \((x, \text{layer}, \text{batch} \_\text{size})\)

Return the output of the specified layer for input \(x\). \text{layer} is specified by layer index (between 0 and \(\text{nb} \_\text{layers} - 1\)) or by name. The number of layers can be determined by counting the results returned by calling \text{layer} \_\text{names}. This function is not supported for this detector.

Raises  NotImplementedError

property input_shape

Return the shape of one input.

Returns  Shape of one input for the classifier.

Return type tuple

learning_phase()

Return the learning phase set by the user for the current classifier. Possible values are True for training, False for prediction and None if it has not been set through the library. In the latter case, the library does not do any explicit learning phase manipulation and the current value of the backend framework is used. If a value has been set by the user for this property, it will impact all following computations for model fitting, prediction and gradients.

Returns  Value of the learning phase.

Return type bool or None

loss_gradient \((x, y, **\text{kwargs})\)

Compute the gradient of the loss function w.r.t. \(x\).
Parameters

- \( x (\text{np.ndarray}) \) – Sample input with shape as expected by the model.
- \( y (\text{np.ndarray}) \) – Correct labels, one-vs-rest encoding.

Returns Array of gradients of the same shape as \( x \).

Return type np.ndarray

property nb_classes

Return the number of output classes.

Returns Number of classes in the data.

Return type int

predict \((x, \text{logits=False, batch_size=128, **kwargs})\)

Perform detection of adversarial data and return prediction as tuple.

Parameters

- \( x (\text{np.ndarray}) \) – Data sample on which to perform detection.
- \( \text{logits} (\text{bool}) \) – True if the prediction should be done at the logits layer.
- \( \text{batch_size} (\text{int}) \) – Size of batches.

Returns Per-sample prediction whether data is adversarial or not, where 0 means non-adversarial. Return variable has the same batch_size (first dimension) as \( x \).

Return type np.ndarray

save \((\text{filename, path=None})\)

Save a model to file in the format specific to the backend framework.

Parameters

- \( \text{filename} (\text{str}) \) – Name of the file where to store the model.
- \( \text{path} (\text{str}) \) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library DATA_PATH.

Returns None

set_learning_phase \((\text{train})\)

Set the learning phase for the backend framework.

Parameters \( \text{train} (\text{bool}) \) – True to set the learning phase to training, False to set it to prediction.

1.7.2 Binary Activation Detector

class art.detection.BinaryActivationDetector \((\text{classifier, detector, layer})\)

Binary detector of adversarial samples coming from evasion attacks. The detector uses an architecture provided by the user and is trained on the values of the activations of a classifier at a given layer.

property channel_index

Returns Index of the axis in data containing the color channels or features.

Return type int

class_gradient \((x, label=None, logits=False, **kwargs)\)

Compute per-class derivatives w.r.t. \( x \).
Parameters

- **x** (np.ndarray) – Sample input with shape as expected by the model.
- **label** (int or list) – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of x, and each value will be used as target for its corresponding sample in x. If None, then gradients for all classes will be computed for each sample.
- **logits** (bool) – True if the prediction should be done at the logits layer.

Returns

Array of gradients of input features w.r.t. each class in the form (batch_size, nb_classes, input_shape) when computing for all classes, otherwise shape becomes (batch_size, 1, input_shape) when label parameter is specified.

Return type np.ndarray

property clip_values

Returns Tuple of the form (min, max) representing the minimum and maximum values allowed for features.

Return type tuple

fit (x, y, batch_size=128, nb_epochs=20, **kwargs)

Fit the detector using training data.

Parameters

- **x** (np.ndarray) – Training set to fit the detector.
- **y** (np.ndarray) – Labels for the training set.
- **batch_size** (int) – Size of batches.
- **nb_epochs** (int) – Number of epochs to use for training.
- **kwargs** (dict) – Other parameters.

Returns None

fit_generator (generator, nb_epochs=20, **kwargs)

Fit the classifier using the generator gen that yields batches as specified. This function is not supported for this detector.

Raises NotImplementedException

get_activations (x, layer, batch_size)

Return the output of the specified layer for input x. layer is specified by layer index (between 0 and nb_layers - 1) or by name. The number of layers can be determined by counting the results returned by calling layer_names. This function is not supported for this detector.

Raises NotImplementedException

property input_shape

Return the shape of one input.

Returns Shape of one input for the classifier.

Return type tuple

learning_phase()

Return the learning phase set by the user for the current classifier. Possible values are True for training, False for prediction and None if it has not been set through the library. In the latter case, the library does not do any explicit learning phase manipulation and the current value of the backend framework is used.
If a value has been set by the user for this property, it will impact all following computations for model fitting, prediction and gradients.

