Foolbox Documentation

Release 2.1.0

Jonas Rauber & Wieland Brendel

Oct 27, 2019

User Guide

Foolbox is a Python toolbox to create adversarial examples that fool neural networks.

It comes with support for many frameworks to build models including

- TensorFlow
- PyTorch
- Keras
- JAX
- MXNet
- Theano
- Lasagne

and it is easy to extend to other frameworks.

In addition, it comes with a **large collection of adversarial attacks**, both gradient-based attacks as well as black-box attacks. See *foolbox.attacks* for details.

The source code and a minimal working example can be found on GitHub.

CHAPTER 1

Installation

Foolbox is a Python package to create adversarial examples. It supports Python 3.5 and newer (try Foolbox 1.x if you still need to use Python 2.7).

1.1 Stable release

You can install the latest stable release of Foolbox from PyPI using *pip*:

```
pip install foolbox
```

Make sure that *pip* installs packages for Python 3, otherwise you might need to use *pip3* instead of *pip*.

1.2 Pre-release versions

You can install the latest stable release of Foolbox from PyPI using *pip*:

```
pip install foolbox --pre
```

Make sure that *pip* installs packages for Python 3, otherwise you might need to use *pip3* instead of *pip*.

1.3 Development version

Alternatively, you can install the latest development version of Foolbox from GitHub. We try to keep the master branch stable, so this version should usually work fine. Feel free to open an issue on GitHub if you encounter any problems.

```
pip install https://github.com/bethgelab/foolbox/archive/master.zip
```

1.4 Contributing to Foolbox

If you would like to contribute the development of Foolbox, install it in editable mode:

```
git clone https://github.com/bethgelab/foolbox.git
cd foolbox
pip install --editable .
```

To contribute your changes, you will need to fork the Foolbox repository on GitHub. You can than add it as a remote:

```
git remote add fork git@github.com/<your-github-name>/foolbox.git
```

You can now commit your changes, push them to your fork and create a pull-request to contribute them to Foolbox. See *Running Tests* for more information on the necessary tools and conventions.

CHAPTER 2

Tutorial

This tutorial will show you how an adversarial attack can be used to find adversarial examples for a model.

2.1 Creating a model

For the tutorial, we will target *VGG19* implemented in *TensorFlow*, but it is straight forward to apply the same to other models or other frameworks such as *Theano* or *PyTorch*.

```
import tensorflow as tf

images = tf.placeholder(tf.float32, (None, 224, 224, 3))
preprocessed = vgg_preprocessing(images)
logits = vgg19(preprocessed)
```

To turn a model represented as a standard TensorFlow graph into a model that can be attacked by the Adversarial Toolbox, all we have to do is to create a new *TensorFlowModel* instance:

```
from foolbox.models import TensorFlowModel

model = TensorFlowModel(images, logits, bounds=(0, 255))
```

2.2 Specifying the criterion

To run an adversarial attack, we need to specify the type of adversarial we are looking for. This can be done using the `Criterion` class.

```
from foolbox.criteria import TargetClassProbability

target_class = 22
criterion = TargetClassProbability(target_class, p=0.99)
```

2.3 Running the attack

Finally, we can create and apply the attack:

```
from foolbox.attacks import LBFGSAttack

attack = LBFGSAttack(model, criterion)
images, labels = foolbox.utils.samples(dataset='imagenet', batchsize=16, data_format=
↪'channels_last', bounds=(0, 255))
adversarial = attack(image, label=label)
```

2.4 Visualizing the adversarial examples

To plot the adversarial example we can use *matplotlib*:

```
import matplotlib.pyplot as plt

plt.subplot(1, 3, 1)
plt.imshow(image)

plt.subplot(1, 3, 2)
plt.imshow(adversarial)

plt.subplot(1, 3, 3)
plt.imshow(adversarial - image)
```

2.5 External Resources

If you would like to share your Foolbox tutorial or example code, please let us know by opening an issue or pull-request on GitHub and we would be happy to add it to this list.

- Fashion-MNIST by akash-joshi

CHAPTER 3

Examples

Here you can find a collection of examples how Foolbox models can be created using different deep learning frameworks and some full-blown attack examples at the end.

3.1 Running an attack

3.1.1 Running a batch attack against a PyTorch model

```
import foolbox
import numpy as np
import torchvision.models as models

# instantiate model (supports PyTorch, Keras, TensorFlow (Graph and Eager), MXNet and␣
↪many more)
model = models.resnet18(pretrained=True).eval()
preprocessing = dict(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225], axis=-3)
fmodel = foolbox.models.PyTorchModel(model, bounds=(0, 1), num_classes=1000,␣
↪preprocessing=preprocessing)

# get a batch of images and labels and print the accuracy
images, labels = foolbox.utils.samples(dataset='imagenet', batchsize=16, data_format=
↪'channels_first', bounds=(0, 1))
print(np.mean(fmodel.forward(images).argmax(axis=-1) == labels))
# -> 0.9375

# apply the attack
attack = foolbox.attacks.FGSM(fmodel)
adversarials = attack(images, labels)
# if the i'th image is misclassfied without a perturbation, then adversarials[i] will␣
↪be the same as images[i]
# if the attack fails to find an adversarial for the i'th image, then adversarials[i]␣
↪will all be np.nan
```

(continues on next page)

(continued from previous page)

```
# Foolbox guarantees that all returned adversarials are in fact in adversarials
print(np.mean(fmodel.forward(adversarials).argmax(axis=-1) == labels))
# -> 0.0

# ---

# In rare cases, it can happen that attacks return adversarials that are so close to␣
↪the decision boundary,
# that they actually might end up on the other (correct) side if you pass them␣
↪through the model again like
# above to get the adversarial class. This is because models are not numerically␣
↪deterministic (on GPU, some
# operations such as `sum` are non-deterministic by default) and indepedent between␣
↪samples (an input might
# be classified differently depending on the other inputs in the same batch).

# You can always get the actual adversarial class that was observed for that sample␣
↪by Foolbox by
# passing `unpack=False` to get the actual `Adversarial` objects:
attack = foolbox.attacks.FGSM(fmodel, distance=foolbox.distances.Linf)
adversarials = attack(images, labels, unpack=False)

adversarial_classes = np.asarray([a.adversarial_class for a in adversarials])
print(labels)
print(adversarial_classes)
print(np.mean(adversarial_classes == labels))  # will always be 0.0

# The `Adversarial` objects also provide a `distance` attribute. Note that the␣
↪distances
# can be 0 (misclassified without perturbation) and inf (attack failed).
distances = np.asarray([a.distance.value for a in adversarials])
print("{:.1e}, {:.1e}, {:.1e}".format(distances.min(), np.median(distances),␣
↪distances.max()))
print("{} of {} attacks failed".format(sum(adv.distance.value == np.inf for adv in␣
↪adversarials), len(adversarials)))
print("{} of {} inputs misclassified without perturbation".format(sum(adv.distance.
↪value == 0 for adv in adversarials), len(adversarials)))
```

3.1.2 Running an attack on single sample against a Keras model

```
import foolbox
import keras
import numpy as np
from keras.applications.resnet50 import ResNet50

# instantiate model
keras.backend.set_learning_phase(0)
kmodel = ResNet50(weights='imagenet')
preprocessing = dict(flip_axis=-1, mean=np.array([104, 116, 123]))  # RGB to BGR and␣
↪mean subtraction
fmodel = foolbox.models.KerasModel(kmodel, bounds=(0, 255),␣
↪preprocessing=preprocessing)

# get source image and label
```

(continues on next page)

(continued from previous page)

```
image, label = foolbox.utils.imagenet_example()

# apply attack on source image
attack = foolbox.v1.attacks.FGSM(fmodel)
adversarial = attack(image, label)
# if the attack fails, adversarial will be None and a warning will be printed
```

3.2 Creating a model

3.2.1 Keras: ResNet50

```
import keras
import numpy as np
import foolbox

keras.backend.set_learning_phase(0)
kmodel = keras.applications.resnet50.ResNet50(weights='imagenet')
preprocessing = dict(flip_axis=-1, mean=np.array([104, 116, 123]))  # RGB to BGR and␣
↪mean subtraction
model = foolbox.models.KerasModel(kmodel, bounds=(0, 255),␣
↪preprocessing=preprocessing)

image, label = foolbox.utils.imagenet_example()
print(np.argmax(model.forward_one(image)), label)
```

3.2.2 PyTorch: ResNet18

You might be interested in checking out the full PyTorch example at the end of this document.

```
import torchvision.models as models
import numpy as np
import foolbox

# instantiate the model
resnet18 = models.resnet18(pretrained=True).cuda().eval()  # for CPU, remove cuda()
mean = np.array([0.485, 0.456, 0.406]).reshape((3, 1, 1))
std = np.array([0.229, 0.224, 0.225]).reshape((3, 1, 1))
model = foolbox.models.PyTorchModel(resnet18, bounds=(0, 1), num_classes=1000,␣
↪preprocessing=(mean, std))

image, label = foolbox.utils.imagenet_example(data_format='channels_first')
image = image / 255
print(np.argmax(model.forward_one(image)), label)
```

3.2.3 TensorFlow: VGG19

First, create the model in TensorFlow.

```
import tensorflow as tf
from tensorflow.contrib.slim.nets import vgg
import numpy as np
import foolbox

images = tf.placeholder(tf.float32, shape=(None, 224, 224, 3))
preprocessed = images - [123.68, 116.78, 103.94]
logits, _ = vgg.vgg_19(preprocessed, is_training=False)
restorer = tf.train.Saver(tf.trainable_variables())

image, _ = foolbox.utils.imagenet_example()
```

Then transform it into a Foolbox model using one of these four options:

Option 1

This option is recommended if you want to keep the code as short as possible. It makes use of the TensorFlow session created by Foolbox internally if no default session is set.

```
with foolbox.models.TensorFlowModel(images, logits, (0, 255)) as model:
    restorer.restore(model.session, '/path/to/vgg_19.ckpt')
    print(np.argmax(model.forward_one(image)))
```

Option 2

This option is recommended if you want to create the TensorFlow session yourself.

```
with tf.Session() as session:
    restorer.restore(session, '/path/to/vgg_19.ckpt')
    model = foolbox.models.TensorFlowModel(images, logits, (0, 255))
    print(np.argmax(model.forward_one(image)))
```

Option 3

This option is recommended if you want to avoid nesting context managers, e.g. during interactive development.

```
session = tf.InteractiveSession()
restorer.restore(session, '/path/to/vgg_19.ckpt')
model = foolbox.models.TensorFlowModel(images, logits, (0, 255))
print(np.argmax(model.forward_one(image)))
session.close()
```

Option 4

This is possible, but usually one of the other options should be preferred.

```
session = tf.Session()
with session.as_default():
    restorer.restore(session, '/path/to/vgg_19.ckpt')
    model = foolbox.models.TensorFlowModel(images, logits, (0, 255))
    print(np.argmax(model.forward_one(image)))
session.close()
```

3.3 Applying an attack

Once you created a Foolbox model (see the previous section), you can apply an attack.

3.3.1 FGSM (GradientSignAttack)

```
# create a model (see previous section)
fmodel = ...

# get source image and label
image, label = foolbox.utils.imagenet_example()

# apply attack on source image
attack  = foolbox.v1.attacks.FGSM(fmodel)
adversarial = attack(image, label)
```

3.4 Creating an untargeted adversarial for a PyTorch model

```
import foolbox
import torch
import torchvision.models as models
import numpy as np

# instantiate the model
resnet18 = models.resnet18(pretrained=True).eval()
if torch.cuda.is_available():
    resnet18 = resnet18.cuda()
mean = np.array([0.485, 0.456, 0.406]).reshape((3, 1, 1))
std = np.array([0.229, 0.224, 0.225]).reshape((3, 1, 1))
fmodel = foolbox.models.PyTorchModel(
    resnet18, bounds=(0, 1), num_classes=1000, preprocessing=(mean, std))

# get source image and label
image, label = foolbox.utils.imagenet_example(data_format='channels_first')
image = image / 255.  # because our model expects values in [0, 1]

print('label', label)
print('predicted class', np.argmax(fmodel.forward_one(image)))

# apply attack on source image
attack = foolbox.v1.attacks.FGSM(fmodel)
adversarial = attack(image, label)

print('adversarial class', np.argmax(fmodel.forward_one(adversarial)))
```

outputs

```
label 282
predicted class 282
adversarial class 281
```

To plot image and adversarial, don't forget to move the channel axis to the end before passing them to matplotlib's imshow, e.g. using `np.transpose(image, (1, 2, 0))`.

3.5 Creating a targeted adversarial for the Keras ResNet model

```
import foolbox
from foolbox.models import KerasModel
from foolbox.attacks import LBFGSAttack
from foolbox.criteria import TargetClassProbability
import numpy as np
import keras
from keras.applications.resnet50 import ResNet50
from keras.applications.resnet50 import preprocess_input
from keras.applications.resnet50 import decode_predictions

keras.backend.set_learning_phase(0)
kmodel = ResNet50(weights='imagenet')
preprocessing = dict(flip_axis=-1, mean=np.array([104, 116, 123]))  # RGB to BGR and␣
↪mean subtraction
fmodel = KerasModel(kmodel, bounds=(0, 255), preprocessing=preprocessing)

image, label = foolbox.utils.imagenet_example()

# run the attack
attack = LBFGSAttack(model=fmodel, criterion=TargetClassProbability(781, p=.5))
adversarial = attack(image, label)

# show results
print(np.argmax(fmodel.forward_one(adversarial)))
print(foolbox.utils.softmax(fmodel.forward_one(adversarial))[781])
preds = kmodel.predict(preprocess_input(adversarial[np.newaxis].copy()))
print("Top 5 predictions (adversarial: ", decode_forward_one(preds, top=5))
```

outputs

```
781
0.832095
Top 5 predictions (adversarial:  [[('n04149813', 'scoreboard', 0.83013469), (
↪'n03196217', 'digital_clock', 0.030192226), ('n04152593', 'screen', 0.016133979), (
↪'n04141975', 'scale', 0.011708578), ('n03782006', 'monitor', 0.0091574294)]]
```

CHAPTER 4

Advanced

The Adversarial class provides an advanced way to specify the adversarial example that should be found by an attack and provides detailed information about the created adversarial. In addition, it provides a way to improve a previously found adversarial example by re-running an attack.

```
from foolbox.v1 import Adversarial
from foolbox.v1.attacks import LBFGSAttack
from foolbox.models import TenosrFlowModel
from foolbox.criteria import TargetClassProbability
```

4.1 Implicit

```
model = TensorFlowModel(inputs, logits, bounds=(0, 255))
criterion = TargetClassProbability('ostrich', p=0.99)
attack = LBFGSAttack(model, criterion)
```

Running the attack by passing an input and a label will implicitly create an Adversarial instance. By passing *unpack=False* we tell the attack to return the Adversarial instance rather than a numpy array.

```
adversarial = attack(image, label=label, unpack=False)
```

We can then get the actual adversarial input using the image attribute:

```
adversarial_image = adversarial.perturbed
```

4.2 Explicit

```
model = TensorFlowModel(images, logits, bounds=(0, 255))
criterion = TargetClassProbability('ostrich', p=0.99)
attack = LBFGSAttack()
```

We can also create the `Adversarial` instance ourselves and then pass it to the attack.

```
adversarial = Adversarial(model, criterion, image, label)
attack(adversarial)
```

Again, we can get the image using the `image` attribute:

```
adversarial_image = adversarial.perturbed
```

This approach gives us more flexibility and allows us to specify a different distance measure:

```
distance = MeanAbsoluteDistance
adversarial = Adversarial(model, criterion, image, label, distance=distance)
```

CHAPTER 5

Model Zoo

This tutorial will show you how the model zoo can be used to run your attack against a robust model.

5.1 Downloading a model

For this tutorial, we will download the *Analysis by Synthesis* model implemented in *PyTorch* and run a *FGSM (GradienSignAttack)* against it.

```
from foolbox import zoo

# download the model
model = zoo.get_model(url="https://github.com/bethgelab/AnalysisBySynthesis")

# read image and label
image = ...
label = ...

# apply attack on source image
attack  = foolbox.attacks.FGSM(model)
adversarial = attack(image, label)
```

CHAPTER 6

Development

To install Foolbox in editable mode, see the installation instructions under *Contributing to Foolbox*.

6.1 Running Tests

6.1.1 pytest

To run the tests, you need to have pytest and pytest-cov installed. Afterwards, you can simply run `pytest` in the root folder of the project. Some tests will require TensorFlow, PyTorch and the other frameworks, so to run all tests, you need to have all of them installed. Note however that this can take quite long (Foolbox has many tests) and installing all frameworks with the correct versions is difficult due to conflicting dependencies. You can also open a pull-request and then we will run all the tests using travis.

6.2 Style Guide

We use Black to format all code in a consistent and PEP-8 conform way. All pull-requests are checked using both `black` and `flake8`. Simply install `black` and run `black .` after all your changes or ideally even on each commit using pre-commit.

6.3 New Adversarial Attacks

Foolbox makes it easy to develop new adversarial attacks that can be applied to arbitrary models.

