# advertorch_test Documentation

tracy

Apr 03, 2019

# User Guide

# CHAPTER 1

# Installation

## 1.1 Latest version (v0.1)

Installing AdverTorch itself

We developed AdverTorch under Python 3.6 and PyTorch 1.0.0 & 0.4.1. To install AdverTorch, simply run

```
pip install advertorch
```

or clone the repo and run

```
python setup.py install
```

To install the package in "editable" mode:

```
pip install -e .
```

## 1.2 Setting up the testing environments

Some attacks are tested against implementations in [Foolbox](https://github.com/bethgelab/foolbox) or [CleverHans](https://github.com/tensorflow/cleverhans) to ensure correctness. Currently, they are tested under the following versions of related libraries.

```
conda install -c anaconda tensorflow-gpu==1.11.0
pip install git+https://github.com/tensorflow/cleverhans.
↪git@336b9f4ed95dccc7f0d12d338c2038c53786ab70
pip install Keras==2.2.2
pip install foolbox==1.3.2
```

# CHAPTER 2

# Attack, Defense, and BPDA

[ ]:
```
# Copyright (c) 2018-present, Royal Bank of Canada.
# All rights reserved.
#
# This source code is licensed under the license found in the
# LICENSE file in the root directory of this source tree.
#
```

[1]:
```
import matplotlib.pyplot as plt
%matplotlib inline

import os
import argparse
import torch
import torch.nn as nn

from advertorch.utils import predict_from_logits
from advertorch_examples.utils import get_mnist_test_loader
from advertorch_examples.utils import _imshow

torch.manual_seed(0)
use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")
```

## 2.1 Load model that is trained with `tut_train_mnist.py`

[2]:
```
from advertorch.test_utils import LeNet5
from advertorch_examples.utils import TRAINED_MODEL_PATH

filename = "mnist_lenet5_clntrained.pt"
# filename = "mnist_lenet5_advtrained.pt"

```

(continues on next page)

(continued from previous page)

```
model = LeNet5()
model.load_state_dict(
    torch.load(os.path.join(TRAINED_MODEL_PATH, filename)))
model.to(device)
model.eval()
```

```
/home/gavin/anaconda3/envs/dev/lib/python3.6/site-packages/h5py/__init__.py:36:␣
↪FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.
↪floating` is deprecated. In future, it will be treated as `np.float64 == np.
↪dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

[2]:
```
LeNet5(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (relu1): ReLU(inplace)
  (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
↪mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (relu2): ReLU(inplace)
  (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_
↪mode=False)
  (linear1): Linear(in_features=3136, out_features=200, bias=True)
  (relu3): ReLU(inplace)
  (linear2): Linear(in_features=200, out_features=10, bias=True)
)
```

## 2.2 Load data

[3]:
```
batch_size = 5
loader = get_mnist_test_loader(batch_size=batch_size)
for cln_data, true_label in loader:
    break
cln_data, true_label = cln_data.to(device), true_label.to(device)
```

## 2.3 Construct a LinfPGDAttack adversary instance

[4]:
```
from advertorch.attacks import LinfPGDAttack