**Returns** Value of the learning phase.

**Return type** bool or None

### loss_gradient (x, y, **kwargs)

Compute the gradient of the loss function w.r.t. \( x \).

**Parameters**

- \( x \) (np.ndarray) – Sample input with shape as expected by the model.
- \( y \) (np.ndarray) – Correct labels, one-vs-rest encoding.

**Returns** Array of gradients of the same shape as \( x \).

**Return type** np.ndarray

### property nb_classes

Return the number of output classes.

**Returns** Number of classes in the data.

**Return type** int

### predict (x, logits=False, batch_size=128, **kwargs)

Perform detection of adversarial data and return prediction as tuple.

**Parameters**

- \( x \) (np.ndarray) – Data sample on which to perform detection.
- \( \text{logits} \) (bool) – True if the prediction should be done at the logits layer.
- \( \text{batch\_size} \) (int) – Size of batches.

**Returns** Per-sample prediction whether data is adversarial or not, where 0 means non-adversarial. Return variable has the same \( \text{batch\_size} \) (first dimension) as \( x \).

**Return type** np.ndarray

### save (filename, path=None)

Save a model to file in the format specific to the backend framework.

**Parameters**

- \( \text{filename} \) (str) – Name of the file where to store the model.
- \( \text{path} \) (str) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library \( DATA\_PATH \).

**Returns** None

### set_learning_phase (train)

Set the learning phase for the backend framework.

**Parameters** \( \text{train} \) (bool) – True to set the learning phase to training, False to set it to prediction.
1.8 art.detection.subsetscanning

1.8.1 Subset Scanning Detector

class art.detection.subsetscanning.SubsetScanningDetector (classifier, bgd_data, layer)

    Fast generalized subset scan based detector

    calculate_pvalue_ranges (eval_x)
    Returns computed p-value ranges.

        Parameters eval_x (np.ndarray) – data being evaluated for anomalies
        Returns p-value ranges ndarray
        Return type np.ndarray

    property channel_index
        Returns Index of the axis in data containing the color channels or features.

        Return type int

    class_gradient (x, label=None, logits=False, **kwargs)
    Compute per-class derivatives w.r.t. x.

        Parameters

            • x (np.ndarray) – Sample input with shape as expected by the model.

            • label (int or list) – Index of a specific per-class derivative. If an integer is provided, the
              gradient of that class output is computed for all samples. If multiple values are provided,
              the first dimension should match the batch size of x, and each value will be used as target
              for its corresponding sample in x. If None, then gradients for all classes will be computed
              for each sample.

            • logits (bool) – True if the prediction should be done at the logits layer.

        Returns Array of gradients of input features w.r.t. each class in the form (batch_size, nb_classes, input_shape) when computing for all classes, otherwise shape becomes (batch_size, 1, input_shape) when label parameter is specified.

        Return type np.ndarray

    property clip_values
        Returns Tuple of the form (min, max) representing the minimum and maximum values allowed
        for features.

        Return type tuple

    fit (x, y, batch_size=128, nb_epochs=20, **kwargs)
    Fit the detector using training data. Assume that the classifier is already trained

        Raises NotImplementedException

    fit_generator (generator, nb_epochs=20, **kwargs)
    Fit the classifier using the generator gen that yields batches as specified. This function is not supported for this detector.

        Raises NotImplementedException
get_activations \((x, \text{layer, batch\_size})\)
Return the output of the specified layer for input \(x\). \text{layer} is specified by layer index (between 0 and \(nb\_layers - 1\)) or by name. The number of layers can be determined by counting the results returned by calling \text{layer\_names}. This function is not supported for this detector.

Raises \text{NotImplementedException}

property input\_shape
Return the shape of one input.

Returns Shape of one input for the classifier.

Return type tuple

learning\_phase()
Return the learning phase set by the user for the current classifier. Possible values are \text{True} for training, \text{False} for prediction and \text{None} if it has not been set through the library. In the latter case, the library does not do any explicit learning phase manipulation and the current value of the backend framework is used. If a value has been set by the user for this property, it will impact all following computations for model fitting, prediction and gradients.

Returns Value of the learning phase.

Return type bool or None

loss\_gradient \((x, y, **kwargs)\)
Compute the gradient of the loss function w.r.t. \(x\).

Parameters
- \(x\) (np.ndarray) – Sample input with shape as expected by the model.
- \(y\) (np.ndarray) – Correct labels, one-vs-rest encoding.

Returns Array of gradients of the same shape as \(x\).

Return type np.ndarray

property nb\_classes
Return the number of output classes.

Returns Number of classes in the data.

Return type int

predict \((x, logits=\text{False}, batch\_size=128, **kwargs)\)
Perform detection of adversarial data and return prediction as tuple.

Raises \text{NotImplementedException}

save (filename, path=\text{None})
Save a model to file in the format specific to the backend framework.