To implement an attack, simply subclass the `Attack` class, implement the `__call__()` method and decorate it with the `call_decorator()`. The `call_decorator()` will make sure that your `__call__()` implementation will be called with an instance of the `Adversarial` class. You can use this instance to ask for model predictions and gradients, get the original image and its label and more. In addition, the `Adversarial` instance automatically keeps track of the best adversarial amongst all the inputs tested by the attack. That way, the implementation of the attack can focus on the attack logic.

To implement an attack that can make use of the batch support introduced in Foolbox 2.0, implement the `as_generator()` method and decorate it with the `generator_decorator()`. All model calls using the `Adversarial` object should use `yield`.

CHAPTER 7

FAQ

How does Foolbox handle inputs that are misclassified without any perturbation? The attacks will not be run and instead the unperturbed input is returned as an *adversarial* with distance 0 to the clean input.

What happens if an attack fails? The attack will return *None* and the distance will be *np.inf*.

Why is the returned adversarial not misclassified by my model? Most likely you have a discrepancy between how you evaluate your model and how you told Foolbox to evaluate it. For example, you might not be using the same preprocessing. Compare the output of the *predictions* method of the Foolbox model instance with your model's output (logits). This problem can also be caused by non-deterministic models. Make sure that your model is not stochastic and always returns the same output when given the same input. In rare cases it can also be that a seemlingly deterministic model becomes numerically stochastic around the decision boundary (e.g. because of non-deterministic floating point *reduce_sum* operations). You can always check *adversarial.output* and *adversarial.adversarial_class* to see the output Foolbox got from your model when deciding that this was an adversarial.

Why are the gradients multiplied by the bounds (*max_ - min_*)? This scaling is meant to make hyperparameters such as the *epsilon* for FGSM independent of the bounds. *epsilon = 0.1* thus means that you perturb the input by 10% relative to the *max - max* range (which could for example go from 0 to 1 or from 0 to 255).

CHAPTER 8

foolbox.models

Provides classes to wrap existing models in different framworks so that they provide a unified API to the attacks.

8.1 Models

Model	Base class to provide attacks with a unified interface to models.
DifferentiableModel	Base class for differentiable models.
TensorFlowModel	Creates a *Model* instance from existing *TensorFlow* tensors.
TensorFlowEagerModel	Creates a *Model* instance from a *TensorFlow* model using eager execution.
PyTorchModel	Creates a *Model* instance from a *PyTorch* module.
KerasModel	Creates a *Model* instance from a *Keras* model.
TheanoModel	Creates a *Model* instance from existing *Theano* tensors.
LasagneModel	Creates a *Model* instance from a *Lasagne* network.
MXNetModel	Creates a *Model* instance from existing *MXNet* symbols and weights.
MXNetGluonModel	Creates a *Model* instance from an existing *MXNet Gluon* Block.
JAXModel	Creates a *Model* instance from a *JAX* predict function.
CaffeModel	

8.2 Wrappers

ModelWrapper	Base class for models that wrap other models.

Continued on next page

Table 2 – continued from previous page

`DifferentiableModelWrapper`	Base class for models that wrap other models and provide gradient methods.
`ModelWithoutGradients`	Turns a model into a model without gradients.
`ModelWithEstimatedGradients`	Turns a model into a model with gradients estimated by the given gradient estimator.
`CompositeModel`	Combines predictions of a (black-box) model with the gradient of a (substitute) model.

8.3 Detailed description

class `foolbox.models.`**`Model`**(*bounds*, *channel_axis*, *preprocessing=(0, 1)*)

Base class to provide attacks with a unified interface to models.

The `Model` class represents a model and provides a unified interface to its predictions. Subclasses must implement forward and num_classes.

`Model` instances can be used as context managers and subclasses can require this to allocate and release resources.

Parameters

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

channel_axis [int] The index of the axis that represents color channels.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

`forward`(*self*, *inputs*)

Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

`forward_one`(*self*, *x*)

Takes a single input and returns the logits predicted by the underlying model.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

Returns

numpy.ndarray Predicted logits with shape (number of classes,).

See also:

forward()

num_classes (*self*)
Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**DifferentiableModel** (*bounds*, *channel_axis*, *preprocessing=(0, 1)*)
Base class for differentiable models.

The *DifferentiableModel* class can be used as a base class for models that can support gradient back-propagation. Subclasses must implement gradient and backward.

A differentiable model does not necessarily provide reasonable values for the gradient, the gradient can be wrong. It only guarantees that the relevant methods can be called.

backward (*self*, *gradient*, *inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

backward_one()

gradient()

backward_one (*self*, *gradient*, *x*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the input.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (number of classes,).

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

Returns

numpy.ndarray The gradient of the respective loss w.r.t the input.

See also:

backward()

forward_and_gradient (*self*, *inputs*, *labels*)
Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

forward_and_gradient_one (*self*, *x*, *label*)
Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

gradient (*self*, *inputs*, *labels*)
Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

gradient_one()

backward()

gradient_one (*self, x, label*)
Takes a single input and label and returns the gradient of the cross-entropy loss w.r.t. the input.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

gradient()

class foolbox.models.**TensorFlowModel** (*inputs, logits, bounds, channel_axis=3, preprocessing=(0, 1)*)
Creates a *Model* instance from existing *TensorFlow* tensors.

Parameters

inputs [*tensorflow.Tensor*] The input to the model, usually a *tensorflow.placeholder*.

logits [*tensorflow.Tensor*] The predictions of the model, before the softmax.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

channel_axis [int] The index of the axis that represents color channels.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

backward (*self, gradient, inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

backward_one()

gradient()

forward(*self*, *inputs*)
Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

forward_one()

forward_and_gradient(*self*, *inputs*, *labels*)
Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

forward_and_gradient_one(*self*, *x*, *label*)
Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

classmethod from_keras (*model*, *bounds*, *input_shape=None*, *channel_axis='auto'*, *preprocessing=(0, 1)*)

Alternative constructor for a TensorFlowModel that accepts a *tf.keras.Model* instance.

Parameters

model [*tensorflow.keras.Model*] A *tensorflow.keras.Model* that accepts a single input tensor and returns a single output tensor representing logits.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

input_shape [tuple] The shape of a single input, e.g. (28, 28, 1) for MNIST. If None, tries to get the the shape from the model's input_shape attribute.

channel_axis [int or 'auto'] The index of the axis that represents color channels. If 'auto', will be set automatically based on keras.backend.image_data_format()

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

gradient (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

num_classes (*self*)

Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**TensorFlowEagerModel** (*model*, *bounds*, *num_classes=None*, *channel_axis=3*, *preprocessing=(0, 1)*)

Creates a `Model` instance from a *TensorFlow* model using eager execution.

Parameters

model [a TensorFlow eager model] The TensorFlow eager model that should be attacked. It will be called with input tensors and should return logits.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

num_classes [int] If None, will try to infer it from the model's output shape.

channel_axis [int] The index of the axis that represents color channels.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

backward (*self*, *gradient*, *inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

backward_one()

gradient()

forward (*self*, *inputs*)
Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

forward_one()

forward_and_gradient (*self*, *inputs*, *labels*)
Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`forward_and_gradient_one` (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`gradient` (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

`num_classes` (*self*)

Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class `foolbox.models.`**`PyTorchModel`** (*model*, *bounds*, *num_classes*, *channel_axis=1*, *device=None*, *preprocessing=(0, 1)*)

Creates a *`Model`* instance from a *PyTorch* module.

Parameters

model [*torch.nn.Module*] The PyTorch model that should be attacked. It should predict logits or log-probabilities, i.e. predictions without the softmax.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

num_classes [int] Number of classes for which the model will output predictions.

channel_axis [int] The index of the axis that represents color channels.

device [string] A string specifying the device to do computation on. If None, will default to "cuda:0" if torch.cuda.is_available() or "cpu" if not.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

`backward`(*self*, *gradient*, *inputs*)

Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

`backward_one()`

`gradient()`

`forward`(*self*, *inputs*)

Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

`forward_and_gradient`(*self*, *inputs*, *labels*)

Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`forward_and_gradient_one` (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`gradient` (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

num_classes (*self*)
Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**JAXModel** (*predict*, *bounds*, *num_classes*, *channel_axis=3*, *preprocessing=(0, 1)*)
Creates a *Model* instance from a *JAX* predict function.

Parameters

predict [*function*] The JAX-compatible function that takes a batch of inputs as and returns a batch of predictions (logits); use functools.partial(predict, params) to pass params if necessary

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

num_classes [int] Number of classes for which the model will output predictions.

channel_axis [int] The index of the axis that represents color channels.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

backward (*self*, *gradient*, *inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

backward_one()

gradient()

forward (*self*, *inputs*)
Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

forward_one()

forward_and_gradient(*self*, *inputs*, *labels*)
Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

gradient(*self*, *inputs*, *labels*)
Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

gradient_one()

backward()

num_classes(*self*)
Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**KerasModel**(*model*, *bounds*, *channel_axis='auto'*, *preprocessing=(0, 1)*, *predicts='probabilities'*)
Creates a *Model* instance from a *Keras* model.

Parameters

model [*keras.models.Model*] The *Keras* model that should be attacked.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

channel_axis [int or 'auto'] The index of the axis that represents color channels. If 'auto', will be set automatically based on keras.backend.image_data_format()

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

predicts [str] Specifies whether the *Keras* model predicts logits or probabilities. Logits are preferred, but probabilities are the default.

backward (*self*, *gradient*, *inputs*)

Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

`backward_one()`

`gradient()`

forward (*self*, *inputs*)

Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

forward_and_gradient (*self*, *inputs*, *labels*)

Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

forward_and_gradient_one (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

gradient (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

gradient_one()

backward()

num_classes (*self*)

Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**TheanoModel** (*inputs, logits, bounds, num_classes, channel_axis=1, preprocessing=[0, 1]*)

Creates a *Model* instance from existing *Theano* tensors.

Parameters

inputs [*theano.tensor*] The input to the model.

logits [*theano.tensor*] The predictions of the model, before the softmax.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

num_classes [int] Number of classes for which the model will output predictions.

channel_axis [int] The index of the axis that represents color channels.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

backward (*self*, *gradient*, *inputs*)

Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

backward_one()

gradient()

forward (*self*, *inputs*)

Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

forward_one()

forward_and_gradient (*self*, *inputs*, *labels*)

Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`forward_and_gradient_one` (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`gradient` (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

num_classes (*self*)
Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**LasagneModel** (*input_layer*, *logits_layer*, *bounds*, *channel_axis=1*, *preprocessing=(0, 1)*)
Creates a *Model* instance from a *Lasagne* network.

Parameters

input_layer [*lasagne.layers.Layer*] The input to the model.

logits_layer [*lasagne.layers.Layer*] The output of the model, before the softmax.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

channel_axis [int] The index of the axis that represents color channels.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

class foolbox.models.**MXNetModel** (*data*, *logits*, *args*, *ctx*, *num_classes*, *bounds*, *channel_axis=1*, *aux_states=None*, *preprocessing=(0, 1)*)
Creates a *Model* instance from existing *MXNet* symbols and weights.

Parameters

data [*mxnet.symbol.Variable*] The input to the model.

logits [*mxnet.symbol.Symbol*] The predictions of the model, before the softmax.

args [*dictionary mapping str to mxnet.nd.array*] The parameters of the model.

ctx [*mxnet.context.Context*] The device, e.g. mxnet.cpu() or mxnet.gpu().

num_classes [int] The number of classes.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

channel_axis [int] The index of the axis that represents color channels.

aux_states [*dictionary mapping str to mxnet.nd.array*] The states of auxiliary parameters of the model.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

backward (*self*, *gradient*, *inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

`backward_one()`

`gradient()`

`forward(self, inputs)`

Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

`forward_and_gradient(self, inputs, labels)`

Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`forward_and_gradient_one(self, x, label)`

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`gradient` (*self*, *inputs*, *labels*)
Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

`num_classes` (*self*)
Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class `foolbox.models.`**`MXNetGluonModel`** (*block*, *bounds*, *num_classes*, *ctx=None*, *channel_axis=1*, *preprocessing=(0, 1)*)
Creates a `Model` instance from an existing *MXNet Gluon* Block.

Parameters

block [*mxnet.gluon.Block*] The Gluon Block representing the model to be run.

ctx [*mxnet.context.Context*] The device, e.g. mxnet.cpu() or mxnet.gpu().

num_classes [int] The number of classes.

bounds [tuple] Tuple of lower and upper bound for the pixel values, usually (0, 1) or (0, 255).

channel_axis [int] The index of the axis that represents color channels.

preprocessing: dict or tuple Can be a tuple with two elements representing mean and standard deviation or a dict with keys "mean" and "std". The two elements should be floats or numpy arrays. "mean" is subtracted from the input, the result is then divided by "std". If "mean" and "std" are 1-dimensional arrays, an additional (negative) "axis" key can be given such that "mean" and "std" will be broadcasted to that axis (typically -1 for "channels_last" and -3 for "channels_first", but might be different when using e.g. 1D convolutions). Finally, a (negative) "flip_axis" can be specified. This axis will be flipped (before "mean" is subtracted), e.g. to convert RGB to BGR.

backward (*self*, *gradient*, *inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

`backward_one()`

`gradient()`

forward (*self*, *inputs*)
Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

forward_and_gradient (*self*, *inputs*, *labels*)
Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

forward_and_gradient_one (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

gradient (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

num_classes (*self*)

Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**CaffeModel** (*net*, *bounds*, *channel_axis=1*, *preprocessing=(0, 1)*, *data_blob_name='data'*, *label_blob_name='label'*, *output_blob_name='output'*)

backward (*self*, *gradient*, *inputs*)

Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

`backward_one()`

`gradient()`

`forward`(*self*, *inputs*)
Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

`forward_and_gradient`(*self*, *inputs*, *labels*)
Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`forward_and_gradient_one`(*self*, *x*, *label*)
Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`gradient` (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

`num_classes` (*self*)

Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

`class foolbox.models.ModelWrapper` (*model*)

Base class for models that wrap other models.

This base class can be used to implement model wrappers that turn models into new models, for example by preprocessing the input or modifying the gradient.

Parameters

model [`Model`] The model that is wrapped.

`forward` (*self*, *inputs*)

Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

num_classes (*self*)
Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

class foolbox.models.**DifferentiableModelWrapper** (*model*)
Base class for models that wrap other models and provide gradient methods.

This base class can be used to implement model wrappers that turn models into new models, for example by preprocessing the input or modifying the gradient.

Parameters

model [*Model*] The model that is wrapped.

backward (*self*, *gradient*, *inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

backward_one()

gradient()

forward_and_gradient (*self*, *x*, *label*)
Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

forward_and_gradient_one (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

gradient (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

gradient_one()

backward()

class foolbox.models.**ModelWithoutGradients** (*model*)

Turns a model into a model without gradients.

class foolbox.models.**ModelWithEstimatedGradients** (*model*, *gradient_estimator*)

Turns a model into a model with gradients estimated by the given gradient estimator.

Parameters

model [*Model*] The model that is wrapped.

gradient_estimator [*callable*] Callable taking three arguments (pred_fn, x, label) and returning the estimated gradients. pred_fn will be the forward method of the wrapped model.

backward (*self*, *gradient*, *inputs*)

Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

`backward_one()`

`gradient()`

`forward_and_gradient` (*self*, *inputs*, *labels*)

Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

`forward_and_gradient_one` (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

`forward_one()`

`gradient_one()`

gradient (*self*, *inputs*, *labels*)
Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

`gradient_one()`

`backward()`

class foolbox.models.**CompositeModel** (*forward_model*, *backward_model*)
Combines predictions of a (black-box) model with the gradient of a (substitute) model.

Parameters

forward_model [`Model`] The model that should be fooled and will be used for predictions.

backward_model [`Model`] The model that provides the gradients.

backward (*self*, *gradient*, *inputs*)
Backpropagates the gradient of some loss w.r.t. the logits through the underlying model and returns the gradient of that loss w.r.t to the inputs.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits with shape (batch size, number of classes).

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray The gradient of the respective loss w.r.t the inputs.

See also:

`backward_one()`

`gradient()`

forward (*self*, *inputs*)
Takes a batch of inputs and returns the logits predicted by the underlying model.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

See also:

`forward_one()`

forward_and_gradient (*self*, *inputs*, *labels*)

Takes inputs and labels and returns both the logits predicted by the underlying model and the gradients of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Inputs with shape as expected by the model (with the batch dimension).

labels [*numpy.ndarray*] Array of the class label of the inputs as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

forward_and_gradient_one (*self*, *x*, *label*)

Takes a single input and label and returns both the logits predicted by the underlying model and the gradient of the cross-entropy loss w.r.t. the input.

Defaults to individual calls to forward_one and gradient_one but can be overriden by subclasses to provide a more efficient implementation.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

label [int] Class label of the input as an integer in [0, number of classes).

Returns

numpy.ndarray Predicted logits with shape (batch size, number of classes).

numpy.ndarray The gradient of the cross-entropy loss w.r.t. the input.

See also:

forward_one()

gradient_one()

gradient (*self*, *inputs*, *labels*)

Takes a batch of inputs and labels and returns the gradient of the cross-entropy loss w.r.t. the inputs.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

Returns

gradient [*numpy.ndarray*] The gradient of the cross-entropy loss w.r.t. the inputs.

See also:

gradient_one()

backward()

num_classes (*self*)

Determines the number of classes.