adversary = LinfPGDAttack(
    model, loss_fn=nn.CrossEntropyLoss(reduction="sum"), eps=0.15,
    nb_iter=40, eps_iter=0.01, rand_init=True, clip_min=0.0, clip_max=1.0,
    targeted=False)
```

## 2.4 Perform untargeted attack

[5]:
```
adv_untargeted = adversary.perturb(cln_data, true_label)
```

## 2.5 Perform targeted attack

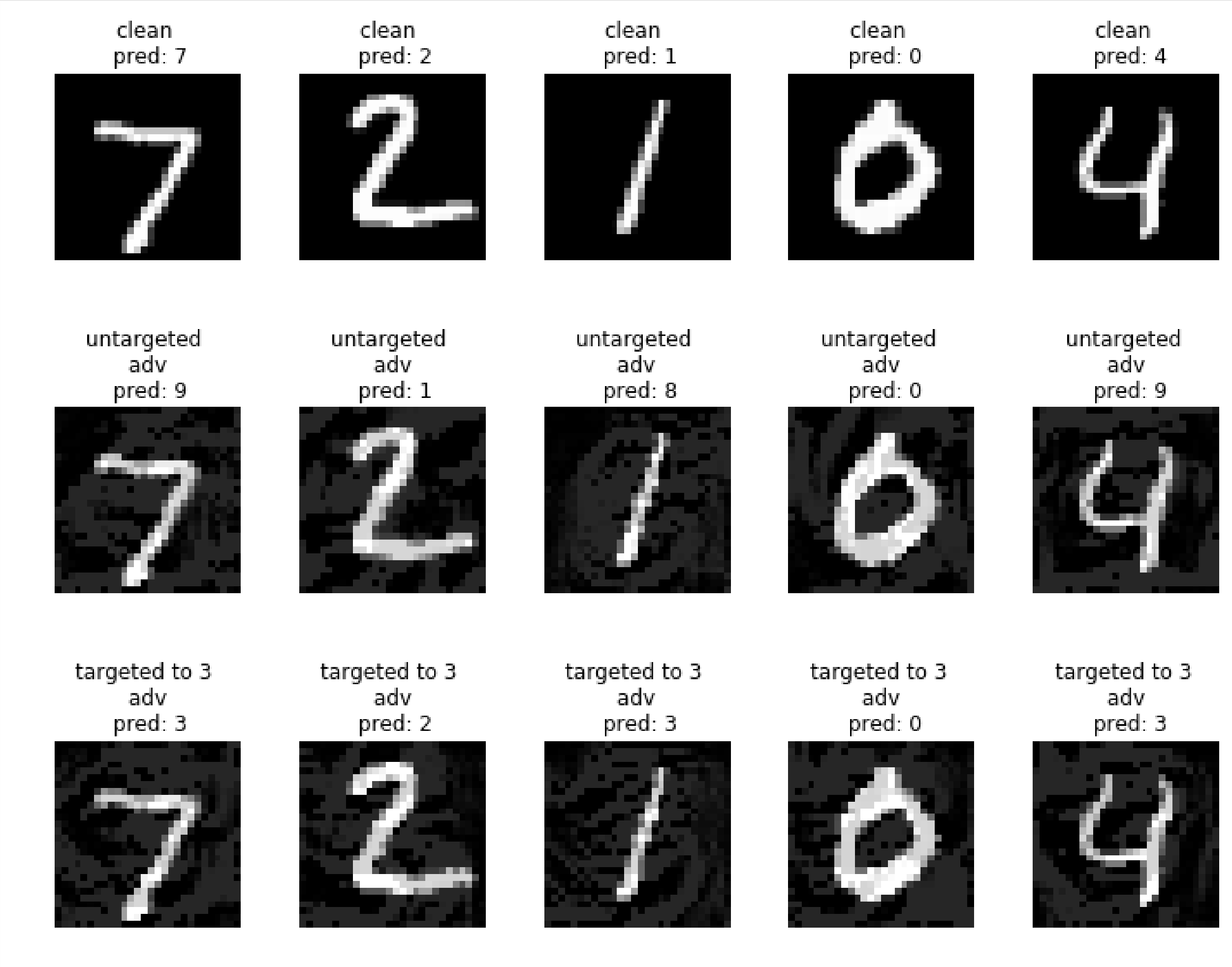
[6]:
```
target = torch.ones_like(true_label) * 3
adversary.targeted = True
adv_targeted = adversary.perturb(cln_data, target)
```

## 2.6 Visualization of attacks

[7]:
```
pred_cln = predict_from_logits(model(cln_data))
pred_untargeted_adv = predict_from_logits(model(adv_untargeted))
pred_targeted_adv = predict_from_logits(model(adv_targeted))

import matplotlib.pyplot as plt
plt.figure(figsize=(10, 8))
for ii in range(batch_size):
    plt.subplot(3, batch_size, ii + 1)
    _imshow(cln_data[ii])
    plt.title("clean \n pred: {}".format(pred_cln[ii]))
    plt.subplot(3, batch_size, ii + 1 + batch_size)
    _imshow(adv_untargeted[ii])
    plt.title("untargeted \n adv \n pred: {}".format(
        pred_untargeted_adv[ii]))
    plt.subplot(3, batch_size, ii + 1 + batch_size * 2)
    _imshow(adv_targeted[ii])
    plt.title("targeted to 3 \n adv \n pred: {}".format(
        pred_targeted_adv[ii]))