Parameters
- \text{filename} (str) – Name of the file where to store the model.
- \text{path} (str) – Path of the folder where to store the model. If no path is specified, the model will be stored in the default data location of the library \text{DATA\_PATH}.

Returns None

scan \((clean\_x, adv\_x, cleansize=\text{None}, adv\_size=\text{None}, run=10)\)
Returns scores of highest scoring subsets

Parameters \text{clean\_x} – data presumably without anomalies
1.9 art.poison_detection

Poison detection defence API. Use the PoisonFilteringDefence wrapper to be able to apply a defence for a preexisting model.

1.9.1 Activation Defence

class art.poison_detection.ActivationDefence(classifier, x_train, y_train)

    analyze_clusters(**kwargs)
        This function analyzes the clusters according to the provided method.

            Parameters kwargs (dict) – A dictionary of cluster-analysis-specific parameters.

            Returns (report, assigned_clean_by_class), where the report is a dict object and assigned_clean_by_class is an array of arrays that contains what data points where classified as clean.

            Return type tuple(dict, np.ndarray)

    cluster_activations(**kwargs)
        Clusters activations and returns cluster_by_class and red_activations_by_class, where cluster_by_class[i][j] is the cluster to which the j-th datapoint in the i-th class belongs and the correspondent activations reduced by class red_activations_by_class[i][j].

            Parameters kwargs (dict) – A dictionary of cluster-specific parameters.

            Returns Clusters per class and activations by class.

            Return type tuple

    detect_poison(**kwargs)
        Returns poison detected and a report.

            Parameters kwargs (dict) – A dictionary of detection-specific parameters.

            Returns (report, is_clean_lst): where a report is a dict object that contains information specified by the clustering analysis technique where is_clean is a list, where is_clean_lst[i]=1 means that x_train[i] there is clean and is_clean_lst[i]=0, means that x_train[i] was classified as poison.

            Return type tuple

    evaluate_defence(is_clean, **kwargs)
        Returns confusion matrix.

            Parameters
• **is_clean** (:class `np.ndarray`) – Ground truth, where is_clean[i]=1 means that x_train[i] is clean and is_clean[i]=0 means x_train[i] is poisonous.

**Returns** JSON object with confusion matrix.

**Return type** jsonObject

`plot_clusters` (**save=True, folder='.'**, **kwargs)

Creates a 3D-plot to visualize each cluster each cluster is assigned a different color in the plot. When save=True, it also stores the 3D-plot per cluster in DATA_PATH.

**Parameters**

- **save** (bool) – Boolean specifying if image should be saved
- **folder** (str) – Directory where the sprites will be saved inside DATA_PATH folder
- **kwargs** (dict) – A dictionary of cluster-analysis-specific parameters

**Returns** None

`static relabel_poison_cross_validation` (**classifier**, **x**, **y_fix**, **n_splits=10**, **tolerable_backdoor=0.01**, **max_epochs=50**, **batch_epochs=10**)

Revert poison attack by continue training the current classifier with x, y_fix. n_splits determine the number of cross validation splits.

**Parameters**

- **classifier** (:class `Classifier`) – Classifier to be fixed
- **x** (:class `np.ndarray`) – Samples that were mislabeled.
- **y_fix** (:class `np.ndarray`) – True label of x.
- **n_splits** (int) – Determines how many splits to use in cross validation (only used if cross_validation=True).
- **tolerable_backdoor** (float) – Threshold that determines what is the maximum tolerable backdoor success rate.
- **max_epochs** (int) – Maximum number of epochs that the model will be trained.
- **batch_epochs** (int) – Number of epochs to be trained before checking current state of model.

**Returns** (improve_factor, classifier)

**Return type** float, Classifier

`static relabel_poison_ground_truth` (**classifier**, **x**, **y_fix**, **test_set_split=0.7**, **tolerable_backdoor=0.01**, **max_epochs=50**, **batch_epochs=10**)

Revert poison attack by continue training the current classifier with x, y_fix. test_set_split determines the percentage in x that will be used as training set, while 1-test_set_split determines how many data points to use for test set.

**Parameters**

- **classifier** (:class `Classifier`) – Classifier to be fixed
- **x** (:class `np.ndarray`) – Samples
- **y_fix** (:class `np.ndarray`) – True label of x_poison
• **test_set_split** – this parameter determines how much data goes to the training set. Here \( \text{test_set_split} \times \text{len(y_fix)} \) determines the number of data points in \( x_{\text{train}} \) and \( (1 - \text{test_set_split}) \times \text{len(y_fix)} \) the number of data points in \( x_{\text{test}} \).

• **tolerable_backdoor** (float) – Threshold that determines what is the maximum tolerable backdoor success rate.