Returns

int The number of classes for which the model creates predictions.

CHAPTER 9

foolbox.criteria

Provides classes that define what is adversarial.

9.1 Criteria

We provide criteria for untargeted and targeted adversarial attacks.

Misclassification	Defines adversarials as inputs for which the predicted class is not the original class.
TopKMisclassification	Defines adversarials as inputs for which the original class is not one of the top k predicted classes.
OriginalClassProbability	Defines adversarials as inputs for which the probability of the original class is below a given threshold.
ConfidentMisclassification	Defines adversarials as inputs for which the probability of any class other than the original is above a given threshold.

TargetClass	Defines adversarials as inputs for which the predicted class is the given target class.
TargetClassProbability	Defines adversarials as inputs for which the probability of a given target class is above a given threshold.

9.2 Examples

Untargeted criteria:

```
>>> from foolbox.criteria import Misclassification
>>> criterion1 = Misclassification()
```

```
>>> from foolbox.criteria import TopKMisclassification
>>> criterion2 = TopKMisclassification(k=5)
```

Targeted criteria:

```
>>> from foolbox.criteria import TargetClass
>>> criterion3 = TargetClass(22)
```

```
>>> from foolbox.criteria import TargetClassProbability
>>> criterion4 = TargetClassProbability(22, p=0.99)
```

Criteria can be combined to create a new criterion:

```
>>> criterion5 = criterion2 & criterion3
```

9.3 Detailed description

class foolbox.criteria.**Criterion**

Base class for criteria that define what is adversarial.

The *Criterion* class represents a criterion used to determine if predictions for an image are adversarial given a reference label. It should be subclassed when implementing new criteria. Subclasses must implement is_adversarial.

is_adversarial (*self*, *predictions*, *label*)

Decides if predictions for an image are adversarial given a reference label.

Parameters

predictions [numpy.ndarray] A vector with the pre-softmax predictions for some image.

label [int] The label of the unperturbed reference image.

Returns

bool True if an image with the given predictions is an adversarial example when the ground-truth class is given by label, False otherwise.

name (*self*)

Returns a human readable name that uniquely identifies the criterion with its hyperparameters.

Returns

str Human readable name that uniquely identifies the criterion with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.criteria.**Misclassification**

Defines adversarials as inputs for which the predicted class is not the original class.

See also:

TopKMisclassification

Notes

Uses *numpy.argmax* to break ties.

is_adversarial(*self*, *predictions*, *label*)

Decides if predictions for an image are adversarial given a reference label.

Parameters

predictions [`numpy.ndarray`] A vector with the pre-softmax predictions for some image.

label [int] The label of the unperturbed reference image.

Returns

bool True if an image with the given predictions is an adversarial example when the ground-truth class is given by label, False otherwise.

name(*self*)

Returns a human readable name that uniquely identifies the criterion with its hyperparameters.

Returns

str Human readable name that uniquely identifies the criterion with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class `foolbox.criteria.`**ConfidentMisclassification**(*p*)

Defines adversarials as inputs for which the probability of any class other than the original is above a given threshold.

Parameters

p [float] The threshold probability. If the probability of any class other than the original is at least p, the image is considered an adversarial. It must satisfy 0 <= p <= 1.

is_adversarial(*self*, *predictions*, *label*)

Decides if predictions for an image are adversarial given a reference label.

Parameters

predictions [`numpy.ndarray`] A vector with the pre-softmax predictions for some image.

label [int] The label of the unperturbed reference image.

Returns

bool True if an image with the given predictions is an adversarial example when the ground-truth class is given by label, False otherwise.

name(*self*)

Returns a human readable name that uniquely identifies the criterion with its hyperparameters.

Returns

str Human readable name that uniquely identifies the criterion with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.criteria.**TopKMisclassification**(*k*)

Defines adversarials as inputs for which the original class is not one of the top k predicted classes.

For k = 1, the *Misclassification* class provides a more efficient implementation.

Parameters

k [int] Number of top predictions to which the reference label is compared to.

See also:

Misclassification Provides a more effcient implementation for k = 1.

Notes

Uses *numpy.argsort* to break ties.

is_adversarial(*self*, *predictions*, *label*)

Decides if predictions for an image are adversarial given a reference label.

Parameters

predictions [numpy.ndarray] A vector with the pre-softmax predictions for some image.

label [int] The label of the unperturbed reference image.

Returns

bool True if an image with the given predictions is an adversarial example when the ground-truth class is given by label, False otherwise.

name(*self*)

Returns a human readable name that uniquely identifies the criterion with its hyperparameters.

Returns

str Human readable name that uniquely identifies the criterion with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.criteria.**TargetClass**(*target_class*)

Defines adversarials as inputs for which the predicted class is the given target class.

Parameters

target_class [int] The target class that needs to be predicted for an image to be considered an adversarial.

Notes

Uses *numpy.argmax* to break ties.

is_adversarial (*self*, *predictions*, *label*)
Decides if predictions for an image are adversarial given a reference label.

Parameters

predictions [`numpy.ndarray`] A vector with the pre-softmax predictions for some image.

label [int] The label of the unperturbed reference image.

Returns

bool True if an image with the given predictions is an adversarial example when the ground-truth class is given by label, False otherwise.

name (*self*)
Returns a human readable name that uniquely identifies the criterion with its hyperparameters.

Returns

str Human readable name that uniquely identifies the criterion with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class `foolbox.criteria.`**OriginalClassProbability** (*p*)
Defines adversarials as inputs for which the probability of the original class is below a given threshold.

This criterion alone does not guarantee that the class predicted for the adversarial image is not the original class (unless p < 1 / number of classes). Therefore, it should usually be combined with a classifcation criterion.

Parameters

p [float] The threshold probability. If the probability of the original class is below this threshold, the image is considered an adversarial. It must satisfy 0 <= p <= 1.

is_adversarial (*self*, *predictions*, *label*)
Decides if predictions for an image are adversarial given a reference label.

Parameters

predictions [`numpy.ndarray`] A vector with the pre-softmax predictions for some image.

label [int] The label of the unperturbed reference image.

Returns

bool True if an image with the given predictions is an adversarial example when the ground-truth class is given by label, False otherwise.

name (*self*)
Returns a human readable name that uniquely identifies the criterion with its hyperparameters.

Returns

str Human readable name that uniquely identifies the criterion with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.criteria.**TargetClassProbability**(*target_class*, *p*)

Defines adversarials as inputs for which the probability of a given target class is above a given threshold.

If the threshold is below 0.5, this criterion does not guarantee that the class predicted for the adversarial image is not the original class. In that case, it should usually be combined with a classification criterion.

Parameters

target_class [int] The target class for which the predicted probability must be above the threshold probability p, otherwise the image is not considered an adversarial.

p [float] The threshold probability. If the probability of the target class is above this threshold, the image is considered an adversarial. It must satisfy 0 <= p <= 1.

is_adversarial(*self*, *predictions*, *label*)

Decides if predictions for an image are adversarial given a reference label.

Parameters

predictions [`numpy.ndarray`] A vector with the pre-softmax predictions for some image.

label [int] The label of the unperturbed reference image.

Returns

bool True if an image with the given predictions is an adversarial example when the ground-truth class is given by label, False otherwise.

name(*self*)

Returns a human readable name that uniquely identifies the criterion with its hyperparameters.

Returns

str Human readable name that uniquely identifies the criterion with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

CHAPTER 10

foolbox.zoo

10.1 Get Model

foolbox.zoo.**get_model**(*url*, *module_name='foolbox_model'*, ***kwargs*)

Provides utilities to download foolbox-compatible robust models to easily test attacks against them by simply providing a git-URL.

Examples

Instantiate a model:

```
>>> from foolbox import zoo
>>> url = "https://github.com/bveliqi/foolbox-zoo-dummy.git"
>>> model = zoo.get_model(url)  # doctest: +SKIP
```

Only works with a foolbox-zoo compatible repository. I.e. models need to have a *foolbox_model.py* file with a *create()*-function, which returns a foolbox-wrapped model.

Using the kwargs parameter it is possible to input an arbitrary number of parameters to this methods call. These parameters are forwarded to the instantiated model.

Example repositories:

- https://github.com/bethgelab/AnalysisBySynthesis
- https://github.com/bethgelab/mnist_challenge
- https://github.com/bethgelab/cifar10_challenge
- https://github.com/bethgelab/convex_adversarial
- https://github.com/wielandbrendel/logit-pairing-foolbox.git
- https://github.com/bethgelab/defensive-distillation.git

Parameters

- **url** – URL to the git repository
- **module_name** – the name of the module to import
- **kwargs** – Optional set of parameters that will be used by the to be instantiated model.

Returns a foolbox-wrapped model instance

10.2 Fetch Weights

foolbox.zoo.**fetch_weights**(*weights_uri*, *unzip=False*)

Provides utilities to download and extract packages containing model weights when creating foolbox-zoo compatible repositories, if the weights are not part of the repository itself.

Examples

Download and unzip weights:

```
>>> from foolbox import zoo
>>> url = 'https://github.com/MadryLab/mnist_challenge_models/raw/master/secret.
↪zip'  # noqa F501
>>> weights_path = zoo.fetch_weights(url, unzip=True)
```

Parameters

- **weights_uri** – the URI to fetch the weights from
- **unzip** – should be *True* if the file to be downloaded is a zipped package

Returns local path where the weights have been downloaded and potentially unzipped to

CHAPTER 11

foolbox.distances

Provides classes to measure the distance between inputs.

11.1 Distances

MeanSquaredDistance	Calculates the mean squared error between two inputs.
MeanAbsoluteDistance	Calculates the mean absolute error between two inputs.
Linfinity	Calculates the L-infinity norm of the difference between two inputs.
L0	Calculates the L0 norm of the difference between two inputs.
ElasticNet	Calculates the Elastic-Net distance between two inputs.

11.2 Aliases

MSE	alias of *foolbox.distances.MeanSquaredDistance*
MAE	alias of *foolbox.distances.MeanAbsoluteDistance*
Linf	alias of *foolbox.distances.Linfinity*
EN	Creates a class definition that assigns ElasticNet a fixed l1_factor.

11.3 Base class

To implement a new distance, simply subclass the *Distance* class and implement the `_calculate()` method.

Distance	Base class for distances.

11.4 Detailed description

class foolbox.distances.**Distance**(*reference=None*, *other=None*, *bounds=None*, *value=None*)
Base class for distances.

This class should be subclassed when implementing new distances. Subclasses must implement _calculate.

class foolbox.distances.**MeanSquaredDistance**(*reference=None*, *other=None*, *bounds=None*, *value=None*)
Calculates the mean squared error between two inputs.

class foolbox.distances.**MeanAbsoluteDistance**(*reference=None*, *other=None*, *bounds=None*, *value=None*)
Calculates the mean absolute error between two inputs.

class foolbox.distances.**Linfinity**(*reference=None*, *other=None*, *bounds=None*, *value=None*)
Calculates the L-infinity norm of the difference between two inputs.

class foolbox.distances.**L0**(*reference=None*, *other=None*, *bounds=None*, *value=None*)
Calculates the L0 norm of the difference between two inputs.

foolbox.distances.**MSE**
alias of *foolbox.distances.MeanSquaredDistance*

foolbox.distances.**MAE**
alias of *foolbox.distances.MeanAbsoluteDistance*

foolbox.distances.**Linf**
alias of *foolbox.distances.Linfinity*

CHAPTER 12

foolbox.attacks

12.1 Gradient-based attacks

class foolbox.attacks.**GradientAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Perturbs the input with the gradient of the loss w.r.t. the input, gradually increasing the magnitude until the input is misclassified.

Does not do anything if the model does not have a gradient.

as_generator(*self*, *a*, *epsilons=1000*, *max_epsilon=1*)

Perturbs the input with the gradient of the loss w.r.t. the input, gradually increasing the magnitude until the input is misclassified.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the gradient direction or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

class foolbox.attacks.**GradientSignAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Adds the sign of the gradient to the input, gradually increasing the magnitude until the input is misclassified.

This attack is often referred to as Fast Gradient Sign Method and was introduced in [R20d0064ee4c9-1].

Does not do anything if the model does not have a gradient.

References

[R20d0064ee4c9-1]

as_generator(*self*, *a*, *epsilons=1000*, *max_epsilon=1*)

Adds the sign of the gradient to the input, gradually increasing the magnitude until the input is misclassified.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the direction of the sign of the gradient or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

`foolbox.attacks.`**`FGSM`**

alias of `foolbox.attacks.gradient.GradientSignAttack`

class `foolbox.attacks.`**`LinfinityBasicIterativeAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The Basic Iterative Method introduced in [R37dbc8f24aee-1].

This attack is also known as Projected Gradient Descent (PGD) (without random start) or FGMS^k.

References

See also:

ProjectedGradientDescentAttack

[R37dbc8f24aee-1]

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**`BasicIterativeMethod`**

alias of `foolbox.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`

`foolbox.attacks.`**`BIM`**

alias of `foolbox.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`

`class` `foolbox.attacks.`**`L1BasicIterativeAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Modified version of the Basic Iterative Method that minimizes the L1 distance.

See also:

`LinfinityBasicIterativeAttack`

`as_generator`(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**L2BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Modified version of the Basic Iterative Method that minimizes the L2 distance.

See also:

LinfinityBasicIterativeAttack

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**ProjectedGradientDescentAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The Projected Gradient Descent Attack introduced in [R367e8e10528a-1] without random start.

When used without a random start, this attack is also known as Basic Iterative Method (BIM) or FGSM^k.

References

See also:

LinfinityBasicIterativeAttack and *RandomStartProjectedGradientDescentAttack*

[R367e8e10528a-1]

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.01*, *iterations=40*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**`ProjectedGradientDescent`**

alias of `foolbox.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`

`foolbox.attacks.`**`PGD`**

alias of `foolbox.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`

class `foolbox.attacks.`**`RandomStartProjectedGradientDescentAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassificatio object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The Projected Gradient Descent Attack introduced in [Re6066bc39e14-1] with random start.

References

See also:

ProjectedGradientDescentAttack

[Re6066bc39e14-1]

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.01*, *iterations=40*, *random_start=True*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

foolbox.attacks.**RandomProjectedGradientDescent**

alias of `foolbox.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`

foolbox.attacks.**RandomPGD**

alias of `foolbox.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`

class foolbox.attacks.**AdamL1BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Modified version of the Basic Iterative Method that minimizes the L1 distance using the Adam optimizer.

See also:

LinfinityBasicIterativeAttack

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**AdamL2BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Modified version of the Basic Iterative Method that minimizes the L2 distance using the Adam optimizer.

See also:

`LinfinityBasicIterativeAttack`

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**AdamProjectedGradientDescentAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

The Projected Gradient Descent Attack introduced in [Re2d4f39a0205-1], [Re2d4f39a0205-2] without random start using the Adam optimizer.

When used without a random start, this attack is also known as Basic Iterative Method (BIM) or FGSM^k.

References

See also:

LinfinityBasicIterativeAttack and *RandomStartProjectedGradientDescentAttack*

[Re2d4f39a0205-1], [Re2d4f39a0205-2]

as_generator(*self, a, binary_search=True, epsilon=0.3, stepsize=0.01, iterations=40, random_start=False, return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or Adversarial] The original, unperturbed input as a *numpy.ndarray* or an Adversarial instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an Adversarial instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**`AdamProjectedGradientDescent`**
alias of `foolbox.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`

`foolbox.attacks.`**`AdamPGD`**
alias of `foolbox.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`

`class` `foolbox.attacks.`**`AdamRandomStartProjectedGradientDescentAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassi object>*, *distance=<class 'foolbox.distances.MeanSquaredDista*, *threshold=None*)

The Projected Gradient Descent Attack introduced in [R3210aa339085-1], [R3210aa339085-2] with random start using the Adam optimizer.

References

See also:

`ProjectedGradientDescentAttack`

[R3210aa339085-1], [R3210aa339085-2]

`as_generator`(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.01*, *iterations=40*, *random_start=True*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**`AdamRandomProjectedGradientDescent`**
alias of `foolbox.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack`

`foolbox.attacks.`**`AdamRandomPGD`**
alias of `foolbox.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack`

`class` `foolbox.attacks.`**`MomentumIterativeAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The Momentum Iterative Method attack introduced in [R86d363e1fb2f-1]. It's like the Basic Iterative Method or Projected Gradient Descent except that it uses momentum.

References

[R86d363e1fb2f-1]

`as_generator`(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.06*, *iterations=10*, *decay_factor=1.0*, *random_start=False*, *return_early=True*)

Momentum-based iterative gradient attack known as Momentum Iterative Method.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

decay_factor [float] Decay factor used by the momentum term.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.MomentumIterativeMethod`
alias of `foolbox.attacks.iterative_projected_gradient.MomentumIterativeAttack`

class `foolbox.attacks.DeepFoolAttack`(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Simple and close to optimal gradient-based adversarial attack.

Implementes DeepFool introduced in [Rb4dd02640756-1].

References

[Rb4dd02640756-1]

as_generator(*self*, *a*, *steps=100*, *subsample=10*, *p=None*)

Simple and close to optimal gradient-based adversarial attack.