plt.tight_layout()
plt.show()
```




## 2.7 Construct defenses based on preprocessing

[8]:
```
from advertorch.defenses import MedianSmoothing2D
from advertorch.defenses import BitSqueezing
from advertorch.defenses import JPEGFilter

bits_squeezing = BitSqueezing(bit_depth=5)
median_filter = MedianSmoothing2D(kernel_size=3)
jpeg_filter = JPEGFilter(10)

defense = nn.Sequential(
    jpeg_filter,
    bits_squeezing,
    median_filter,
)
```

## 2.8 Process the inputs using the defense

here we use the previous untargeted attack as the running example.

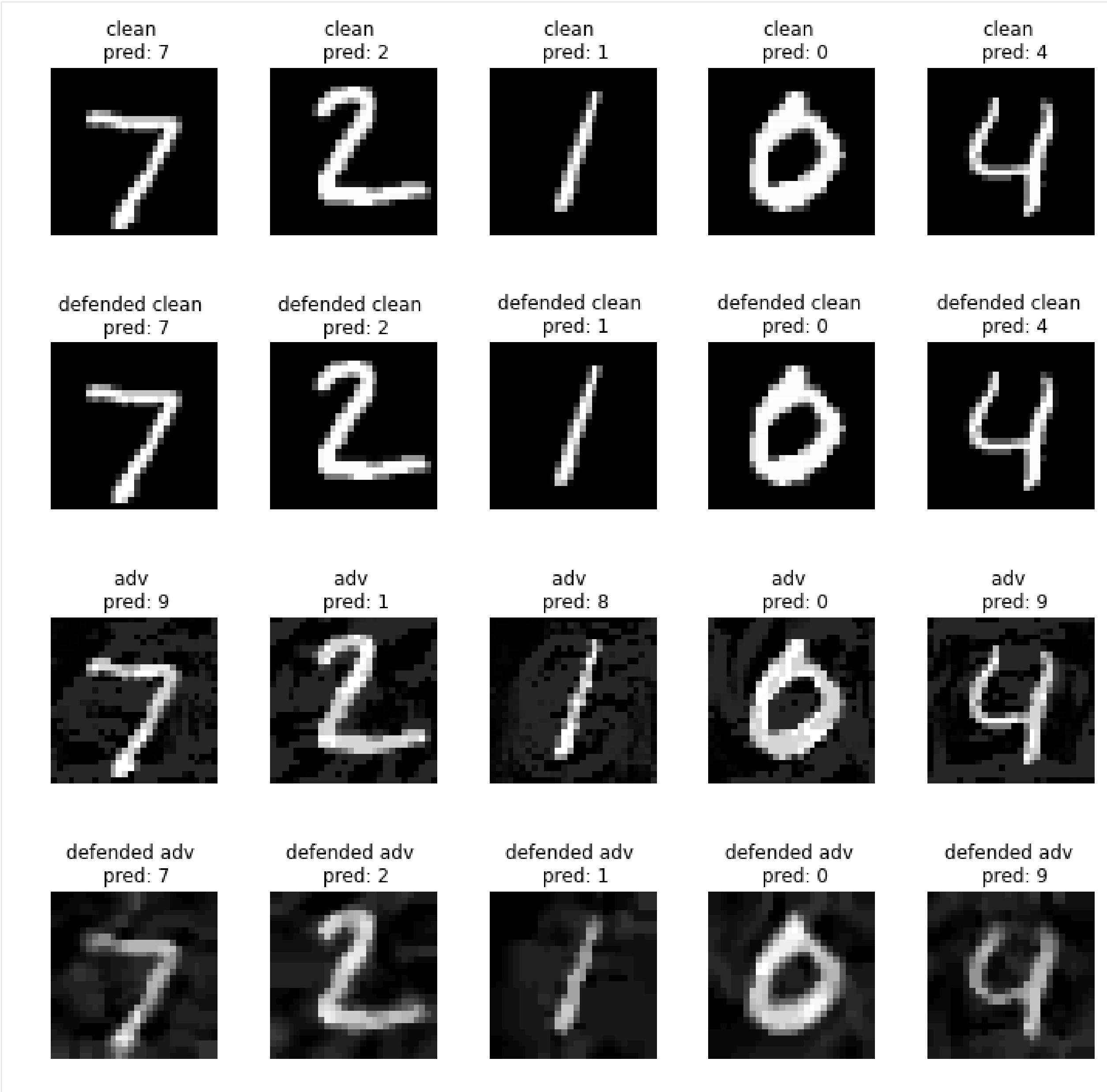
[9]:
```
adv = adv_untargeted
adv_defended = defense(adv)
cln_defended = defense(cln_data)
```

## 2.9 Visualization of defenses

[10]:
```
pred_cln = predict_from_logits(model(cln_data))
pred_cln_defended = predict_from_logits(model(cln_defended))
pred_adv = predict_from_logits(model(adv))
pred_adv_defended = predict_from_logits(model(adv_defended))