• **max_epochs** (int) – Maximum number of epochs that the model will be trained

• **batch_epochs** (int) – Number of epochs to be trained before checking current state of model

**Returns** (improve_factor, classifier)

**Return type** float, Classifier

**set_params** (**kwargs**)

Take in a dictionary of parameters and applies defence-specific checks before saving them as attributes. If a parameter is not provided, it takes its default value.

**Parameters**

• **nb_clusters** (int) – Number of clusters to be produced. Should be greater than 2.

• **clustering_method** (str) – Clustering method to use

• **nbDims** (int) – Number of dimensions to project on

• **reduce** (str) – Reduction technique

• **cluster_analysis** (str) – Method to analyze the clusters

**visualize_clusters** (**kwargs**)  

This function creates the sprite/mosaic visualization for clusters. When save=True, it also stores a sprite (mosaic) per cluster in DATA_PATH.

**Parameters**

• **x_raw** (np.darray) – Images used to train the classifier (before pre-processing)

• **save** (bool) – Boolean specifying if image should be saved

• **folder** (str) – Directory where the sprites will be saved inside DATA_PATH folder

• **kwargs** (dict) – a dictionary of cluster-analysis-specific parameters

**Returns** sprites_by_class: Array with sprite images sprites_by_class, where sprites_by_class[i][j] contains the sprite of class i cluster j.

**Return type** sprites_by_class: np.ndarray

### 1.9.2 Base Class

**class** art.poison_detection.PoisonFilteringDefence(classifier, x_train, y_train)**

Base class for all poison filtering defences.

**abstract** detect_poison (**kwargs**)**

Detect poison.

**Parameters** **kwargs** (dict) – Defence-specific parameters used by child classes.

**Returns** (dict, list) dictionary with report and list with items identified as poison

**abstract** evaluate_defence (is_clean, **kwargs**)**

Evaluate the defence given the labels specifying if the data is poisoned or not.
Parameters

• **is_clean** – 1-D array where is_clean[i]=1 means x_train[i] is clean and is_clean[i]=0 that it’s poison.

• **kwargs** (*dict*) – Defence-specific parameters used by child classes.

Returns JSON object with confusion matrix

get_params()
Returns dictionary of parameters used to run defence.

Returns *dict*

set_params(**kwargs)
Take in a dictionary of parameters and apply attack-specific checks before saving them as attributes.

Parameters **kwargs** (*dict*) – a dictionary of defence-specific parameters

Returns True when parsing was successful

1.10 art.metrics

Module implementing varying metrics for assessing model robustness. These fall mainly under two categories: attack-dependent and attack-independent.

1.10.1 Loss Sensitivity

art.metrics.loss_sensitivity(classifier, x, y)

Local loss sensitivity estimated through the gradients of the prediction at points in \( x \), as defined in https://arxiv.org/pdf/1706.05394.pdf.

Parameters

• **classifier** (*Classifier*) – A trained model

• **x** (*np.ndarray*) – Data sample of shape that can be fed into classifier

• **y** (*np.ndarray*) – Labels for sample \( x \), one-hot encoded.

Returns The average loss sensitivity of the model

Return type *float*

1.10.2 Empirical Robustness

art.metrics.empirical_robustness(classifier, x, attack_name, attack_params=None)

Compute the Empirical Robustness of a classifier object over the sample \( x \) for a given adversarial crafting method **attack**. This is equivalent to computing the minimal perturbation that the attacker must introduce for a successful attack. Paper link: https://arxiv.org/abs/1511.04599

Parameters

• **classifier** (*Classifier*) – A trained model

• **x** (*np.ndarray*) – Data sample of shape that can be fed into classifier

• **attack_name** (*str*) – adversarial attack name

• **attack_params** (*dict*) – Parameters specific to the adversarial attack
Returns The average empirical robustness computed on $x$

Return type float

1.10.3 CLEVER

art.metrics.clever_u(classifier, x, nb_batches, batch_size, radius, norm, c_init=1, pool_factor=10)
Compute CLEVER score for an untargeted attack. Paper link: https://arxiv.org/abs/1801.10578

Parameters

- **classifier** (*Classifier*) – A trained model.
- **x** (*np.ndarray*) – One input sample
- **nb_batches** (*int*) – Number of repetitions of the estimate
- **batch_size** (*int*) – Number of random examples to sample per batch
- **radius** (*float*) – Radius of the maximum perturbation
- **norm** (*int*) – Current support: 1, 2, np.inf
- **c_init** (*float*) – Initialization of Weibull distribution
- **pool_factor** (*int*) – The factor to create a pool of random samples with size pool_factor