Parameters

- **input_or_adv** [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.
- **label** [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.
- **unpack** [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.
- **steps** [int] Maximum number of steps to perform.
- **subsample** [int] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster.
- **p** [int or float] Lp-norm that should be minimzed, must be 2 or np.inf.

class `foolbox.attacks.NewtonFoolAttack`(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Implements the NewtonFool Attack.

The attack was introduced in [R6a972939b320-1].

References

[R6a972939b320-1]

as_generator(*self*, *a*, *max_iter=100*, *eta=0.01*)

Parameters

- **input_or_adv** [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.
- **label** [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.
- **unpack** [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.
- **max_iter** [int] The maximum number of iterations.
- **eta** [float] the eta coefficient

class foolbox.attacks.**DeepFoolL2Attack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

as_generator(*self*, *a*, *steps=100*, *subsample=10*)
Simple and close to optimal gradient-based adversarial attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

steps [int] Maximum number of steps to perform.

subsample [int] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster.

p [int or float] Lp-norm that should be minimzed, must be 2 or np.inf.

class foolbox.attacks.**DeepFoolLinfinityAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

as_generator(*self*, *a*, *steps=100*, *subsample=10*)
Simple and close to optimal gradient-based adversarial attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

steps [int] Maximum number of steps to perform.

subsample [int] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster.

p [int or float] Lp-norm that should be minimzed, must be 2 or np.inf.

class foolbox.attacks.**ADefAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Adversarial attack that distorts the image, i.e. changes the locations of pixels. The algorithm is described in [Rf241e6d2664d-1], a Repository with the original code can be found in [Rf241e6d2664d-2]. References ——— .. [Rf241e6d2664d-1] Rima Alaifari, Giovanni S. Alberti, and Tandri Gauksson:

"ADef: an Iterative Algorithm to Construct Adversarial Deformations", https://arxiv.org/abs/1804.07729

as_generator(*self*, *a*, *max_iter=100*, *smooth=1.0*, *subsample=10*)

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

max_iter [int > 0] Maximum number of iterations (default max_iter = 100).

smooth [float >= 0] Width of the Gaussian kernel used for smoothing. (default is smooth = 0 for no smoothing).

subsample [int >= 2] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster. (default subsample = 10)

class foolbox.attacks.**SaliencyMapAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Implements the Saliency Map Attack.

The attack was introduced in [R08e06ca693ba-1].

References

[R08e06ca693ba-1]

as_generator(*self, a, max_iter=2000, num_random_targets=0, fast=True, theta=0.1, max_perturbations_per_pixel=7*)

Implements the Saliency Map Attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

max_iter [int] The maximum number of iterations to run.

num_random_targets [int] Number of random target classes if no target class is given by the criterion.

fast [bool] Whether to use the fast saliency map calculation.

theta [float] perturbation per pixel relative to [min, max] range.

max_perturbations_per_pixel [int] Maximum number of times a pixel can be modified.

class foolbox.attacks.**IterativeGradientAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Like GradientAttack but with several steps for each epsilon.

as_generator(*self, a, epsilons=100, max_epsilon=1, steps=10*)

Like GradientAttack but with several steps for each epsilon.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the gradient direction or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

steps [int] Number of iterations to run.

class `foolbox.attacks.`**`IterativeGradientSignAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Like GradientSignAttack but with several steps for each epsilon.

`as_generator`(*self*, *a*, *epsilons=100*, *max_epsilon=1*, *steps=10*)

Like GradientSignAttack but with several steps for each epsilon.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the direction of the sign of the gradient or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

steps [int] Number of iterations to run.

class `foolbox.attacks.`**`CarliniWagnerL2Attack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The L2 version of the Carlini & Wagner attack.

This attack is described in [Rc2cb572b91c5-1]. This implementation is based on the reference implementation by Carlini [Rc2cb572b91c5-2]. For bounds (0, 1), it differs from [Rc2cb572b91c5-2] because we normalize the squared L2 loss with the bounds.

References

[Rc2cb572b91c5-1], [Rc2cb572b91c5-2]

`as_generator`(*self*, *a*, *binary_search_steps=5*, *max_iterations=1000*, *confidence=0*, *learning_rate=0.005*, *initial_const=0.01*, *abort_early=True*)

The L2 version of the Carlini & Wagner attack.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search_steps [int] The number of steps for the binary search used to find the optimal tradeoff-constant between distance and confidence.

max_iterations [int] The maximum number of iterations. Larger values are more accurate; setting it too small will require a large learning rate and will produce poor results.

confidence [int or float] Confidence of adversarial examples: a higher value produces adversarials that are further away, but more strongly classified as adversarial.

learning_rate [float] The learning rate for the attack algorithm. Smaller values produce better results but take longer to converge.

initial_const [float] The initial tradeoff-constant to use to tune the relative importance of distance and confidence. If *binary_search_steps* is large, the initial constant is not important.

abort_early [bool] If True, Adam will be aborted if the loss hasn't decreased for some time (a tenth of max_iterations).

`static best_other_class`(*logits*, *exclude*)
Returns the index of the largest logit, ignoring the class that is passed as *exclude*.

`classmethod loss_function`(*const*, *a*, *x*, *logits*, *reconstructed_original*, *confidence*, *min_*, *max_*)
Returns the loss and the gradient of the loss w.r.t. x, assuming that logits = model(x).

`class foolbox.attacks.EADAttack`(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Gradient based attack which uses an elastic-net regularization [1]. This implementation is based on the attacks description [1] and its reference implementation [2].

References

[Rf0e4124daa63-1], [Rf0e4124daa63-2]

`as_generator`(*self*, *a*, *binary_search_steps=5*, *max_iterations=1000*, *confidence=0*, *initial_learning_rate=0.01*, *regularization=0.01*, *initial_const=0.01*, *abort_early=True*)
The L2 version of the Carlini & Wagner attack.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search_steps [int] The number of steps for the binary search used to find the optimal tradeoff-constant between distance and confidence.

max_iterations [int] The maximum number of iterations. Larger values are more accurate; setting it too small will require a large learning rate and will produce poor results.

confidence [int or float] Confidence of adversarial examples: a higher value produces adversarials that are further away, but more strongly classified as adversarial.

initial_learning_rate [float] The initial learning rate for the attack algorithm. Smaller values produce better results but take longer to converge. During the attack a square-root decay in the learning rate is performed.

initial_const [float] The initial tradeoff-constant to use to tune the relative importance of distance and confidence. If *binary_search_steps* is large, the initial constant is not important.

regularization [float] The L1 regularization parameter (also called beta). A value of *0* corresponds to the `attacks.CarliniWagnerL2Attack` attack.

abort_early [bool] If True, Adam will be aborted if the loss hasn't decreased for some time (a tenth of max_iterations).

static best_other_class(*logits*, *exclude*)
Returns the index of the largest logit, ignoring the class that is passed as *exclude*.

classmethod loss_function(*const*, *a*, *x*, *logits*, *reconstructed_original*, *confidence*, *min_*, *max_*)
Returns the loss and the gradient of the loss w.r.t. x, assuming that logits = model(x).

classmethod project_shrinkage_thresholding(*z*, *x0*, *regularization*, *min_*, *max_*)
Performs the element-wise projected shrinkage-thresholding operation

class foolbox.attacks.**DecoupledDirectionNormL2Attack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
The Decoupled Direction and Norm L2 adversarial attack from [R0e9d4da0ab48-1].

References

Robert Sabourin, Eric Granger, "Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses", https://arxiv.org/abs/1811.09600

[R0e9d4da0ab48-1]

as_generator(*self*, *a*, *steps=100*, *gamma=0.05*, *initial_norm=1*, *quantize=True*, *levels=256*)
The Decoupled Direction and Norm L2 adversarial attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

steps [int] Number of steps for the optimization.

gamma [float, optional] Factor by which the norm will be modified. new_norm = norm * (1 + or - gamma).

init_norm [float, optional] Initial value for the norm.

quantize [bool, optional] If True, the returned adversarials will have quantized values to the specified number of levels.

levels [int, optional] Number of levels to use for quantization (e.g. 256 for 8 bit images).

class foolbox.attacks.**SparseL1BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Sparse version of the Basic Iterative Method that minimizes the L1 distance introduced in [R0591d14da1c3-1].

References

See also:

L1BasicIterativeAttack

[R0591d14da1c3-1]

as_generator(*self*, *a*, *q=80.0*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Sparse version of a gradient-based attack that minimizes the L1 distance.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

q [float] Relative percentile to make gradients sparse (must be in [0, 100))

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**VirtualAdversarialAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Calculate an untargeted adversarial perturbation by performing a approximated second order optimization step on the KL divergence between the unperturbed predictions and the predictions for the adversarial perturbation. This attack was introduced in [Rc6516d158ac2-1].

References

[Rc6516d158ac2-1]

as_generator(*self*, *a*, *xi=1e-05*, *iterations=1*, *epsilons=1000*, *max_epsilon=0.3*)

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

xi [float] The finite difference size for performing the power method.

iterations [int] Number of iterations to perform power method to search for second order perturbation of KL divergence.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the direction of the sign of the gradient or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

12.2 Score-based attacks

class `foolbox.attacks.`**SinglePixelAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Perturbs just a single pixel and sets it to the min or max.

as_generator(*self*, *a*, *max_pixels=1000*)

Perturbs just a single pixel and sets it to the min or max.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

max_pixels [int] Maximum number of pixels to try.

class `foolbox.attacks.`**LocalSearchAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

A black-box attack based on the idea of greedy local search.

This implementation is based on the algorithm in [Rb320cee6998a-1].

References

[Rb320cee6998a-1]

as_generator (*self*, *a*, *r=1.5*, *p=10.0*, *d=5*, *t=5*, *R=150*)
A black-box attack based on the idea of greedy local search.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

r [float] Perturbation parameter that controls the cyclic perturbation; must be in [0, 2]

p [float] Perturbation parameter that controls the pixel sensitivity estimation

d [int] The half side length of the neighborhood square

t [int] The number of pixels perturbed at each round

R [int] An upper bound on the number of iterations

12.3 Decision-based attacks

class `foolbox.attacks.`**BoundaryAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

A powerful adversarial attack that requires neither gradients nor probabilities.

This is the reference implementation for the attack introduced in [Re72ca268aa55-1].

Notes

This implementation provides several advanced features:

- ability to continue previous attacks by passing an instance of the Adversarial class
- ability to pass an explicit starting point; especially to initialize a targeted attack
- ability to pass an alternative attack used for initialization
- fine-grained control over logging
- ability to specify the batch size
- optional automatic batch size tuning
- optional multithreading for random number generation
- optional multithreading for candidate point generation

References

[Re72ca268aa55-1]

as_generator(*self, a, iterations=5000, max_directions=25, starting_point=None, initialization_attack=None, log_every_n_steps=None, spherical_step=0.01, source_step=0.01, step_adaptation=1.5, batch_size=1, tune_batch_size=True, threaded_rnd=True, threaded_gen=True, alternative_generator=False, internal_dtype=<Mock name='mock.float64' id='140620647442920'>, loggingLevel=30*)

Applies the Boundary Attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

iterations [int] Maximum number of iterations to run. Might converge and stop before that.

max_directions [int] Maximum number of trials per ieration.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point, in particular for targeted attacks.

initialization_attack [`Attack`] Attack to use to find a starting point. Defaults to BlendedUniformNoiseAttack.

log_every_n_steps [int] Determines verbositity of the logging.

spherical_step [float] Initial step size for the orthogonal (spherical) step.

source_step [float] Initial step size for the step towards the target.

step_adaptation [float] Factor by which the step sizes are multiplied or divided.

batch_size [int] Batch size or initial batch size if tune_batch_size is True

tune_batch_size [bool] Whether or not the batch size should be automatically chosen between 1 and max_directions.

threaded_rnd [bool] Whether the random number generation should be multithreaded.

threaded_gen [bool] Whether the candidate point generation should be multithreaded.

alternative_generator: bool Whether an alternative implemenation of the candidate generator should be used.

internal_dtype [np.float32 or np.float64] Higher precision might be slower but is numerically more stable.

loggingLevel [int] Controls the verbosity of the logging, e.g. logging.INFO or logging.WARNING.

class foolbox.attacks.**SpatialAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Adversarially chosen rotations and translations [1].

This implementation is based on the reference implementation by Madry et al.: https://github.com/MadryLab/adversarial_spatial

References

[Rdffd25498f9d-1]

as_generator(*self*, *a*, *do_rotations=True*, *do_translations=True*, *x_shift_limits=(-5, 5)*, *y_shift_limits=(-5, 5)*, *angular_limits=(-5, 5)*, *granularity=10*, *random_sampling=False*, *abort_early=True*)

Adversarially chosen rotations and translations.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

do_rotations [bool] If False no rotations will be applied to the image.

do_translations [bool] If False no translations will be applied to the image.

x_shift_limits [int or (int, int)] Limits for horizontal translations in pixels. If one integer is provided the limits will be (-x_shift_limits, x_shift_limits).

y_shift_limits [int or (int, int)] Limits for vertical translations in pixels. If one integer is provided the limits will be (-y_shift_limits, y_shift_limits).

angular_limits [int or (int, int)] Limits for rotations in degrees. If one integer is provided the limits will be [-angular_limits, angular_limits].

granularity [int] Density of sampling within limits for each dimension.

random_sampling [bool] If True we sample translations/rotations randomly within limits, otherwise we use a regular grid.

abort_early [bool] If True, the attack stops as soon as it finds an adversarial.

class foolbox.attacks.**PointwiseAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Starts with an adversarial and performs a binary search between the adversarial and the original for each dimension of the input individually.

References

[R739f80a24875-1]

as_generator(*self*, *a*, *starting_point=None*, *initialization_attack=None*)

Starts with an adversarial and performs a binary search between the adversarial and the original for each dimension of the input individually.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point, in particular for targeted attacks.

initialization_attack [`Attack`] Attack to use to find a starting point. Defaults to SaltAndPepperNoiseAttack.

class `foolbox.attacks.`**`GaussianBlurAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Blurs the input until it is misclassified.

`as_generator`(*self*, *a*, *epsilons=1000*)

Blurs the input until it is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if input is a *numpy.ndarray*, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of standard deviations of the Gaussian blur or number of standard deviations between 0 and 1 that should be tried.

class `foolbox.attacks.`**`ContrastReductionAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Reduces the contrast of the input until it is misclassified.

`as_generator`(*self*, *a*, *epsilons=1000*)

Reduces the contrast of the input until it is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of contrast levels or number of contrast levels between 1 and 0 that should be tried. Epsilons are one minus the contrast level.

class foolbox.attacks.**AdditiveUniformNoiseAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Adds uniform noise to the input, gradually increasing the standard deviation until the input is misclassified.

__call__(*self*, *inputs*, *labels*, *unpack=True*, *individual_kwargs=None*, ***kwargs*)
Call self as a function.

__class__
alias of `abc.ABCMeta`

__delattr__(*self*, *name*, */*)
Implement delattr(self, name).

__dir__()
default dir() implementation

__eq__(*self*, *value*, */*)
Return self==value.

__format__()
default object formatter

__ge__(*self*, *value*, */*)
Return self>=value.

__getattribute__(*self*, *name*, */*)
Return getattr(self, name).

__gt__(*self*, *value*, */*)
Return self>value.

__hash__(*self*, */*)
Return hash(self).

__init__(*self*, *model=None*, *criterion=<foolbox.criteria.Misclassification object at 0x7fe4c68a1b70>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Initialize self. See help(type(self)) for accurate signature.

__le__(*self*, *value*, */*)
Return self<=value.

__lt__(*self*, *value*, */*)
Return self<value.

__ne__(*self*, *value*, */*)
Return self!=value.

__new__(**args*, ***kwargs*)
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__(*self*, */*)
Return repr(self).

__setattr__(*self*, *name*, *value*, /)
Implement setattr(self, name, value).

__sizeof__()
size of object in memory, in bytes

__str__(*self*, /)
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

as_generator(*self*, *a*, *epsilons=1000*)
Adds uniform or Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of noise levels or number of noise levels between 0 and 1 that should be tried.

name(*self*)
Returns a human readable name that uniquely identifies the attack with its hyperparameters.

Returns

str Human readable name that uniquely identifies the attack with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.attacks.**AdditiveGaussianNoiseAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Adds Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.

__call__(*self*, *inputs*, *labels*, *unpack=True*, *individual_kwargs=None*, ***kwargs*)
Call self as a function.

__class__
alias of `abc.ABCMeta`

__delattr__(*self*, *name*, /)
Implement delattr(self, name).

__dir__()
default dir() implementation

__eq__(*self*, *value*, /)
Return self==value.

__format__()
default object formatter

__ge__(*self*, *value*, /)
Return self>=value.

__getattribute__(*self*, *name*, /)
Return getattr(self, name).

__gt__(*self*, *value*, /)
Return self>value.

__hash__(*self*, /)
Return hash(self).

__init__(*self*, *model=None*, *criterion=<foolbox.criteria.Misclassification object at 0x7fe4c68a1b70>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Initialize self. See help(type(self)) for accurate signature.