import matplotlib.pyplot as plt
plt.figure(figsize=(10, 10))
for ii in range(batch_size):
    plt.subplot(4, batch_size, ii + 1)
    _imshow(cln_data[ii])
    plt.title("clean \n pred: {}".format(pred_cln[ii]))
    plt.subplot(4, batch_size, ii + 1 + batch_size)
    _imshow(cln_data[ii])
    plt.title("defended clean \n pred: {}".format(pred_cln_defended[ii]))
    plt.subplot(4, batch_size, ii + 1 + batch_size * 2)
    _imshow(adv[ii])
    plt.title("adv \n pred: {}".format(
        pred_adv[ii]))
    plt.subplot(4, batch_size, ii + 1 + batch_size * 3)
    _imshow(adv_defended[ii])
    plt.title("defended adv \n pred: {}".format(
        pred_adv_defended[ii]))

plt.tight_layout()
plt.show()
```




## 2.10 BPDA (Backward Pass Differentiable Approximation)

BPDA is a method proposed in [1], which can be used to attack non-differentiable preprocessing based defenses. Here we use $f(x)$ to denote a non-differentiable component, and $g(x)$ to denote a differentiable component that is similar to $f(x)$. In BPDA, $f(x)$ is used in forward computation, and in the backward computation $g(x)$ is used to propagate down the gradients.

Here we use BPDA to perform adaptive attack towards the defenses we used above.

[1] Athalye, A., Carlini, N. & Wagner, D.. (2018). Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. Proceedings of the 35th International Conference on Machine Learning, in PMLR 80:274-283