Returns CLEVER score

Return type float

art.metrics.clever_t(classifier, x, target_class, nb_batches, batch_size, radius, norm, c_init=1, pool_factor=10)
Compute CLEVER score for a targeted attack. Paper link: https://arxiv.org/abs/1801.10578

Parameters

- **classifier** (*Classifier*) – A trained model
- **x** (*np.ndarray*) – One input sample
- **target_class** (*int*) – Targeted class
- **nb_batches** (*int*) – Number of repetitions of the estimate
- **batch_size** (*int*) – Number of random examples to sample per batch
- **radius** (*float*) – Radius of the maximum perturbation
- **norm** (*int*) – Current support: 1, 2, np.inf
- **c_init** (*float*) – Initialization of Weibull distribution
- **pool_factor** (*int*) – The factor to create a pool of random samples with size pool_factor

Returns CLEVER score

Return type float

1.11 art.utils

Module providing convenience functions.
1.11.1 Load Datasets

art.utils.load_dataset(name)

Loads or downloads the dataset corresponding to name. Options are: mnist, cifar10 and stl10.

Parameters
name (str) – Name of the dataset

Returns
The dataset separated in training and test sets as (x_train, y_train), (x_test, y_test), min, max

Return type
(np.ndarray, np.ndarray), (np.ndarray, np.ndarray), float, float

Raises
NotImplementedError – If the dataset is unknown.

1.11.2 Maths Operations

art.utils.original_to_tanh(x_original, clip_min, clip_max, tanh_smother=0.999999)

Transform input from original to tanh space.

Parameters
• x_original (np.ndarray) – An array with the input to be transformed.
• clip_min (float or np.ndarray) – Minimum clipping value.
• clip_max (float or np.ndarray) – Maximum clipping value.
• tanh_smother (float) – Scalar for multiplying arguments of arctanh to avoid division by zero.

Returns
An array holding the transformed input.

Return type
np.ndarray

art.utils.projection(values, eps, norm_p)

Project values on the L_p norm ball of size eps.

Parameters
• values (np.ndarray) – Array of perturbations to clip.
• eps (float) – Maximum norm allowed.
• norm_p (int) – L_p norm to use for clipping. Only 1, 2 and np.Inf supported for now.

Returns
Values of values after projection.

Return type
np.ndarray

art.utils.random_sphere(nb_points, nb_dims, radius, norm)

Generate randomly m x n-dimension points with radius radius and centered around 0.

Parameters
• nb_points (int) – Number of random data points
• nb_dims (int) – Dimensionality
• radius (float) – Radius
• norm (int) – Current support: 1, 2, np.inf

Returns
The generated random sphere

Return type
np.ndarray
art.utils.tanh_to_original(x_tanh, clip_min, clip_max, tanh_smooother=0.999999)
Transform input from tanh to original space.

Parameters
• x_tanh (np.ndarray) – An array with the input to be transformed.
• clip_min (float or np.ndarray) – Minimum clipping value.
• clip_max (float or np.ndarray) – Maximum clipping value.
• tanh_smooother (float) – Scalar for dividing arguments of tanh to avoid division by zero.

Returns An array holding the transformed input.
Return type np.ndarray

1.11.3 Preprocess Labels
art.utils.least_likely_class(x, classifier)
Compute the least likely class predictions for sample x. This strategy for choosing attack targets was used in

Parameters
• x (np.ndarray) – A data sample of shape accepted by classifier.
• classifier (Classifier) – The classifier used for computing predictions.

Returns Least-likely class predicted by classifier for sample x in one-hot encoding.
Return type np.ndarray

art.utils.random_targets(labels, nb_classes)
Given a set of correct labels, randomly choose target labels different from the original ones. These can be
one-hot encoded or integers.

Parameters
• labels (np.ndarray) – The correct labels
• nb_classes (int) – The number of classes for this model

Returns An array holding the randomly-selected target classes, one-hot encoded.
Return type np.ndarray

art.utils.second_most_likely_class(x, classifier)
Compute the second most likely class predictions for sample x. This strategy can be used for choosing target
labels for an attack to improve its chances to succeed.

Parameters
• x (np.ndarray) – A data sample of shape accepted by classifier.
• classifier (Classifier) – The classifier used for computing predictions.

Returns Second most likely class predicted by classifier for sample x in one-hot encoding.
Return type np.ndarray

art.utils.to_categorical(labels, nb_classes=None)
Convert an array of labels to binary class matrix.

Parameters
• labels (np.ndarray) – An array of integer labels of shape (nb_samples,)
• **nb_classes** (*int*) – The number of classes (possible labels)

**Returns** A binary matrix representation of y in the shape *(nb_samples, nb_classes)*

**Return type** *np.ndarray*

### 1.11.4 Preprocess Datasets

**art.utils.preprocess** *(x, y, nb_classes=10, clip_values=None)*

Scales x to [0, 1] and converts y to class categorical confidences.