__le__(*self*, *value*, /)
Return self<=value.

__lt__(*self*, *value*, /)
Return self<value.

__ne__(*self*, *value*, /)
Return self!=value.

__new__(**args*, ***kwargs*)
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__(*self*, /)
Return repr(self).

__setattr__(*self*, *name*, *value*, /)
Implement setattr(self, name, value).

__sizeof__()
size of object in memory, in bytes

__str__(*self*, /)
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

as_generator (*self*, *a*, *epsilons=1000*)
Adds uniform or Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of noise levels or number of noise levels between 0 and 1 that should be tried.

name (*self*)
Returns a human readable name that uniquely identifies the attack with its hyperparameters.

Returns

str Human readable name that uniquely identifies the attack with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.attacks.**SaltAndPepperNoiseAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Increases the amount of salt and pepper noise until the input is misclassified.

as_generator (*self*, *a*, *epsilons=100*, *repetitions=10*)
Increases the amount of salt and pepper noise until the input is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int] Number of steps to try between probability 0 and 1.

repetitions [int] Specifies how often the attack will be repeated.

class foolbox.attacks.**BlendedUniformNoiseAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Blends the input with a uniform noise input until it is misclassified.

as_generator(*self*, *a*, *epsilons=1000*, *max_directions=1000*)

Blends the input with a uniform noise input until it is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of blending steps or number of blending steps between 0 and 1 that should be tried.

max_directions [int] Maximum number of random inputs to try.

class foolbox.attacks.**HopSkipJumpAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

A powerful adversarial attack that requires neither gradients nor probabilities.

Notes

Features: * ability to switch between two types of distances: MSE and Linf. * ability to continue previous attacks by passing an instance of the

Adversarial class

- ability to pass an explicit starting point; especially to initialize a targeted attack
- ability to pass an alternative attack used for initialization
- ability to specify the batch size

References

HopSkipJumpAttack was originally proposed by Chen, Jordan and Wainwright. It is a decision-based attack that requires access to output labels of a model alone. Paper link: https://arxiv.org/abs/1904.02144 The implementation in Foolbox is based on Boundary Attack.

approximate_gradient(*self*, *decision_function*, *sample*, *num_evals*, *delta*)

Gradient direction estimation

as_generator(*self*, *a*, *iterations=64*, *initial_num_evals=100*, *max_num_evals=10000*, *stepsize_search='geometric_progression'*, *gamma=1.0*, *starting_point=None*, *batch_size=256*, *internal_dtype=<Mock name='mock.float64' id='140620647442920'>*, *log_every_n_steps=None*, *loggingLevel=30*)

Applies HopSkipJumpAttack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

iterations [int] Number of iterations to run.

initial_num_evals: int Initial number of evaluations for gradient estimation. Larger initial_num_evals increases time efficiency, but may decrease query efficiency.

max_num_evals: int Maximum number of evaluations for gradient estimation.

stepsize_search: str How to search for stepsize; choices are 'geometric_progression', 'grid_search'. 'geometric progression' initializes the stepsize by ||x_t - x||_p / sqrt(iteration), and keep decreasing by half until reaching the target side of the boundary. 'grid_search' chooses the optimal epsilon over a grid, in the scale of ||x_t - x||_p.

gamma: float

The binary search threshold theta is gamma / d^1.5 for l2 attack and gamma / d^2 for linf attack.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point, required for targeted attacks.

batch_size [int] Batch size for model prediction.

internal_dtype [np.float32 or np.float64] Higher precision might be slower but is numerically more stable.

log_every_n_steps [int] Determines verbositity of the logging.

loggingLevel [int] Controls the verbosity of the logging, e.g. logging.INFO or logging.WARNING.

`attack`(*self*, *a*, *iterations*)

iterations [int] Maximum number of iterations to run.

`binary_search_batch`(*self*, *unperturbed*, *perturbed_inputs*, *decision_function*)
Binary search to approach the boundary.

`geometric_progression_for_stepsize`(*self*, *x*, *update*, *dist*, *decision_function*, *current_iteration*)
Geometric progression to search for stepsize. Keep decreasing stepsize by half until reaching the desired side of the boundary.

`project`(*self*, *unperturbed*, *perturbed_inputs*, *alphas*)
Projection onto given l2 / linf balls in a batch.

`select_delta`(*self*, *dist_post_update*, *current_iteration*)
Choose the delta at the scale of distance between x and perturbed sample.

class `foolbox.attacks.`**`GenAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
The GenAttack introduced in [R996613153a1e-1].

This attack is performs a genetic search in order to find an adversarial perturbation in a black-box scenario in as few queries as possible.

References

[R996613153a1e-1]

as_generator(*self*, *a*, *generations=10*, *alpha=1.0*, *p=0.05*, *N=10*, *tau=0.1*, *search_shape=None*, *epsilon=0.3*, *binary_search=20*)

A black-box attack based on genetic algorithms. Can either try to find an adversarial perturbation for a fixed epsilon distance or perform a binary search over epsilon values in order to find a minimal perturbation. Parameters ———- inputs : *numpy.ndarray*

Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

generations [int] Number of generations, i.e. iterations, in the genetic algorithm.

alpha [float] Mutation-range.

p [float] Mutation probability.

N [int] Population size of the genetic algorithm.

tau: float Temperature for the softmax sampling used to determine the parents of the new crossover.

search_shape [tuple (default: None)] Set this to a smaller image shape than the true shape to search in a smaller input space. The input will be scaled using a linear interpolation to match the required input shape of the model.

binary_search [bool or int] Whether to perform a binary search over epsilon and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

12.4 Other attacks

class foolbox.attacks.**BinarizationRefinementAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

For models that preprocess their inputs by binarizing the inputs, this attack can improve adversarials found by other attacks. It does os by utilizing information about the binarization and mapping values to the corresponding value in the clean input or to the right side of the threshold.

as_generator(*self*, *a*, *starting_point=None*, *threshold=None*, *included_in='upper'*)

For models that preprocess their inputs by binarizing the inputs, this attack can improve adversarials found by other attacks. It does this by utilizing information about the binarization and mapping values to the corresponding value in the clean input or to the right side of the threshold.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point.

threshold [float] The treshold used by the models binarization. If none, defaults to (model.bounds()[1] - model.bounds()[0]) / 2.

included_in [str] Whether the threshold value itself belongs to the lower or upper interval.

class `foolbox.attacks.`**`PrecomputedAdversarialsAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Attacks a model using precomputed adversarial candidates.

`as_generator`(*self*, *a*, *candidate_inputs*, *candidate_outputs*)

Attacks a model using precomputed adversarial candidates.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

candidate_inputs [*numpy.ndarray*] The original inputs that will be expected by this attack.

candidate_outputs [*numpy.ndarray*] The adversarial candidates corresponding to the inputs.

class `foolbox.attacks.`**`InversionAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Creates "negative images" by inverting the pixel values according to [R57cf8375f1ff-1].

References

[R57cf8375f1ff-1]

`as_generator`(*self*, *a*)

Creates "negative images" by inverting the pixel values.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

Gradient-based attacks

GradientAttack	Perturbs the input with the gradient of the loss w.r.t.
GradientSignAttack	Adds the sign of the gradient to the input, gradually increasing the magnitude until the input is misclassified.
FGSM	alias of `foolbox.attacks.gradient.GradientSignAttack`
LinfinityBasicIterativeAttack	The Basic Iterative Method introduced in [R37dbc8f24aee-1].
BasicIterativeMethod	alias of `foolbox.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`
BIM	alias of `foolbox.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`
L1BasicIterativeAttack	Modified version of the Basic Iterative Method that minimizes the L1 distance.
L2BasicIterativeAttack	Modified version of the Basic Iterative Method that minimizes the L2 distance.
ProjectedGradientDescentAttack	The Projected Gradient Descent Attack introduced in [R367e8e10528a-1] without random start.
ProjectedGradientDescent	alias of `foolbox.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`
PGD	alias of `foolbox.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`
RandomStartProjectedGradientDescentAttack	The Projected Gradient Descent Attack introduced in [Re6066bc39e14-1] with random start.
RandomProjectedGradientDescent	alias of `foolbox.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`
RandomPGD	alias of `foolbox.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`
AdamL1BasicIterativeAttack	Modified version of the Basic Iterative Method that minimizes the L1 distance using the Adam optimizer.
AdamL2BasicIterativeAttack	Modified version of the Basic Iterative Method that minimizes the L2 distance using the Adam optimizer.
AdamProjectedGradientDescentAttack	The Projected Gradient Descent Attack introduced in [Re2d4f39a0205-1], [Re2d4f39a0205-2] without random start using the Adam optimizer.
AdamProjectedGradientDescent	alias of `foolbox.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`
AdamPGD	alias of `foolbox.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`
AdamRandomStartProjectedGradientDescentAttack	The Projected Gradient Descent Attack introduced in [R3210aa339085-1], [R3210aa339085-2] with random start using the Adam optimizer.

Continued on next page

Table 1 – continued from previous page

AdamRandomProjectedGradientDescent	alias of foolbox.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack
AdamRandomPGD	alias of foolbox.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack
MomentumIterativeAttack	The Momentum Iterative Method attack introduced in [R86d363e1fb2f-1].
MomentumIterativeMethod	alias of foolbox.attacks.iterative_projected_gradient.MomentumIterativeAttack
LBFGSAttack	
DeepFoolAttack	Simple and close to optimal gradient-based adversarial attack.
NewtonFoolAttack	Implements the NewtonFool Attack.
DeepFoolL2Attack	
DeepFoolLinfinityAttack	
ADefAttack	Adversarial attack that distorts the image, i.e.
SLSQPAttack	
SaliencyMapAttack	Implements the Saliency Map Attack.
IterativeGradientAttack	Like GradientAttack but with several steps for each epsilon.
IterativeGradientSignAttack	Like GradientSignAttack but with several steps for each epsilon.
CarliniWagnerL2Attack	The L2 version of the Carlini & Wagner attack.
EADAttack	Gradient based attack which uses an elastic-net regularization [1].
DecoupledDirectionNormL2Attack	The Decoupled Direction and Norm L2 adversarial attack from [R0e9d4da0ab48-1].
SparseFoolAttack	
SparseL1BasicIterativeAttack	Sparse version of the Basic Iterative Method that minimizes the L1 distance introduced in [R0591d14da1c3-1].
VirtualAdversarialAttack	Calculate an untargeted adversarial perturbation by performing a approximated second order optimization step on the KL divergence between the unperturbed predictions and the predictions for the adversarial perturbation.

Score-based attacks

SinglePixelAttack	Perturbs just a single pixel and sets it to the min or max.
LocalSearchAttack	A black-box attack based on the idea of greedy local search.
ApproximateLBFGSAttack	

Decision-based attacks

BoundaryAttack	A powerful adversarial attack that requires neither gradients nor probabilities.
SpatialAttack	Adversarially chosen rotations and translations [1].
PointwiseAttack	Starts with an adversarial and performs a binary search between the adversarial and the original for each dimension of the input individually.
GaussianBlurAttack	Blurs the input until it is misclassified.
ContrastReductionAttack	Reduces the contrast of the input until it is misclassified.
AdditiveUniformNoiseAttack	Adds uniform noise to the input, gradually increasing the standard deviation until the input is misclassified.
AdditiveGaussianNoiseAttack	Adds Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.
SaltAndPepperNoiseAttack	Increases the amount of salt and pepper noise until the input is misclassified.
BlendedUniformNoiseAttack	Blends the input with a uniform noise input until it is misclassified.
BoundaryAttackPlusPlus	
GenAttack	The GenAttack introduced in [R996613153a1e-1].
HopSkipJumpAttack	A powerful adversarial attack that requires neither gradients nor probabilities.

Other attacks

BinarizationRefinementAttack	For models that preprocess their inputs by binarizing the inputs, this attack can improve adversarials found by other attacks.
PrecomputedAdversarialsAttack	Attacks a model using precomputed adversarial candidates.
InversionAttack	Creates "negative images" by inverting the pixel values according to [R57cf8375f1ff-1].

CHAPTER 13

foolbox.adversarial

Provides a class that represents an adversarial example.

class foolbox.adversarial.**Adversarial**(*model*, *criterion*, *unperturbed*, *original_class*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*, *verbose=False*)

adversarial_class
The argmax of the model predictions for the best adversarial found so far.

None if no adversarial has been found.

backward_one(*self*, *gradient*, *x=None*, *strict=True*)
Interface to model.backward_one for attacks.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits.

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

Returns

gradient [*numpy.ndarray*] The gradient w.r.t the input.

See also:

gradient()

channel_axis(*self*, *batch*)
Interface to model.channel_axis for attacks.

Parameters

batch [bool] Controls whether the index of the axis for a batch of inputs (4 dimensions) or a single input (3 dimensions) should be returned.

distance
The distance of the adversarial input to the original input.

forward (*self*, *inputs*, *greedy=False*, *strict=True*, *return_details=False*)
Interface to model.forward for attacks.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the model.

greedy [bool] Whether the first adversarial should be returned.

strict [bool] Controls if the bounds for the pixel values should be checked.

forward_and_gradient (*self*, *x*, *label=None*, *strict=True*, *return_details=False*)
Interface to model.forward_and_gradient_one for attacks.

Parameters

x [*numpy.ndarray*] Multiple input with shape as expected by the model (with the batch dimension).

label [*numpy.ndarray*] Labels used to calculate the loss that is differentiated. Defaults to the original label.

strict [bool] Controls if the bounds for the pixel values should be checked.

forward_and_gradient_one (*self*, *x=None*, *label=None*, *strict=True*, *return_details=False*)
Interface to model.forward_and_gradient_one for attacks.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension). Defaults to the original input.

label [int] Label used to calculate the loss that is differentiated. Defaults to the original label.

strict [bool] Controls if the bounds for the pixel values should be checked.

forward_one (*self*, *x*, *strict=True*, *return_details=False*)
Interface to model.forward_one for attacks.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

strict [bool] Controls if the bounds for the pixel values should be checked.

gradient_one (*self*, *x=None*, *label=None*, *strict=True*)
Interface to model.gradient_one for attacks.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension). Defaults to the original input.

label [int] Label used to calculate the loss that is differentiated. Defaults to the original label.

strict [bool] Controls if the bounds for the pixel values should be checked.

has_gradient (*self*)
Returns true if _backward and _forward_backward can be called by an attack, False otherwise.

normalized_distance (*self*, *x*)
Calculates the distance of a given input x to the original input.

Parameters

x [*numpy.ndarray*] The input x that should be compared to the original input.

Returns

Distance The distance between the given input and the original input.

original_class
The class of the original input (ground-truth, not model prediction).

output
The model predictions for the best adversarial found so far.

None if no adversarial has been found.

perturbed
The best adversarial example found so far.

reached_threshold (*self*)
Returns True if a threshold is given and the currently best adversarial distance is smaller than the threshold.

target_class
Interface to criterion.target_class for attacks.

unperturbed
The original input.

CHAPTER 14

foolbox.utils

foolbox.utils.**softmax**(*logits*)
Transforms predictions into probability values.

Parameters

logits [array_like] The logits predicted by the model.

Returns

numpy.ndarray Probability values corresponding to the logits.

foolbox.utils.**crossentropy**(*label*, *logits*)
Calculates the cross-entropy.

Parameters

logits [array_like] The logits predicted by the model.

label [int] The label describing the target distribution.

Returns

float The cross-entropy between softmax(logits) and onehot(label).

foolbox.utils.**batch_crossentropy**(*label*, *logits*)
Calculates the cross-entropy for a batch of logits.

Parameters

logits [array_like] The logits predicted by the model for a batch of inputs.

label [int] The label describing the target distribution.

Returns

np.ndarray The cross-entropy between softmax(logits[i]) and onehot(label) for all i.

foolbox.utils.**binarize**(*x*, *values*, *threshold=None*, *included_in='upper'*)
Binarizes the values of x.

Parameters

values [tuple of two floats] The lower and upper value to which the inputs are mapped.

threshold [float] The threshold; defaults to (values[0] + values[1]) / 2 if None.

included_in [str] Whether the threshold value itself belongs to the lower or upper interval.

`foolbox.utils.`**`imagenet_example`**(*shape=(224, 224)*, *data_format='channels_last'*, *bounds=(0, 255)*)

Returns an example image and its imagenet class label.

Parameters

shape [list of integers] The shape of the returned image.

data_format [str] "channels_first" or "channels_last"

bounds [tuple] smallest and largest allowed pixel value

Returns

image [array_like] The example image.

label [int] The imagenet label associated with the image.

NOTE: This function is deprecated and will be removed in the future.

`foolbox.utils.`**`samples`**(*dataset='imagenet'*, *index=0*, *batchsize=1*, *shape=(224, 224)*, *data_format='channels_last'*, *bounds=(0, 255)*)

Returns a batch of example images and the corresponding labels

Parameters

dataset [string] The data set to load (options: imagenet, mnist, cifar10, cifar100, fashionMNIST)

index [int] For each data set 20 example images exist. The returned batch contains the images with index [index, index + 1, index + 2, . . .]

batchsize [int] Size of batch.

shape [list of integers] The shape of the returned image (only relevant for Imagenet).

data_format [str] "channels_first" or "channels_last"

bounds [tuple] smallest and largest allowed pixel value

Returns

images [array_like] The batch of example images

labels [array of int] The labels associated with the images.