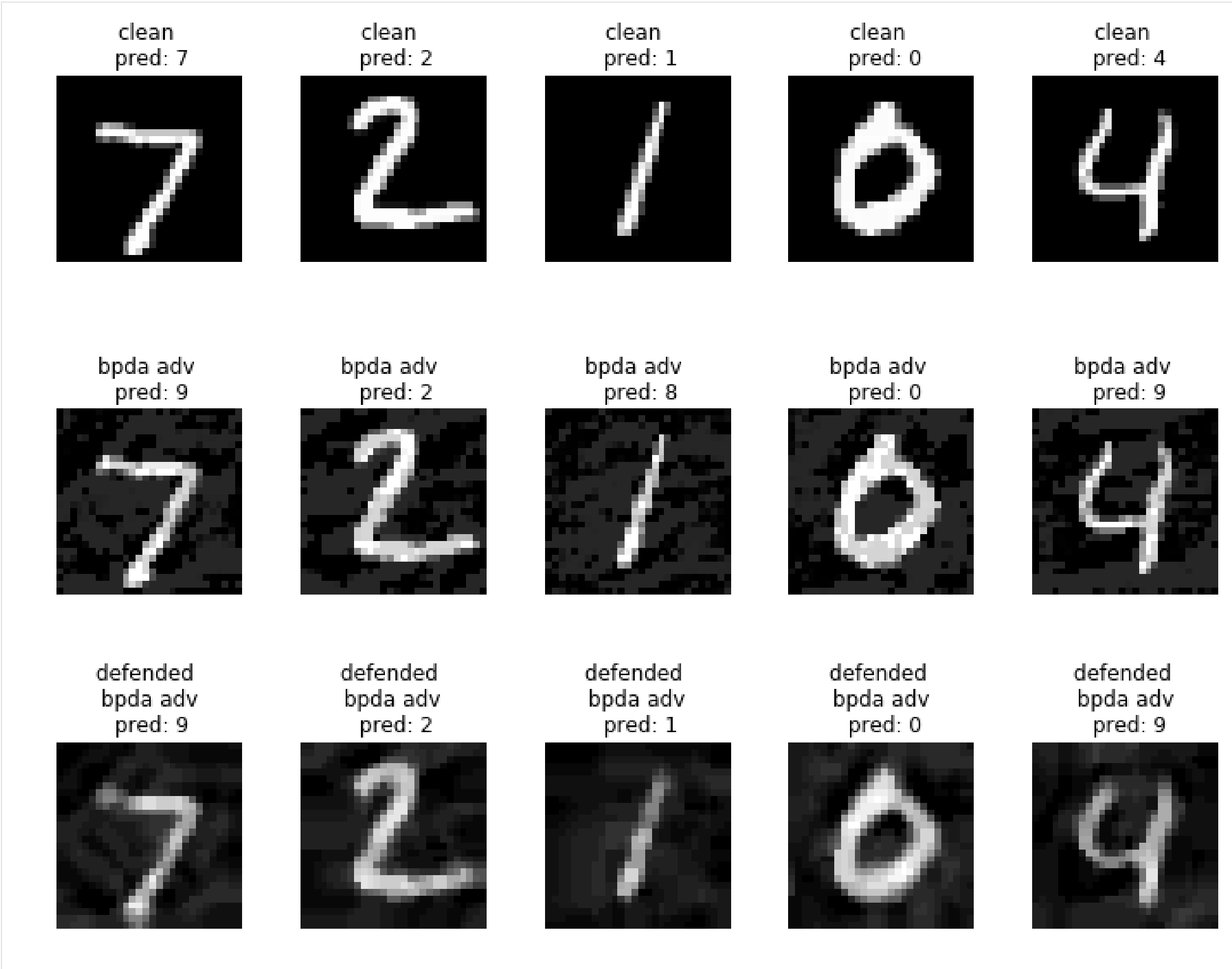
[11]:
```
from advertorch.bpda import BPDAWrapper
defense_withbpda = BPDAWrapper(defense, forwardsub=lambda x: x)
defended_model = nn.Sequential(defense_withbpda, model)
bpda_adversary = LinfPGDAttack(
    defended_model, loss_fn=nn.CrossEntropyLoss(reduction="sum"), eps=0.15,
    nb_iter=1000, eps_iter=0.005, rand_init=True, clip_min=0.0, clip_max=1.0,
    targeted=False)


bpda_adv = bpda_adversary.perturb(cln_data, true_label)
bpda_adv_defended = defense(bpda_adv)
```

[12]:
```
pred_cln = predict_from_logits(model(cln_data))
pred_bpda_adv = predict_from_logits(model(bpda_adv))
pred_bpda_adv_defended = predict_from_logits(model(bpda_adv_defended))


import matplotlib.pyplot as plt
plt.figure(figsize=(10, 8))
for ii in range(batch_size):
    plt.subplot(3, batch_size, ii + 1)
    _imshow(cln_data[ii])
    plt.title("clean \n pred: {}".format(pred_cln[ii]))
    plt.subplot(3, batch_size, ii + 1 + batch_size)
    _imshow(bpda_adv[ii])
    plt.title("bpda adv \n pred: {}".format(
        pred_bpda_adv[ii]))
    plt.subplot(3, batch_size, ii + 1 + batch_size * 2)
    _imshow(bpda_adv_defended[ii])
    plt.title("defended \n bpda adv \n pred: {}".format(
        pred_bpda_adv_defended[ii]))