**Parameters**

- *nb_classes* (*int*) – Number of classes in dataset.
- *clip_values* (*tuple(float, float) or tuple(np.ndarray, np.ndarray]*) – Original data range allowed value for features, either one respective scalar or one value per feature.

**Returns** Rescaled values of x, y

**Return type** *tuple*

### 1.11.5 Fix the Seed for Random Number Generators

**art.utils.master_seed** *(seed)*

Set the seed for all random number generators used in the library. This ensures experiments reproducibility and stable testing.

**Parameters** *seed* (*int*) – The value to be seeded in the random number generators.

### 1.12 art.utils_test

Module providing convenience functions specifically for unit tests.

#### 1.12.1 Unit Test Pretrained Models

**art.utils_test.get_classifier_kr()**

Standard Keras classifier for unit testing

The weights and biases are identical to the Tensorflow model in get_classifier_tf().

**Returns** KerasClassifier, tf.Session()

**art.utils_test.get_classifier_pt()**

Standard PyTorch classifier for unit testing

**Returns** PyTorchClassifier

**art.utils_test.get_classifier_tf()**

Standard Tensorflow classifier for unit testing.

The following hyper-parameters were used to obtain the weights and biases: learning_rate: 0.01 batch size: 10 number of epochs: 2 optimizer: tf.train.AdamOptimizer
Returns  TFClassifier, tf.Session()

1.13 art.wrappers

Module providing wrappers for Classifier instances to simulate different capacities and behaviours, like black-box gradient estimation.

1.13.1 Expectation over Transformations

class art.wrappers.ExpectationOverTransformations (classifier, sample_size, transformation)


class_gradient (x, label=None, logits=False)

Compute per-class derivatives of the given classifier w.r.t. x, taking an expectation over transformations.

Parameters

• x (np.ndarray) – Sample input with shape as expected by the model.

• label (int or list) – Index of a specific per-class derivative. If an integer is provided, the gradient of that class output is computed for all samples. If multiple values as provided, the first dimension should match the batch size of x, and each value will be used as target for its corresponding sample in x. If None, then gradients for all classes will be computed for each sample.

• logits (bool) – True if the prediction should be done at the logits layer.

Returns  Array of gradients of input features w.r.t. each class in the form (batch_size, nb_classes, input_shape) when computing for all classes, otherwise shape becomes (batch_size, 1, input_shape) when label parameter is specified.

Return type  np.ndarray

loss_gradient (x, y)

Compute the gradient of the given classifier’s loss function w.r.t. x, taking an expectation over transformations.

Parameters

• x (np.ndarray) – Sample input with shape as expected by the model.

• y (np.ndarray) – Correct labels, one-hot encoded.

Returns  Array of gradients of the same shape as x.

Return type  np.ndarray

predict (x, logits=False, batch_size=128)

Perform prediction of the given classifier for a batch of inputs, taking an expectation over transformations.

Parameters

• x (np.ndarray) – Test set.

• logits (bool) – True if the prediction should be done at the logits layer.

• batch_size (int) – Size of batches.

Returns  Array of predictions of shape (nb_inputs, self.nb_classes).
1.13.2 Query-Efficient Black-Box Attack

class art.wrappers.QueryEfficientBBGradientEstimation(classifier, num_basis, sigma, round_samples=0)

Implementation of Query-Efficient Black-box Adversarial Examples. The attack approximates the gradient by maximizing the loss function over samples drawn from random Gaussian noise around the input.


loss_gradient(x, y)

Compute the gradient of the loss function w.r.t. x.

Parameters

• x (np.ndarray) – Sample input with shape as expected by the model.

• y (np.ndarray) – Correct labels, one-vs-rest encoding.

Returns Array of gradients of the same shape as x.

Return type np.ndarray

1.13.3 Base Wrapper

class art.wrappers.ClassifierWrapper(classifier)

Wrapper class for any classifier instance.

set_params(**kwargs)

Take in a dictionary of parameters and pass them down to the underlying wrapped classifier instance.

Parameters kwargs (dict) – A dictionary of attack-specific parameters.