`foolbox.utils.`**`onehot_like`**(*a*, *index*, *value=1*)

Creates an array like a, with all values set to 0 except one.

Parameters

a [array_like] The returned one-hot array will have the same shape and dtype as this array

index [int] The index that should be set to *value*

value [single value compatible with a.dtype] The value to set at the given index

Returns

numpy.ndarray One-hot array with the given value at the given location and zeros everywhere else.

CHAPTER 15

foolbox.v1.attacks

15.1 Gradient-based attacks

class foolbox.attacks.**GradientAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Perturbs the input with the gradient of the loss w.r.t. the input, gradually increasing the magnitude until the input is misclassified.

Does not do anything if the model does not have a gradient.

as_generator(*self*, *a*, *epsilons=1000*, *max_epsilon=1*)

Perturbs the input with the gradient of the loss w.r.t. the input, gradually increasing the magnitude until the input is misclassified.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the gradient direction or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

class foolbox.attacks.**GradientSignAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Adds the sign of the gradient to the input, gradually increasing the magnitude until the input is misclassified.

This attack is often referred to as Fast Gradient Sign Method and was introduced in [R20d0064ee4c9-1].

Does not do anything if the model does not have a gradient.

References

[R20d0064ee4c9-1]

as_generator (*self*, *a*, *epsilons=1000*, *max_epsilon=1*)

Adds the sign of the gradient to the input, gradually increasing the magnitude until the input is misclassified.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the direction of the sign of the gradient or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

`foolbox.attacks.`**FGSM**

alias of `foolbox.attacks.gradient.GradientSignAttack`

class `foolbox.attacks.`**LinfinityBasicIterativeAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The Basic Iterative Method introduced in [R37dbc8f24aee-1].

This attack is also known as Projected Gradient Descent (PGD) (without random start) or FGMS^k.

References

See also:

ProjectedGradientDescentAttack

[R37dbc8f24aee-1]

as_generator (*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**`BasicIterativeMethod`**
alias of `foolbox.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`

`foolbox.attacks.`**`BIM`**
alias of `foolbox.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`

`class` `foolbox.attacks.`**`L1BasicIterativeAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Modified version of the Basic Iterative Method that minimizes the L1 distance.

See also:

`LinfinityBasicIterativeAttack`

`as_generator`(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)
Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**L2BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Modified version of the Basic Iterative Method that minimizes the L2 distance.

See also:

LinfinityBasicIterativeAttack

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**ProjectedGradientDescentAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The Projected Gradient Descent Attack introduced in [R367e8e10528a-1] without random start.

When used without a random start, this attack is also known as Basic Iterative Method (BIM) or FGSM^k.

References

See also:

LinfinityBasicIterativeAttack and *RandomStartProjectedGradientDescentAttack*

[R367e8e10528a-1]

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.01*, *iterations=40*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**`ProjectedGradientDescent`**

alias of `foolbox.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`

`foolbox.attacks.`**`PGD`**

alias of `foolbox.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`

class `foolbox.attacks.`**`RandomStartProjectedGradientDescentAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The Projected Gradient Descent Attack introduced in [Re6066bc39e14-1] with random start.

References

See also:

ProjectedGradientDescentAttack

[Re6066bc39e14-1]

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.01*, *iterations=40*, *random_start=True*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**RandomProjectedGradientDescent**

alias of `foolbox.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`

`foolbox.attacks.`**RandomPGD**

alias of `foolbox.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`

class `foolbox.attacks.`**AdamL1BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Modified version of the Basic Iterative Method that minimizes the L1 distance using the Adam optimizer.

See also:

LinfinityBasicIterativeAttack

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**AdamL2BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Modified version of the Basic Iterative Method that minimizes the L2 distance using the Adam optimizer.

See also:

LinfinityBasicIterativeAttack

as_generator(*self*, *a*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class `foolbox.attacks.`**`AdamProjectedGradientDescentAttack`**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

The Projected Gradient Descent Attack introduced in [Re2d4f39a0205-1], [Re2d4f39a0205-2] without random start using the Adam optimizer.

When used without a random start, this attack is also known as Basic Iterative Method (BIM) or FGSM^k.

References

See also:

`LinfinityBasicIterativeAttack` and *`RandomStartProjectedGradientDescentAttack`*

[Re2d4f39a0205-1], [Re2d4f39a0205-2]

`as_generator`(*self, a, binary_search=True, epsilon=0.3, stepsize=0.01, iterations=40, random_start=False, return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.`**`AdamProjectedGradientDescent`**
alias of `foolbox.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`

`foolbox.attacks.`**`AdamPGD`**
alias of `foolbox.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`

`class` `foolbox.attacks.`**`AdamRandomStartProjectedGradientDescentAttack`**(*model=None, criterion=<foolbox.criteria.Misclassi object>, distance=<class 'foolbox.distances.MeanSquaredDista threshold=None*)

The Projected Gradient Descent Attack introduced in [R3210aa339085-1], [R3210aa339085-2] with random start using the Adam optimizer.

References

See also:

`ProjectedGradientDescentAttack`

[R3210aa339085-1], [R3210aa339085-2]

`as_generator`(*self, a, binary_search=True, epsilon=0.3, stepsize=0.01, iterations=40, random_start=True, return_early=True*)

Simple iterative gradient-based attack known as Basic Iterative Method, Projected Gradient Descent or FGSM^k.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

`foolbox.attacks.AdamRandomProjectedGradientDescent`
alias of `foolbox.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack`

`foolbox.attacks.AdamRandomPGD`
alias of `foolbox.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack`

class `foolbox.attacks.MomentumIterativeAttack`(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

The Momentum Iterative Method attack introduced in [R86d363e1fb2f-1]. It's like the Basic Iterative Method or Projected Gradient Descent except that it uses momentum.

References

[R86d363e1fb2f-1]

`as_generator`(*self, a, binary_search=True, epsilon=0.3, stepsize=0.06, iterations=10, decay_factor=1.0, random_start=False, return_early=True*)

Momentum-based iterative gradient attack known as Momentum Iterative Method.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search [bool] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

decay_factor [float] Decay factor used by the momentum term.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

foolbox.attacks.**MomentumIterativeMethod**
alias of foolbox.attacks.iterative_projected_gradient.MomentumIterativeAttack

class foolbox.attacks.**DeepFoolAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Simple and close to optimal gradient-based adversarial attack.

Implementes DeepFool introduced in [Rb4dd02640756-1].

References

[Rb4dd02640756-1]

as_generator(*self*, *a*, *steps=100*, *subsample=10*, *p=None*)
Simple and close to optimal gradient-based adversarial attack.

Parameters

input_or_adv [*numpy.ndarray* or Adversarial] The original, unperturbed input as a *numpy.ndarray* or an Adversarial instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an Adversarial instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

steps [int] Maximum number of steps to perform.

subsample [int] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster.

p [int or float] Lp-norm that should be minimzed, must be 2 or np.inf.

class foolbox.attacks.**NewtonFoolAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Implements the NewtonFool Attack.

The attack was introduced in [R6a972939b320-1].

References

[R6a972939b320-1]

as_generator(*self*, *a*, *max_iter=100*, *eta=0.01*)

Parameters

input_or_adv [*numpy.ndarray* or Adversarial] The original, unperturbed input as a *numpy.ndarray* or an Adversarial instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an Adversarial instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

max_iter [int] The maximum number of iterations.

eta [float] the eta coefficient

class foolbox.attacks.**DeepFoolL2Attack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

as_generator(*self*, *a*, *steps=100*, *subsample=10*)
Simple and close to optimal gradient-based adversarial attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

steps [int] Maximum number of steps to perform.

subsample [int] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster.

p [int or float] Lp-norm that should be minimzed, must be 2 or np.inf.

class foolbox.attacks.**DeepFoolLinfinityAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

as_generator(*self*, *a*, *steps=100*, *subsample=10*)
Simple and close to optimal gradient-based adversarial attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

steps [int] Maximum number of steps to perform.

subsample [int] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster.

p [int or float] Lp-norm that should be minimzed, must be 2 or np.inf.

class foolbox.attacks.**ADefAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Adversarial attack that distorts the image, i.e. changes the locations of pixels. The algorithm is described in [Rf241e6d2664d-1], a Repository with the original code can be found in [Rf241e6d2664d-2]. References ——— .. [Rf241e6d2664d-1] Rima Alaifari, Giovanni S. Alberti, and Tandri Gauksson:

"ADef: an Iterative Algorithm to Construct Adversarial Deformations", https://arxiv.org/abs/1804.07729

as_generator(*self*, *a*, *max_iter=100*, *smooth=1.0*, *subsample=10*)

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

max_iter [int > 0] Maximum number of iterations (default max_iter = 100).

smooth [float >= 0] Width of the Gaussian kernel used for smoothing. (default is smooth = 0 for no smoothing).

subsample [int >= 2] Limit on the number of the most likely classes that should be considered. A small value is usually sufficient and much faster. (default subsample = 10)

class foolbox.attacks.**SaliencyMapAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Implements the Saliency Map Attack.

The attack was introduced in [R08e06ca693ba-1].

References

[R08e06ca693ba-1]

as_generator(*self, a, max_iter=2000, num_random_targets=0, fast=True, theta=0.1, max_perturbations_per_pixel=7*)

Implements the Saliency Map Attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

max_iter [int] The maximum number of iterations to run.

num_random_targets [int] Number of random target classes if no target class is given by the criterion.

fast [bool] Whether to use the fast saliency map calculation.

theta [float] perturbation per pixel relative to [min, max] range.

max_perturbations_per_pixel [int] Maximum number of times a pixel can be modified.

class foolbox.attacks.**IterativeGradientAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Like GradientAttack but with several steps for each epsilon.

as_generator(*self, a, epsilons=100, max_epsilon=1, steps=10*)

Like GradientAttack but with several steps for each epsilon.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the gradient direction or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

steps [int] Number of iterations to run.

class `foolbox.attacks.`**`IterativeGradientSignAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Like GradientSignAttack but with several steps for each epsilon.

`as_generator`(*self*, *a*, *epsilons=100*, *max_epsilon=1*, *steps=10*)

Like GradientSignAttack but with several steps for each epsilon.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the direction of the sign of the gradient or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

steps [int] Number of iterations to run.

class `foolbox.attacks.`**`CarliniWagnerL2Attack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

The L2 version of the Carlini & Wagner attack.

This attack is described in [Rc2cb572b91c5-1]. This implementation is based on the reference implementation by Carlini [Rc2cb572b91c5-2]. For bounds (0, 1), it differs from [Rc2cb572b91c5-2] because we normalize the squared L2 loss with the bounds.

References

[Rc2cb572b91c5-1], [Rc2cb572b91c5-2]

`as_generator`(*self*, *a*, *binary_search_steps=5*, *max_iterations=1000*, *confidence=0*, *learning_rate=0.005*, *initial_const=0.01*, *abort_early=True*)

The L2 version of the Carlini & Wagner attack.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search_steps [int] The number of steps for the binary search used to find the optimal tradeoff-constant between distance and confidence.

max_iterations [int] The maximum number of iterations. Larger values are more accurate; setting it too small will require a large learning rate and will produce poor results.

confidence [int or float] Confidence of adversarial examples: a higher value produces adversarials that are further away, but more strongly classified as adversarial.

learning_rate [float] The learning rate for the attack algorithm. Smaller values produce better results but take longer to converge.

initial_const [float] The initial tradeoff-constant to use to tune the relative importance of distance and confidence. If *binary_search_steps* is large, the initial constant is not important.

abort_early [bool] If True, Adam will be aborted if the loss hasn't decreased for some time (a tenth of max_iterations).

static best_other_class(*logits*, *exclude*)
Returns the index of the largest logit, ignoring the class that is passed as *exclude*.

classmethod loss_function(*const*, *a*, *x*, *logits*, *reconstructed_original*, *confidence*, *min_*, *max_*)
Returns the loss and the gradient of the loss w.r.t. x, assuming that logits = model(x).

class foolbox.attacks.**EADAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Gradient based attack which uses an elastic-net regularization [1]. This implementation is based on the attacks description [1] and its reference implementation [2].

References

[Rf0e4124daa63-1], [Rf0e4124daa63-2]

as_generator(*self*, *a*, *binary_search_steps=5*, *max_iterations=1000*, *confidence=0*, *initial_learning_rate=0.01*, *regularization=0.01*, *initial_const=0.01*, *abort_early=True*)
The L2 version of the Carlini & Wagner attack.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

binary_search_steps [int] The number of steps for the binary search used to find the optimal tradeoff-constant between distance and confidence.

max_iterations [int] The maximum number of iterations. Larger values are more accurate; setting it too small will require a large learning rate and will produce poor results.

confidence [int or float] Confidence of adversarial examples: a higher value produces adversarials that are further away, but more strongly classified as adversarial.

initial_learning_rate [float] The initial learning rate for the attack algorithm. Smaller values produce better results but take longer to converge. During the attack a square-root decay in the learning rate is performed.

initial_const [float] The initial tradeoff-constant to use to tune the relative importance of distance and confidence. If *binary_search_steps* is large, the initial constant is not important.

regularization [float] The L1 regularization parameter (also called beta). A value of *0* corresponds to the `attacks.CarliniWagnerL2Attack` attack.

abort_early [bool] If True, Adam will be aborted if the loss hasn't decreased for some time (a tenth of max_iterations).

static best_other_class(*logits*, *exclude*)
Returns the index of the largest logit, ignoring the class that is passed as *exclude*.

classmethod loss_function(*const*, *a*, *x*, *logits*, *reconstructed_original*, *confidence*, *min_*, *max_*)
Returns the loss and the gradient of the loss w.r.t. x, assuming that logits = model(x).

classmethod project_shrinkage_thresholding(*z*, *x0*, *regularization*, *min_*, *max_*)
Performs the element-wise projected shrinkage-thresholding operation

class foolbox.attacks.**DecoupledDirectionNormL2Attack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
The Decoupled Direction and Norm L2 adversarial attack from [R0e9d4da0ab48-1].

References

Robert Sabourin, Eric Granger, "Decoupling Direction and Norm for Efficient Gradient-Based L2 Adversarial Attacks and Defenses", https://arxiv.org/abs/1811.09600

[R0e9d4da0ab48-1]

as_generator(*self*, *a*, *steps=100*, *gamma=0.05*, *initial_norm=1*, *quantize=True*, *levels=256*)
The Decoupled Direction and Norm L2 adversarial attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

steps [int] Number of steps for the optimization.

gamma [float, optional] Factor by which the norm will be modified. new_norm = norm * (1 + or - gamma).

init_norm [float, optional] Initial value for the norm.

quantize [bool, optional] If True, the returned adversarials will have quantized values to the specified number of levels.

levels [int, optional] Number of levels to use for quantization (e.g. 256 for 8 bit images).

class foolbox.attacks.**SparseL1BasicIterativeAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Sparse version of the Basic Iterative Method that minimizes the L1 distance introduced in [R0591d14da1c3-1].

References

See also:

L1BasicIterativeAttack

[R0591d14da1c3-1]

as_generator(*self*, *a*, *q=80.0*, *binary_search=True*, *epsilon=0.3*, *stepsize=0.05*, *iterations=10*, *random_start=False*, *return_early=True*)

Sparse version of a gradient-based attack that minimizes the L1 distance.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

q [float] Relative percentile to make gradients sparse (must be in [0, 100))

binary_search [bool or int] Whether to perform a binary search over epsilon and stepsize, keeping their ratio constant and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

stepsize [float] Step size for gradient descent; if binary_search is True, this value is only for initialization and automatically adapted.

iterations [int] Number of iterations for each gradient descent run.

random_start [bool] Start the attack from a random point rather than from the original input.

return_early [bool] Whether an individual gradient descent run should stop as soon as an adversarial is found.

class foolbox.attacks.**VirtualAdversarialAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Calculate an untargeted adversarial perturbation by performing a approximated second order optimization step on the KL divergence between the unperturbed predictions and the predictions for the adversarial perturbation. This attack was introduced in [Rc6516d158ac2-1].

References

[Rc6516d158ac2-1]

as_generator (*self*, *a*, *xi=1e-05*, *iterations=1*, *epsilons=1000*, *max_epsilon=0.3*)

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

xi [float] The finite difference size for performing the power method.

iterations [int] Number of iterations to perform power method to search for second order perturbation of KL divergence.

epsilons [int or Iterable[float]] Either Iterable of step sizes in the direction of the sign of the gradient or number of step sizes between 0 and max_epsilon that should be tried.

max_epsilon [float] Largest step size if epsilons is not an iterable.

15.2 Score-based attacks

class foolbox.attacks.**SinglePixelAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Perturbs just a single pixel and sets it to the min or max.

as_generator (*self*, *a*, *max_pixels=1000*)

Perturbs just a single pixel and sets it to the min or max.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

max_pixels [int] Maximum number of pixels to try.

class foolbox.attacks.**LocalSearchAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

A black-box attack based on the idea of greedy local search.

This implementation is based on the algorithm in [Rb320cee6998a-1].