plt.tight_layout()
plt.show()
```


clean
pred: 7
clean
pred: 2
clean
pred: 1
clean
pred: 0
clean
pred: 4
bpda adv
pred: 9
bpda adv
pred: 2
bpda adv
pred: 8
bpda adv
pred: 0
bpda adv
pred: 9
defended
bpda adv
pred: 9
defended
bpda adv
pred: 2
defended
bpda adv
pred: 1
defended
bpda adv
pred: 0
defended
bpda adv
pred: 9

# CHAPTER 3

## advertorch.attacks

### 3.1 Attacks

| | |
|---|---|
| *Attack* | Abstract base class for all attack classes. |
| *GradientAttack* | Perturbs the input with gradient (not gradient sign) of the loss wrt the input. |
| *GradientSignAttack* | One step fast gradient sign method (Goodfellow et al, 2014). |
| *FastFeatureAttack* | Fast attack against a target internal representation of a model using gradient descent (Sabour et al. |
| *L2BasicIterativeAttack* | Like GradientAttack but with several steps for each epsilon. |
| *LinfBasicIterativeAttack* | Like GradientSignAttack but with several steps for each epsilon. |
| *PGDAttack* | The projected gradient descent attack (Madry et al, 2017). |
| *LinfPGDAttack* | PGD Attack with order=Linf |
| *L2PGDAttack* | PGD Attack with order=L2 |
| *MomentumIterativeAttack* | The L-inf projected gradient descent attack (Dong et al. |
| *CarliniWagnerL2Attack* | Carlini, Nicholas, and David Wagner "Towards evaluating the robustness of neural networks" 2017 IEEE Symposium on Security and Privacy (SP) IEEE, 2017. |
| *LBFGSAttack* | The attack that uses L-BFGS to minimize the distance of the original and perturbed images |
| *SinglePixelAttack* | Single Pixel Attack Algorithm 1 in https://arxiv.org/pdf/1612.06299.pdf |
| *LocalSearchAttack* | Local Search Attack Algorithm 3 in https://arxiv.org/pdf/1612.06299.pdf |
| *SpatialTransformAttack* | Sptially Transformed Attack |

Continued on next page

Table 1 – continued from previous page

| | |
|---|---|
| *JacobianSaliencyMapAttack* | Jacobian Saliency Map Attack This includes Algorithm 1 and 3 in v1 |

## 3.2 Detailed description

**class** advertorch.attacks.**Attack**(*predict*, *loss_fn*, *clip_min*, *clip_max*)

Abstract base class for all attack classes.

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function that takes .
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.

**perturb**(*x*, ***kwargs*)

Generate the adversarial examples. This method should be overriden in any child class that implements an actual attack.

**Parameters**

- **x** – the model's input tensor.
- ****kwargs** – optional parameters used by child classes.

**Returns** adversarial examples.

**class** advertorch.attacks.**GradientAttack**(*predict*, *loss_fn=None*, *eps=0.3*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

Perturbs the input with gradient (not gradient sign) of the loss wrt the input.

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – attack step size.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **targeted** – indicate if this is a targeted attack.

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

Given examples (x, y), returns their adversarial counterparts with an attack length of eps.

**Parameters**

- **x** – input tensor.
- **y** – label tensor. - if None and self.targeted=False, compute y as predicted

  labels.

  – if self.targeted=True, then y must be the targeted labels.

**Returns** tensor containing perturbed inputs.

**class** advertorch.attacks.**GradientSignAttack**(*predict*, *loss_fn=None*, *eps=0.3*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

One step fast gradient sign method (Goodfellow et al, 2014). Paper: https://arxiv.org/abs/1412.6572

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – attack step size.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **targeted** – indicate if this is a targeted attack.

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

Given examples (x, y), returns their adversarial counterparts with an attack length of eps.

**Parameters**

- **x** – input tensor.
- **y** – label tensor. - if None and self.targeted=False, compute y as predicted

  labels.

  – if self.targeted=True, then y must be the targeted labels.

**Returns** tensor containing perturbed inputs.

**class** advertorch.attacks.**FastFeatureAttack**(*predict*, *loss_fn=None*, *eps=0.3*, *eps_iter=0.05*, *nb_iter=10*, *rand_init=True*, *clip_min=0.0*, *clip_max=1.0*)

Fast attack against a target internal representation of a model using gradient descent (Sabour et al. 2016). Paper: https://arxiv.org/abs/1511.05122

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – maximum distortion.
- **eps_iter** – attack step size.
- **nb_iter** – number of iterations
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.

**perturb**(*source*, *guide*, *delta=None*)

Given source, returns their adversarial counterparts with representations close to that of the guide.