Returns True when parsing was successful.
INDICES AND TABLES

- genindex
- modindex
- search
a
art.attacks, 5
art.classifiers, 21
art.data_generators, 38
art.defences, 39
art.detection, 52
art.detection.subsetscanning, 57
art.metrics, 62
art.poison_detection, 59
art.utils, 63
art.utils_test, 66
art.wrappers, 67
Symbols

__call__() (art.defences.FeatureSqueezing method), 39
__call__() (art.defences.GaussianAugmentation method), 44
__call__() (art.defences.JpegCompression method), 47
__call__() (art.defences.LabelSmoothing method), 42
__call__() (art.defences.PixelDefend method), 46
__call__() (art.defences.Preprocessor method), 51
__call__() (art.defences.SpatialSmoothing method), 41
__call__() (art.defences.ThermometerEncoding method), 48
__call__() (art.defences.TotalVarMin method), 50
__getstate__() (art.classifiers.KerasClassifier method), 21
__getstate__() (art.classifiers.PyTorchClassifier method), 27
__getstate__() (art.classifiers.TFClassifier method), 30
__init__() (art.attacks.AdversarialPatch method), 5
__init__() (art.attacks.BasicIterativeMethod method), 13
__init__() (art.attacks.BoundaryAttack method), 6
__init__() (art.attacks.CarliniL2Method method), 7
__init__() (art.attacks.CarliniLInfMethod method), 9
__init__() (art.attacks.DeepFool method), 10
__init__() (art.attacks.ElasticNet method), 11
__init__() (art.attacks.FastGradientMethod method), 12
__init__() (art.attacks.HopSkipJump method), 20
__init__() (art.attacks.NewtonFool method), 15
__init__() (art.attacksProjectedGradientDescent method), 13
__init__() (art.attacks.SaliencyMapMethod method), 14
__init__() (art.attacks.SpatialTransformation method), 16
__init__() (art.attacks.UniversalPerturbation method), 17
__init__() (art.attacks.VirtualAdversarialMethod method), 17
__init__() (art.attacks.ZooAttack method), 18
__init__() (art.classifiers.EnsembleClassifier method), 33
__init__() (art.classifiers.KerasClassifier method), 22
__init__() (art.classifiers.MXClassifier method), 24
__init__() (art.classifiers.PyTorchClassifier method), 27
__init__() (art.classifiers.TFClassifier method), 30
__init__() (art.defences.AdversarialTrainer method), 43
__init__() (art.defences.FeatureSqueezing method), 40
__init__() (art.defences.GaussianAugmentation method), 45
__init__() (art.defences.JpegCompression method), 47
__init__() (art.defences.LabelSmoothing method), 42
__init__() (art.defences.PixelDefend method), 46
__init__() (art.defences.Preprocessor method), 51
__init__() (art.defences.SpatialSmoothing method), 41
__init__() (art.defences.ThermometerEncoding method), 48
__init__() (art.defences.TotalVarMin method), 50
__repr__() (art.classifiers.EnsembleClassifier method), 33
__repr__() (art.classifiers.KerasClassifier method), 22
__repr__() (art.classifiers.MXClassifier method), 25
__repr__() (art.classifiers.PyTorchClassifier method), 28
__repr__() (art.classifiers.TFClassifier method), 31
__setstate__() (art.classifiers.KerasClassifier method), 22
__setstate__() (art.classifiers.PyTorchClassifier method), 28
__setstate__() (art.classifiers.TFClassifier method), 31
clip_values() (art.detection.BinaryActivationDetector method), 55
clip_values() (art.detection.BinaryInputDetector method), 53
clip_values() (art.detection.subsetscanning.SubsetScanningDetector method), 57
cluster_activations() (art.poison_detection.ActivationDefence method), 59

D
DataGenerator (class in art.data_generators), 38
DeepFool (class in art.attacks), 10
detect_poison() (art.poison_detection.ActivationDefence method), 59
detect_poison() (art.poison_detection.PoisonFilteringDefence method), 61

E
ElasticNet (class in art.attacks), 11
empirical_robustness() (in module art.metrics), 62
EnsembleClassifier (class in art.classifiers), 33
estimate_gradient() (art.defences.FeatureSqueezing method), 40
estimate_gradient() (art.defences.GaussianAugmentation method), 45
estimate_gradient() (art.defences.JpegCompression method), 48
estimate_gradient() (art.defences.LabelSmoothing method), 43
estimate_gradient() (art.defences.PixelDefend method), 46
estimate_gradient() (art.defences.Preprocessor method), 51
estimate_gradient() (art.defences.SpatialSmoothing method), 41
estimate_gradient() (art.defences.ThermometerEncoding method), 49
estimate_gradient() (art.defences.TotalVarMin method), 50
evaluate_defence() (art.poison_detection.ActivationDefence method), 59
evaluate_defence() (art.poison_detection.PoisonFilteringDefence method), 61
ExpectationOverTransformations (class in art.wrappers), 67