References

[Rb320cee6998a-1]

as_generator (*self*, *a*, *r=1.5*, *p=10.0*, *d=5*, *t=5*, *R=150*)
A black-box attack based on the idea of greedy local search.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

r [float] Perturbation parameter that controls the cyclic perturbation; must be in [0, 2]

p [float] Perturbation parameter that controls the pixel sensitivity estimation

d [int] The half side length of the neighborhood square

t [int] The number of pixels perturbed at each round

R [int] An upper bound on the number of iterations

15.3 Decision-based attacks

class foolbox.attacks.**BoundaryAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

A powerful adversarial attack that requires neither gradients nor probabilities.

This is the reference implementation for the attack introduced in [Re72ca268aa55-1].

Notes

This implementation provides several advanced features:

- ability to continue previous attacks by passing an instance of the Adversarial class
- ability to pass an explicit starting point; especially to initialize a targeted attack
- ability to pass an alternative attack used for initialization
- fine-grained control over logging
- ability to specify the batch size
- optional automatic batch size tuning
- optional multithreading for random number generation
- optional multithreading for candidate point generation

References

[Re72ca268aa55-1]

as_generator(*self, a, iterations=5000, max_directions=25, starting_point=None, initialization_attack=None, log_every_n_steps=None, spherical_step=0.01, source_step=0.01, step_adaptation=1.5, batch_size=1, tune_batch_size=True, threaded_rnd=True, threaded_gen=True, alternative_generator=False, internal_dtype=<Mock name='mock.float64' id='140620647442920'>, loggingLevel=30*)

Applies the Boundary Attack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

iterations [int] Maximum number of iterations to run. Might converge and stop before that.

max_directions [int] Maximum number of trials per ieration.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point, in particular for targeted attacks.

initialization_attack [`Attack`] Attack to use to find a starting point. Defaults to BlendedUniformNoiseAttack.

log_every_n_steps [int] Determines verbositity of the logging.

spherical_step [float] Initial step size for the orthogonal (spherical) step.

source_step [float] Initial step size for the step towards the target.

step_adaptation [float] Factor by which the step sizes are multiplied or divided.

batch_size [int] Batch size or initial batch size if tune_batch_size is True

tune_batch_size [bool] Whether or not the batch size should be automatically chosen between 1 and max_directions.

threaded_rnd [bool] Whether the random number generation should be multithreaded.

threaded_gen [bool] Whether the candidate point generation should be multithreaded.

alternative_generator: bool Whether an alternative implemenation of the candidate generator should be used.

internal_dtype [np.float32 or np.float64] Higher precision might be slower but is numerically more stable.

loggingLevel [int] Controls the verbosity of the logging, e.g. logging.INFO or logging.WARNING.

class foolbox.attacks.**SpatialAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Adversarially chosen rotations and translations [1].

This implementation is based on the reference implementation by Madry et al.: https://github.com/MadryLab/adversarial_spatial

References

[Rdffd25498f9d-1]

as_generator(*self*, *a*, *do_rotations=True*, *do_translations=True*, *x_shift_limits=(-5, 5)*, *y_shift_limits=(-5, 5)*, *angular_limits=(-5, 5)*, *granularity=10*, *random_sampling=False*, *abort_early=True*)

Adversarially chosen rotations and translations.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

do_rotations [bool] If False no rotations will be applied to the image.

do_translations [bool] If False no translations will be applied to the image.

x_shift_limits [int or (int, int)] Limits for horizontal translations in pixels. If one integer is provided the limits will be (-x_shift_limits, x_shift_limits).

y_shift_limits [int or (int, int)] Limits for vertical translations in pixels. If one integer is provided the limits will be (-y_shift_limits, y_shift_limits).

angular_limits [int or (int, int)] Limits for rotations in degrees. If one integer is provided the limits will be [-angular_limits, angular_limits].

granularity [int] Density of sampling within limits for each dimension.

random_sampling [bool] If True we sample translations/rotations randomly within limits, otherwise we use a regular grid.

abort_early [bool] If True, the attack stops as soon as it finds an adversarial.

class foolbox.attacks.**PointwiseAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Starts with an adversarial and performs a binary search between the adversarial and the original for each dimension of the input individually.

References

[R739f80a24875-1]

as_generator(*self*, *a*, *starting_point=None*, *initialization_attack=None*)

Starts with an adversarial and performs a binary search between the adversarial and the original for each dimension of the input individually.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point, in particular for targeted attacks.

initialization_attack [`Attack`] Attack to use to find a starting point. Defaults to SaltAndPepperNoiseAttack.

class foolbox.attacks.**GaussianBlurAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Blurs the input until it is misclassified.

as_generator(*self*, *a*, *epsilons=1000*)

Blurs the input until it is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if input is a *numpy.ndarray*, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of standard deviations of the Gaussian blur or number of standard deviations between 0 and 1 that should be tried.

class foolbox.attacks.**ContrastReductionAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Reduces the contrast of the input until it is misclassified.

as_generator(*self*, *a*, *epsilons=1000*)

Reduces the contrast of the input until it is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of contrast levels or number of contrast levels between 1 and 0 that should be tried. Epsilons are one minus the contrast level.

class foolbox.attacks.**AdditiveUniformNoiseAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Adds uniform noise to the input, gradually increasing the standard deviation until the input is misclassified.

`__call__`(*self, inputs, labels, unpack=True, individual_kwargs=None, **kwargs*)
Call self as a function.

`__class__`
alias of `abc.ABCMeta`

`__delattr__`(*self, name, /*)
Implement delattr(self, name).

`__dir__`()
default dir() implementation

`__eq__`(*self, value, /*)
Return self==value.

`__format__`()
default object formatter

`__ge__`(*self, value, /*)
Return self>=value.

`__getattribute__`(*self, name, /*)
Return getattr(self, name).

`__gt__`(*self, value, /*)
Return self>value.

`__hash__`(*self, /*)
Return hash(self).

`__init__`(*self, model=None, criterion=<foolbox.criteria.Misclassification object at 0x7fe4c68a1b70>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)
Initialize self. See help(type(self)) for accurate signature.

`__le__`(*self, value, /*)
Return self<=value.

`__lt__`(*self, value, /*)
Return self<value.

`__ne__`(*self, value, /*)
Return self!=value.

`__new__`(**args, **kwargs*)
Create and return a new object. See help(type) for accurate signature.

`__reduce__`()
helper for pickle

`__reduce_ex__`()
helper for pickle

`__repr__`(*self, /*)
Return repr(self).

__setattr__ (*self*, *name*, *value*, */*)
Implement setattr(self, name, value).

__sizeof__ ()
size of object in memory, in bytes

__str__ (*self*, */*)
Return str(self).

__subclasshook__ ()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

as_generator (*self*, *a*, *epsilons=1000*)
Adds uniform or Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of noise levels or number of noise levels between 0 and 1 that should be tried.

name (*self*)
Returns a human readable name that uniquely identifies the attack with its hyperparameters.

Returns

str Human readable name that uniquely identifies the attack with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.attacks.**AdditiveGaussianNoiseAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Adds Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.

__call__ (*self*, *inputs*, *labels*, *unpack=True*, *individual_kwargs=None*, ***kwargs*)
Call self as a function.

__class__
alias of `abc.ABCMeta`

__delattr__(*self*, *name*, /)
Implement delattr(self, name).

__dir__()
default dir() implementation

__eq__(*self*, *value*, /)
Return self==value.

__format__()
default object formatter

__ge__(*self*, *value*, /)
Return self>=value.

__getattribute__(*self*, *name*, /)
Return getattr(self, name).

__gt__(*self*, *value*, /)
Return self>value.

__hash__(*self*, /)
Return hash(self).

__init__(*self*, *model=None*, *criterion=<foolbox.criteria.Misclassification object at 0x7fe4c68a1b70>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Initialize self. See help(type(self)) for accurate signature.

__le__(*self*, *value*, /)
Return self<=value.

__lt__(*self*, *value*, /)
Return self<value.

__ne__(*self*, *value*, /)
Return self!=value.

__new__(**args*, ***kwargs*)
Create and return a new object. See help(type) for accurate signature.

__reduce__()
helper for pickle

__reduce_ex__()
helper for pickle

__repr__(*self*, /)
Return repr(self).

__setattr__(*self*, *name*, *value*, /)
Implement setattr(self, name, value).

__sizeof__()
size of object in memory, in bytes

__str__(*self*, /)
Return str(self).

__subclasshook__()
Abstract classes can override this to customize issubclass().

This is invoked early on by abc.ABCMeta.__subclasscheck__(). It should return True, False or NotImplemented. If it returns NotImplemented, the normal algorithm is used. Otherwise, it overrides the normal algorithm (and the outcome is cached).

__weakref__
list of weak references to the object (if defined)

as_generator (*self*, *a*, *epsilons=1000*)
Adds uniform or Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of noise levels or number of noise levels between 0 and 1 that should be tried.

name (*self*)
Returns a human readable name that uniquely identifies the attack with its hyperparameters.

Returns

str Human readable name that uniquely identifies the attack with its hyperparameters.

Notes

Defaults to the class name but subclasses can provide more descriptive names and must take hyperparameters into account.

class foolbox.attacks.**SaltAndPepperNoiseAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
Increases the amount of salt and pepper noise until the input is misclassified.

as_generator (*self*, *a*, *epsilons=100*, *repetitions=10*)
Increases the amount of salt and pepper noise until the input is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int] Number of steps to try between probability 0 and 1.

repetitions [int] Specifies how often the attack will be repeated.

class foolbox.attacks.**BlendedUniformNoiseAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

Blends the input with a uniform noise input until it is misclassified.

as_generator(*self, a, epsilons=1000, max_directions=1000*)

Blends the input with a uniform noise input until it is misclassified.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

epsilons [int or Iterable[float]] Either Iterable of blending steps or number of blending steps between 0 and 1 that should be tried.

max_directions [int] Maximum number of random inputs to try.

class foolbox.attacks.**HopSkipJumpAttack**(*model=None, criterion=<foolbox.criteria.Misclassification object>, distance=<class 'foolbox.distances.MeanSquaredDistance'>, threshold=None*)

A powerful adversarial attack that requires neither gradients nor probabilities.

Notes

Features: * ability to switch between two types of distances: MSE and Linf. * ability to continue previous attacks by passing an instance of the

Adversarial class

- ability to pass an explicit starting point; especially to initialize a targeted attack
- ability to pass an alternative attack used for initialization
- ability to specify the batch size

References

HopSkipJumpAttack was originally proposed by Chen, Jordan and Wainwright. It is a decision-based attack that requires access to output labels of a model alone. Paper link: https://arxiv.org/abs/1904.02144 The implementation in Foolbox is based on Boundary Attack.

approximate_gradient(*self, decision_function, sample, num_evals, delta*)

Gradient direction estimation

as_generator(*self, a, iterations=64, initial_num_evals=100, max_num_evals=10000, stepsize_search='geometric_progression', gamma=1.0, starting_point=None, batch_size=256, internal_dtype=<Mock name='mock.float64' id='140620647442920'>, log_every_n_steps=None, loggingLevel=30*)

Applies HopSkipJumpAttack.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, correctly classified input. If it is a numpy array, label must be passed as well. If it is an `Adversarial` instance, label must not be passed.

label [int] The reference label of the original input. Must be passed if input is a numpy array, must not be passed if input is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

iterations [int] Number of iterations to run.

initial_num_evals: int Initial number of evaluations for gradient estimation. Larger initial_num_evals increases time efficiency, but may decrease query efficiency.

max_num_evals: int Maximum number of evaluations for gradient estimation.

stepsize_search: str How to search for stepsize; choices are 'geometric_progression', 'grid_search'. 'geometric progression' initializes the stepsize by ||x_t - x||_p / sqrt(iteration), and keep decreasing by half until reaching the target side of the boundary. 'grid_search' chooses the optimal epsilon over a grid, in the scale of ||x_t - x||_p.

gamma: float

The binary search threshold theta is gamma / d^1.5 for l2 attack and gamma / d^2 for linf attack.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point, required for targeted attacks.

batch_size [int] Batch size for model prediction.

internal_dtype [np.float32 or np.float64] Higher precision might be slower but is numerically more stable.

log_every_n_steps [int] Determines verbositity of the logging.

loggingLevel [int] Controls the verbosity of the logging, e.g. logging.INFO or logging.WARNING.

`attack` (*self*, *a*, *iterations*)

iterations [int] Maximum number of iterations to run.

`binary_search_batch` (*self*, *unperturbed*, *perturbed_inputs*, *decision_function*)
Binary search to approach the boundary.

`geometric_progression_for_stepsize` (*self*, *x*, *update*, *dist*, *decision_function*, *current_iteration*)
Geometric progression to search for stepsize. Keep decreasing stepsize by half until reaching the desired side of the boundary.

`project` (*self*, *unperturbed*, *perturbed_inputs*, *alphas*)
Projection onto given l2 / linf balls in a batch.

`select_delta` (*self*, *dist_post_update*, *current_iteration*)
Choose the delta at the scale of distance between x and perturbed sample.

class `foolbox.attacks.`**`GenAttack`**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)
The GenAttack introduced in [R996613153a1e-1].

This attack is performs a genetic search in order to find an adversarial perturbation in a black-box scenario in as few queries as possible.

References

[R996613153a1e-1]

as_generator (*self*, *a*, *generations=10*, *alpha=1.0*, *p=0.05*, *N=10*, *tau=0.1*, *search_shape=None*, *epsilon=0.3*, *binary_search=20*)

A black-box attack based on genetic algorithms. Can either try to find an adversarial perturbation for a fixed epsilon distance or perform a binary search over epsilon values in order to find a minimal perturbation. Parameters ———- inputs : *numpy.ndarray*

Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

generations [int] Number of generations, i.e. iterations, in the genetic algorithm.

alpha [float] Mutation-range.

p [float] Mutation probability.

N [int] Population size of the genetic algorithm.

tau: float Temperature for the softmax sampling used to determine the parents of the new crossover.

search_shape [tuple (default: None)] Set this to a smaller image shape than the true shape to search in a smaller input space. The input will be scaled using a linear interpolation to match the required input shape of the model.

binary_search [bool or int] Whether to perform a binary search over epsilon and using their values to start the search. If False, hyperparameters are not optimized. Can also be an integer, specifying the number of binary search steps (default 20).

epsilon [float] Limit on the perturbation size; if binary_search is True, this value is only for initialization and automatically adapted.

15.4 Other attacks

class foolbox.attacks.**BinarizationRefinementAttack** (*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

For models that preprocess their inputs by binarizing the inputs, this attack can improve adversarials found by other attacks. It does os by utilizing information about the binarization and mapping values to the corresponding value in the clean input or to the right side of the threshold.

as_generator (*self*, *a*, *starting_point=None*, *threshold=None*, *included_in='upper'*)

For models that preprocess their inputs by binarizing the inputs, this attack can improve adversarials found by other attacks. It does this by utilizing information about the binarization and mapping values to the corresponding value in the clean input or to the right side of the threshold.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

starting_point [*numpy.ndarray*] Adversarial input to use as a starting point.

threshold [float] The treshold used by the models binarization. If none, defaults to (model.bounds()[1] - model.bounds()[0]) / 2.

included_in [str] Whether the threshold value itself belongs to the lower or upper interval.

class foolbox.attacks.**PrecomputedAdversarialsAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Attacks a model using precomputed adversarial candidates.

as_generator(*self*, *a*, *candidate_inputs*, *candidate_outputs*)

Attacks a model using precomputed adversarial candidates.

Parameters

input_or_adv [*numpy.ndarray* or `Adversarial`] The original, unperturbed input as a *numpy.ndarray* or an `Adversarial` instance.

label [int] The reference label of the original input. Must be passed if *a* is a *numpy.ndarray*, must not be passed if *a* is an `Adversarial` instance.

unpack [bool] If true, returns the adversarial input, otherwise returns the Adversarial object.

candidate_inputs [*numpy.ndarray*] The original inputs that will be expected by this attack.

candidate_outputs [*numpy.ndarray*] The adversarial candidates corresponding to the inputs.

class foolbox.attacks.**InversionAttack**(*model=None*, *criterion=<foolbox.criteria.Misclassification object>*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*)

Creates "negative images" by inverting the pixel values according to [R57cf8375f1ff-1].

References

[R57cf8375f1ff-1]

as_generator(*self*, *a*)

Creates "negative images" by inverting the pixel values.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the underlying model.

labels [*numpy.ndarray*] Class labels of the inputs as a vector of integers in [0, number of classes).

unpack [bool] If true, returns the adversarial inputs as an array, otherwise returns Adversarial objects.