**Parameters**

- **source** – input tensor which we want to perturb.
- **guide** – targeted input.
- **delta** – tensor contains the random initialization.

**Returns** tensor containing perturbed inputs.

**class** advertorch.attacks.**L2BasicIterativeAttack**(*predict*, *loss_fn=None*, *eps=0.1*, *nb_iter=10*, *eps_iter=0.05*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

Like GradientAttack but with several steps for each epsilon.

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – maximum distortion.
- **nb_iter** – number of iterations.
- **eps_iter** – attack step size.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **targeted** – if the attack is targeted.

**class** advertorch.attacks.**LinfBasicIterativeAttack**(*predict*, *loss_fn=None*, *eps=0.1*, *nb_iter=10*, *eps_iter=0.05*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

Like GradientSignAttack but with several steps for each epsilon. Aka Basic Iterative Attack. Paper: https://arxiv.org/pdf/1611.01236.pdf

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – maximum distortion.
- **nb_iter** – number of iterations.
- **eps_iter** – attack step size.
- **rand_init** – (optional bool) random initialization.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **ord** – (optional) the order of maximum distortion (inf or 2).
- **targeted** – if the attack is targeted.

**class** advertorch.attacks.**PGDAttack**(*predict*, *loss_fn=None*, *eps=0.3*, *nb_iter=40*, *eps_iter=0.01*, *rand_init=True*, *clip_min=0.0*, *clip_max=1.0*, *ord=<Mock name='mock.inf' id='139818535385576'>*, *targeted=False*)

The projected gradient descent attack (Madry et al, 2017). The attack performs nb_iter steps of size eps_iter, while always staying within eps from the initial point. Paper: https://arxiv.org/pdf/1706.06083.pdf

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – maximum distortion.

- **nb_iter** – number of iterations.
- **eps_iter** – attack step size.
- **rand_init** – (optional bool) random initialization.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **ord** – (optional) the order of maximum distortion (inf or 2).
- **targeted** – if the attack is targeted.

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

Given examples (x, y), returns their adversarial counterparts with an attack length of eps.

**Parameters**

- **x** – input tensor.
- **y** – label tensor. - if None and self.targeted=False, compute y as predicted

  labels.

  – if self.targeted=True, then y must be the targeted labels.

**Returns** tensor containing perturbed inputs.

**class** advertorch.attacks.**LinfPGDAttack** (*predict*, *loss_fn=None*, *eps=0.3*, *nb_iter=40*, *eps_iter=0.01*, *rand_init=True*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

PGD Attack with order=Linf

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – maximum distortion.
- **nb_iter** – number of iterations.
- **eps_iter** – attack step size.
- **rand_init** – (optional bool) random initialization.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **targeted** – if the attack is targeted.

**class** advertorch.attacks.**L2PGDAttack** (*predict*, *loss_fn=None*, *eps=0.3*, *nb_iter=40*, *eps_iter=0.01*, *rand_init=True*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

PGD Attack with order=L2

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – maximum distortion.
- **nb_iter** – number of iterations.

- **eps_iter** – attack step size.
- **rand_init** – (optional bool) random initialization.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **targeted** – if the attack is targeted.

**class** advertorch.attacks.**MomentumIterativeAttack**(*predict*, *loss_fn=None*, *eps=0.3*, *nb_iter=40*, *decay_factor=1.0*, *eps_iter=0.01*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

The L-inf projected gradient descent attack (Dong et al. 2017). The attack performs nb_iter steps of size eps_iter, while always staying within eps from the initial point. The optimization is performed with momentum. Paper: https://arxiv.org/pdf/1710.06081.pdf

**Parameters**

- **predict** – forward pass function.
- **loss_fn** – loss function.
- **eps** – maximum distortion.
- **nb_iter** – number of iterations
- **decay_factor** – momentum decay factor.
- **eps_iter** – attack step size.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **targeted** – if the attack is targeted.

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

Given examples (x, y), returns their adversarial counterparts with an attack length of eps.

**Parameters**

- **x** – input tensor.
- **y** – label tensor. - if None and self.targeted=False, compute y as predicted

  labels.

  – if self.targeted=True, then y must be the targeted labels.