F
FastGradientMethod (class in art.attacks), 12
FeatureSqueezing (class in art.defences), 39
fit() (art.classifiers.Classifier method), 36
fit() (art.classifiers.EnsembleClassifier method), 34
fit() (art.classifiers.KerasClassifier method), 23
fit() (art.classifiers.MXClassifier method), 25
fit() (art.classifiers.PyTorchClassifier method), 28
fit() (art.classifiers.TFClassifier method), 31
fit() (art.defences.AdversarialTrainer method), 43
fit() (art.defences.FeatureSqueezing method), 40
fit() (art.defences.GaussianAugmentation method), 45
fit() (art.defences.JpegCompression method), 48
fit() (art.defences.LabelSmoothing method), 43
fit() (art.defences.PixelDefend method), 47
fit() (art.defences.Preprocessor method), 52
fit() (art.defences.SpatialSmoothing method), 42
fit() (art.defences.ThermometerEncoding method), 49
fit() (art.defences.TotalVarMin method), 51
fit() (art.detection.BinaryActivationDetector method), 55
fit() (art.detection.BinaryInputDetector method), 53
fit() (art.detection.subsetscanning.SubsetScanningDetector method), 57
fit_generator() (art.classifiers.Classifier method), 36
fit_generator() (art.classifiers.EnsembleClassifier method), 34
fit_generator() (art.classifiers.KerasClassifier method), 23
fit_generator() (art.classifiers.MXClassifier method), 25
fit_generator() (art.classifiers.PyTorchClassifier method), 28
fit_generator() (art.classifiers.TFClassifier method), 31
fit_generator() (art.defences.AdversarialTrainer method), 44
fit_generator() (art.detection.BinaryActivationDetector method), 55
fit_generator() (art.detection.BinaryInputDetector method), 53
fit_generator() (art.detection.subsetscanning.SubsetScanningDetector method), 57

G
GaussianAugmentation (class in art.defences), 44
generate() (art.attacks.AdversarialPatch method), 5
generate() (art.attacks.Attack method), 21
generate() (art.attacks.BoundaryAttack method), 7
generate() (art.attacks.CarliniL2Method method), 8
generate() (art.attacks.CarliniLInfMethod method), 9
generate() (art.attacks.DeepFool method), 10
Index
set_learning_phase()
   (art.classifiers.PyTorchClassifier method), 30
set_learning_phase()
   (art.classifiers.TFClassifier method), 33
set_learning_phase()
   (art.detection.BinaryActivationDetector method), 56
set_learning_phase()
   (art.detection.BinaryInputDetector method), 54
set_learning_phase()
   (art.detection.subsetscanning.SubsetScanningDetector method), 59
set_params()
   (art.attacks.AdversarialPatch method), 6
set_params()
   (art.attacks.Attack method), 21
set_params()
   (art.attacks.BoundaryAttack method), 7
set_params()
   (art.attacks.CarliniL2Method method), 8
set_params()
   (art.attacks.CarliniLInfMethod method), 9
set_params()
   (art.attacks.DeepFool method), 10
set_params()
   (art.attacks.ElasticNet method), 11
set_params()
   (art.attacks.FastGradientMethod method), 13
set_params()
   (art.attacks.HopSkipJump method), 20
set_params()
   (art.attacks.NewtonFool method), 15
set_params()
   (art.attacksProjectedGradientDescent method), 14
set_params()
   (art.attacks.SaliencyMapMethod method), 15
set_params()
   (art.attacks.SpatialTransformation method), 16
set_params()
   (art.attacks.UniversalPerturbation method), 17
set_params()
   (art.attacks.VirtualAdversarialMethod method), 18
set_params()
   (art.attacks.ZooAttack method), 19
set_params()
   (art.defences.FeatureSqueezing method), 40
set_params()
   (art.defences.GaussianAugmentation method), 45
set_params()
   (art.defences.JpegCompression method), 48
set_params()
   (art.defences.LabelSmoothing method), 43
set_params()
   (art.defences.PixelDefend method), 47
set_params()
   (art.defences.Preprocessor method), 52
set_params()
   (art.defences.SpatialSmoothing method), 42
set_params()
   (art.defences.ThermometerEncoding method), 49
set_params()
   (art.defences.TotalVarMin method), 51
set_params()
   (art.poison_detection.ActivationDefence method), 61
set_params()
   (art.poison_detection.PoisonFilteringDefence method), 62
set_params()
   (art.wrappers.ClassifierWrapper method), 68
SpatialSmoothing (class in art.defences), 41
SpatialTransformation (class in art.attacks), 16
SubsetScanningDetector (class in art.detection.subsetscanning), 57
TanhToOriginal () (in module art.utils), 64
TFClassifier (class in art.classifiers), 30
TFDataGenerator (class in art.data_generators), 39
ThermometerEncoding (class in art.defences), 48
to_categorical() (in module art.utils), 65
TotalVarMin (class in art.defences), 50
UniversalPerturbation (class in art.attacks), 16
VirtualAdversarialMethod (class in art.attacks), 17
visualize_clusters()
   (art.poison_detection.ActivationDefence method), 61
ZooAttack (class in art.attacks), 18