Gradient-based attacks

`GradientAttack`	Perturbs the input with the gradient of the loss w.r.t.
`GradientSignAttack`	Adds the sign of the gradient to the input, gradually increasing the magnitude until the input is misclassified.
`FGSM`	alias of `foolbox.v1.attacks.gradient.GradientSignAttack`
`LinfinityBasicIterativeAttack`	The Basic Iterative Method introduced in [Rbd27454db950-1].
`BasicIterativeMethod`	alias of `foolbox.v1.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`
`BIM`	alias of `foolbox.v1.attacks.iterative_projected_gradient.LinfinityBasicIterativeAttack`
`L1BasicIterativeAttack`	Modified version of the Basic Iterative Method that minimizes the L1 distance.
`L2BasicIterativeAttack`	Modified version of the Basic Iterative Method that minimizes the L2 distance.
`ProjectedGradientDescentAttack`	The Projected Gradient Descent Attack introduced in [R37229719ede6-1] without random start.
`ProjectedGradientDescent`	alias of `foolbox.v1.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`
`PGD`	alias of `foolbox.v1.attacks.iterative_projected_gradient.ProjectedGradientDescentAttack`
`RandomStartProjectedGradientDescentAttack`	The Projected Gradient Descent Attack introduced in [R876f5a9eb8eb-1] with random start.
`RandomProjectedGradientDescent`	alias of `foolbox.v1.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`
`RandomPGD`	alias of `foolbox.v1.attacks.iterative_projected_gradient.RandomStartProjectedGradientDescentAttack`
`AdamL1BasicIterativeAttack`	Modified version of the Basic Iterative Method that minimizes the L1 distance using the Adam optimizer.
`AdamL2BasicIterativeAttack`	Modified version of the Basic Iterative Method that minimizes the L2 distance using the Adam optimizer.
`AdamProjectedGradientDescentAttack`	The Projected Gradient Descent Attack introduced in [R78a2267bf0c5-1], [R78a2267bf0c5-2] without random start using the Adam optimizer.
`AdamProjectedGradientDescent`	alias of `foolbox.v1.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`
`AdamPGD`	alias of `foolbox.v1.attacks.iterative_projected_gradient.AdamProjectedGradientDescentAttack`
`AdamRandomStartProjectedGradientDescentAttack`	The Projected Gradient Descent Attack introduced in [Rb42f1f35d85c-1], [Rb42f1f35d85c-2] with random start using the Adam optimizer.

Continued on next page

Table 1 – continued from previous page

`AdamRandomProjectedGradientDescent`	alias of `foolbox.v1.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack`
`AdamRandomPGD`	alias of `foolbox.v1.attacks.iterative_projected_gradient.AdamRandomStartProjectedGradientDescentAttack`
`MomentumIterativeAttack`	The Momentum Iterative Method attack introduced in [R0c7c08fb6fc4-1].
`MomentumIterativeMethod`	alias of `foolbox.v1.attacks.iterative_projected_gradient.MomentumIterativeAttack`
`LBFGSAttack`	Uses L-BFGS-B to minimize the distance between the input and the adversarial as well as the cross-entropy between the predictions for the adversarial and the the one-hot encoded target class.
`DeepFoolAttack`	Simple and close to optimal gradient-based adversarial attack.
`NewtonFoolAttack`	Implements the NewtonFool Attack.
`DeepFoolL2Attack`	
`DeepFoolLinfinityAttack`	
`ADefAttack`	Adversarial attack that distorts the image, i.e.
`SLSQPAttack`	Uses SLSQP to minimize the distance between the input and the adversarial under the constraint that the input is adversarial.
`SaliencyMapAttack`	Implements the Saliency Map Attack.
`IterativeGradientAttack`	Like GradientAttack but with several steps for each epsilon.
`IterativeGradientSignAttack`	Like GradientSignAttack but with several steps for each epsilon.
`CarliniWagnerL2Attack`	The L2 version of the Carlini & Wagner attack.
`EADAttack`	Gradient based attack which uses an elastic-net regularization [1].
`DecoupledDirectionNormL2Attack`	The Decoupled Direction and Norm L2 adversarial attack from [R1326043d948c-1].
`SparseFoolAttack`	A geometry-inspired and fast attack for computing sparse adversarial perturbations.
`SparseL1BasicIterativeAttack`	
`VirtualAdversarialAttack`	

Score-based attacks

`SinglePixelAttack`	Perturbs just a single pixel and sets it to the min or max.
`LocalSearchAttack`	A black-box attack based on the idea of greedy local search.
`ApproximateLBFGSAttack`	Same as `LBFGSAttack` with approximate_gradient set to True.

Decision-based attacks

`BoundaryAttack`	A powerful adversarial attack that requires neither gradients nor probabilities.
`SpatialAttack`	Adversarially chosen rotations and translations [1].
`PointwiseAttack`	Starts with an adversarial and performs a binary search between the adversarial and the original for each dimension of the input individually.
`GaussianBlurAttack`	Blurs the input until it is misclassified.
`ContrastReductionAttack`	Reduces the contrast of the input until it is misclassified.
`AdditiveUniformNoiseAttack`	Adds uniform noise to the input, gradually increasing the standard deviation until the input is misclassified.
`AdditiveGaussianNoiseAttack`	Adds Gaussian noise to the input, gradually increasing the standard deviation until the input is misclassified.
`SaltAndPepperNoiseAttack`	Increases the amount of salt and pepper noise until the input is misclassified.
`BlendedUniformNoiseAttack`	Blends the input with a uniform noise input until it is misclassified.
`BoundaryAttackPlusPlus`	
`GenAttack`	
`HopSkipJumpAttack`	A powerful adversarial attack that requires neither gradients nor probabilities.

Other attacks

`BinarizationRefinementAttack`	For models that preprocess their inputs by binarizing the inputs, this attack can improve adversarials found by other attacks.
`PrecomputedAdversarialsAttack`	Attacks a model using precomputed adversarial candidates.
`InversionAttack`	

CHAPTER 16

foolbox.v1.adversarial

Provides a class that represents an adversarial example.

class foolbox.v1.adversarial.**Adversarial**(*model*, *criterion*, *unperturbed*, *original_class*, *distance=<class 'foolbox.distances.MeanSquaredDistance'>*, *threshold=None*, *verbose=False*)

Defines an adversarial that should be found and stores the result.

The *Adversarial* class represents a single adversarial example for a given model, criterion and reference input. It can be passed to an adversarial attack to find the actual adversarial perturbation.

Parameters

model [a `Model` instance] The model that should be fooled by the adversarial.

criterion [a `Criterion` instance] The criterion that determines which inputs are adversarial.

unperturbed [a `numpy.ndarray`] The unperturbed input to which the adversarial input should be as close as possible.

original_class [int] The ground-truth label of the unperturbed input.

distance [a `Distance` class] The measure used to quantify how close inputs are.

threshold [float or `Distance`] If not None, the attack will stop as soon as the adversarial perturbation has a size smaller than this threshold. Can be an instance of the `Distance` class passed to the distance argument, or a float assumed to have the same unit as the the given distance. If None, the attack will simply minimize the distance as good as possible. Note that the threshold only influences early stopping of the attack; the returned adversarial does not necessarily have smaller perturbation size than this threshold; the *reached_threshold()* method can be used to check if the threshold has been reached.

adversarial_class

The argmax of the model predictions for the best adversarial found so far.

None if no adversarial has been found.

backward_one (*self*, *gradient*, *x=None*, *strict=True*)
Interface to model.backward_one for attacks.

Parameters

gradient [*numpy.ndarray*] Gradient of some loss w.r.t. the logits.

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

Returns

gradient [*numpy.ndarray*] The gradient w.r.t the input.

See also:

gradient()

channel_axis (*self*, *batch*)
Interface to model.channel_axis for attacks.

Parameters

batch [bool] Controls whether the index of the axis for a batch of inputs (4 dimensions) or a single input (3 dimensions) should be returned.

distance
The distance of the adversarial input to the original input.

forward (*self*, *inputs*, *greedy=False*, *strict=True*, *return_details=False*)
Interface to model.forward for attacks.

Parameters

inputs [*numpy.ndarray*] Batch of inputs with shape as expected by the model.

greedy [bool] Whether the first adversarial should be returned.

strict [bool] Controls if the bounds for the pixel values should be checked.

forward_and_gradient (*self*, *x*, *label=None*, *strict=True*, *return_details=False*)
Interface to model.forward_and_gradient_one for attacks.

Parameters

x [*numpy.ndarray*] Multiple input with shape as expected by the model (with the batch dimension).

label [*numpy.ndarray*] Labels used to calculate the loss that is differentiated. Defaults to the original label.

strict [bool] Controls if the bounds for the pixel values should be checked.

forward_and_gradient_one (*self*, *x=None*, *label=None*, *strict=True*, *return_details=False*)
Interface to model.forward_and_gradient_one for attacks.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension). Defaults to the original input.

label [int] Label used to calculate the loss that is differentiated. Defaults to the original label.

strict [bool] Controls if the bounds for the pixel values should be checked.

forward_one (*self*, *x*, *strict=True*, *return_details=False*)
Interface to model.forward_one for attacks.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension).

strict [bool] Controls if the bounds for the pixel values should be checked.

gradient_one (*self*, *x=None*, *label=None*, *strict=True*)
Interface to model.gradient_one for attacks.

Parameters

x [*numpy.ndarray*] Single input with shape as expected by the model (without the batch dimension). Defaults to the original input.

label [int] Label used to calculate the loss that is differentiated. Defaults to the original label.

strict [bool] Controls if the bounds for the pixel values should be checked.

has_gradient (*self*)
Returns true if _backward and _forward_backward can be called by an attack, False otherwise.

normalized_distance (*self*, *x*)
Calculates the distance of a given input x to the original input.

Parameters

x [*numpy.ndarray*] The input x that should be compared to the original input.

Returns

Distance The distance between the given input and the original input.

original_class
The class of the original input (ground-truth, not model prediction).

output
The model predictions for the best adversarial found so far.

None if no adversarial has been found.

perturbed
The best adversarial example found so far.

reached_threshold (*self*)
Returns True if a threshold is given and the currently best adversarial distance is smaller than the threshold.

target_class
Interface to criterion.target_class for attacks.

unperturbed
The original input.

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

Bibliography

[R20d0064ee4c9-1] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, "Explaining and Harnessing Adversarial Examples", https://arxiv.org/abs/1412.6572

[R37dbc8f24aee-1] Alexey Kurakin, Ian Goodfellow, Samy Bengio, "Adversarial examples in the physical world",

https://arxiv.org/abs/1607.02533

[R367e8e10528a-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[Re6066bc39e14-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[Re2d4f39a0205-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[Re2d4f39a0205-2] Nicholas Carlini, David Wagner: "Towards Evaluating the Robustness of Neural Networks", https://arxiv.org/abs/1608.04644

[R3210aa339085-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[R3210aa339085-2] Nicholas Carlini, David Wagner: "Towards Evaluating the Robustness of Neural Networks", https://arxiv.org/abs/1608.04644

[R86d363e1fb2f-1] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, Jianguo Li, "Boosting Adversarial Attacks with Momentum", https://arxiv.org/abs/1710.06081

[Rb4dd02640756-1] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard, "DeepFool: a simple and accurate method to fool deep neural networks", https://arxiv.org/abs/1511.04599

[R6a972939b320-1] Uyeong Jang et al., "Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning", https://dl.acm.org/citation.cfm?id=3134635

[R08e06ca693ba-1] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, Ananthram Swami, "The Limitations of Deep Learning in Adversarial Settings", https://arxiv.org/abs/1511.07528

[Rc2cb572b91c5-1] Nicholas Carlini, David Wagner: "Towards Evaluating the Robustness of Neural Networks", https://arxiv.org/abs/1608.04644

[Rc2cb572b91c5-2] https://github.com/carlini/nn_robust_attacks

[Rf0e4124daa63-1] Pin-Yu Chen (*), Yash Sharma (*), Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, "EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples", https://arxiv.org/abs/1709.04114

[Rf0e4124daa63-2] Pin-Yu Chen (*), Yash Sharma (*), Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, "Reference Implementation of 'EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples'", https://github.com/ysharma1126/EAD_Attack/blob/master/en_attack.py

[R0e9d4da0ab48-1] Jérôme Rony, Luiz G. Hafemann, Luiz S. Oliveira, Ismail Ben Ayed,

[R0591d14da1c3-1] Florian Tramèr, Dan Boneh, "Adversarial Training and Robustness for Multiple Perturbations", https://arxiv.org/abs/1904.13000

[Rc6516d158ac2-1] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, Shin Ishii, "Distributional Smoothing with Virtual Adversarial Training", https://arxiv.org/abs/1507.00677

[Rb320cee6998a-1] Nina Narodytska, Shiva Prasad Kasiviswanathan, "Simple Black-Box Adversarial Perturbations for Deep Networks", https://arxiv.org/abs/1612.06299

[Re72ca268aa55-1] Wieland Brendel (*), Jonas Rauber (*), Matthias Bethge, "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models", https://arxiv.org/abs/1712.04248

[Rdffd25498f9d-1] Logan Engstrom*, Brandon Tran*, Dimitris Tsipras*, Ludwig Schmidt, Aleksander Mądry: "A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations", http://arxiv.org/abs/1712.02779

[R739f80a24875-1] L. Schott, J. Rauber, M. Bethge, W. Brendel: "Towards the first adversarially robust neural network model on MNIST", ICLR (2019) https://arxiv.org/abs/1805.09190

[R996613153a1e-1] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, Mani Srivastava, "GenAttack: Practical Black-box Attacks with Gradient-Free Optimization",

https://arxiv.org/abs/1607.02533

[R57cf8375f1ff-1] Hossein Hosseini, Baicen Xiao, Mayoore Jaiswal, Radha Poovendran, "On the Limitation of Convolutional Neural Networks in Recognizing Negative Images",

https://arxiv.org/abs/1607.02533

[R20d0064ee4c9-1] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, "Explaining and Harnessing Adversarial Examples", https://arxiv.org/abs/1412.6572

[R37dbc8f24aee-1] Alexey Kurakin, Ian Goodfellow, Samy Bengio, "Adversarial examples in the physical world",

https://arxiv.org/abs/1607.02533

[R367e8e10528a-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[Re6066bc39e14-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[Re2d4f39a0205-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[Re2d4f39a0205-2] Nicholas Carlini, David Wagner: "Towards Evaluating the Robustness of Neural Networks", https://arxiv.org/abs/1608.04644

[R3210aa339085-1] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks", https://arxiv.org/abs/1706.06083

[R3210aa339085-2] Nicholas Carlini, David Wagner: "Towards Evaluating the Robustness of Neural Networks", https://arxiv.org/abs/1608.04644

[R86d363e1fb2f-1] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, Jianguo Li, "Boosting Adversarial Attacks with Momentum", https://arxiv.org/abs/1710.06081

[Rb4dd02640756-1] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Pascal Frossard, "DeepFool: a simple and accurate method to fool deep neural networks", https://arxiv.org/abs/1511.04599

[R6a972939b320-1] Uyeong Jang et al., "Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning", https://dl.acm.org/citation.cfm?id=3134635

[R08e06ca693ba-1] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, Ananthram Swami, "The Limitations of Deep Learning in Adversarial Settings", https://arxiv.org/abs/1511.07528

[Rc2cb572b91c5-1] Nicholas Carlini, David Wagner: "Towards Evaluating the Robustness of Neural Networks", https://arxiv.org/abs/1608.04644

[Rc2cb572b91c5-2] https://github.com/carlini/nn_robust_attacks

[Rf0e4124daa63-1] Pin-Yu Chen (*), Yash Sharma (*), Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, "EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples", https://arxiv.org/abs/1709.04114

[Rf0e4124daa63-2] Pin-Yu Chen (*), Yash Sharma (*), Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, "Reference Implementation of 'EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples'", https://github.com/ysharma1126/EAD_Attack/blob/master/en_attack.py

[R0e9d4da0ab48-1] Jérôme Rony, Luiz G. Hafemann, Luiz S. Oliveira, Ismail Ben Ayed,

[R0591d14da1c3-1] Florian Tramèr, Dan Boneh, "Adversarial Training and Robustness for Multiple Perturbations", https://arxiv.org/abs/1904.13000

[Rc6516d158ac2-1] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, Shin Ishii, "Distributional Smoothing with Virtual Adversarial Training", https://arxiv.org/abs/1507.00677

[Rb320cee6998a-1] Nina Narodytska, Shiva Prasad Kasiviswanathan, "Simple Black-Box Adversarial Perturbations for Deep Networks", https://arxiv.org/abs/1612.06299

[Re72ca268aa55-1] Wieland Brendel (*), Jonas Rauber (*), Matthias Bethge, "Decision-Based Adversarial Attacks: Reliable Attacks Against Black-Box Machine Learning Models", https://arxiv.org/abs/1712.04248

[Rdffd25498f9d-1] Logan Engstrom*, Brandon Tran*, Dimitris Tsipras*, Ludwig Schmidt, Aleksander Mądry: "A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations", http://arxiv.org/abs/1712.02779

[R739f80a24875-1] L. Schott, J. Rauber, M. Bethge, W. Brendel: "Towards the first adversarially robust neural network model on MNIST", ICLR (2019) https://arxiv.org/abs/1805.09190

[R996613153a1e-1] Moustafa Alzantot, Yash Sharma, Supriyo Chakraborty, Huan Zhang, Cho-Jui Hsieh, Mani Srivastava, "GenAttack: Practical Black-box Attacks with Gradient-Free Optimization",

https://arxiv.org/abs/1607.02533

[R57cf8375f1ff-1] Hossein Hosseini, Baicen Xiao, Mayoore Jaiswal, Radha Poovendran, "On the Limitation of Convolutional Neural Networks in Recognizing Negative Images",

https://arxiv.org/abs/1607.02533

Python Module Index

f

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N