**Returns** tensor containing perturbed inputs.

**class** advertorch.attacks.**CarliniWagnerL2Attack**(*predict*, *num_classes*, *confidence=0*, *targeted=False*, *learning_rate=0.01*, *binary_search_steps=9*, *max_iterations=10000*, *abort_early=True*, *initial_const=0.001*, *clip_min=0.0*, *clip_max=1.0*, *loss_fn=None*)

Carlini, Nicholas, and David Wagner "Towards evaluating the robustness of neural networks" 2017 IEEE Symposium on Security and Privacy (SP) IEEE, 2017. https://arxiv.org/abs/1608.04644

**Parameters**

- **predict** – forward pass function.

- **num_classes** – number of clasess.
- **confidence** – confidence of the adversarial examples.
- **targeted** – TODO
- **learning_rate** – the learning rate for the attack algorithm
- **binary_search_steps** – number of binary search times to find the optimum
- **max_iterations** – the maximum number of iterations
- **abort_early** – if set to true, abort early if getting stuck in local min
- **initial_const** – initial value of the constant c
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **loss_fn** – loss function

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

Generate the adversarial examples. This method should be overriden in any child class that implements an actual attack.

**Parameters**

- **x** – the model's input tensor.
- ****kwargs** – optional parameters used by child classes.

**Returns** adversarial examples.

**class** advertorch.attacks.**LBFGSAttack** (*predict*, *num_classes*, *batch_size=1*, *binary_search_steps=9*, *max_iterations=100*, *initial_const=0.01*, *clip_min=0*, *clip_max=1*, *loss_fn=None*, *targeted=False*)

The attack that uses L-BFGS to minimize the distance of the original and perturbed images

**Parameters**

- **predict** – forward pass function.
- **num_classes** – number of clasess.
- **batch_size** – number of samples in the batch
- **binary_search_steps** – number of binary search times to find the optimum
- **max_iterations** – the maximum number of iterations
- **initial_const** – initial value of the constant c
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **loss_fn** – loss function
- **targeted** – if the attack is targeted.

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

Generate the adversarial examples. This method should be overriden in any child class that implements an actual attack.

**Parameters**

- **x** – the model's input tensor.

- **`**kwargs`** – optional parameters used by child classes.

**Returns** adversarial examples.

**class** advertorch.attacks.**SinglePixelAttack**(*predict*, *max_pixels=100*, *clip_min=0.0*, *loss_fn=None*, *clip_max=1.0*, *comply_with_foolbox=False*, *targeted=False*)

Single Pixel Attack Algorithm 1 in https://arxiv.org/pdf/1612.06299.pdf

**Parameters**

- **predict** – forward pass function.
- **max_pixels** – max number of pixels to perturb.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **loss_fn** – loss function
- **targeted** – if the attack is targeted.

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

Generate the adversarial examples. This method should be overriden in any child class that implements an actual attack.

**Parameters**

- **x** – the model's input tensor.
- **`**kwargs`** – optional parameters used by child classes.

**Returns** adversarial examples.

**class** advertorch.attacks.**LocalSearchAttack**(*predict*, *clip_min=0.0*, *clip_max=1.0*, *p=1.0*, *r=1.5*, *loss_fn=None*, *d=5*, *t=5*, *k=1*, *round_ub=10*, *seed_ratio=0.1*, *max_nb_seeds=128*, *comply_with_foolbox=False*, *targeted=False*)

Local Search Attack Algorithm 3 in https://arxiv.org/pdf/1612.06299.pdf

**Parameters**

- **predict** – forward pass function.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **p** – parameter controls pixel complexity
- **r** – perturbation value
- **loss_fn** – loss function
- **d** – the half side length of the neighbourhood square
- **t** – the number of pixels perturbed at each round
- **k** – the threshold for k-misclassification
- **round_ub** – an upper bound on the number of rounds

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

Generate the adversarial examples. This method should be overriden in any child class that implements an actual attack.

**Parameters**

- **x** – the model's input tensor.
- ****kwargs** – optional parameters used by child classes.

**Returns** adversarial examples.

**class** advertorch.attacks.**SpatialTransformAttack**(*predict*, *num_classes*, *confidence=0*, *initial_const=1*, *max_iterations=1000*, *search_steps=1*, *loss_fn=None*, *clip_min=0.0*, *clip_max=1.0*, *abort_early=True*, *targeted=False*)

Sptially Transformed Attack

**Parameters**

- **predict** – forward pass function.
- **num_classes** – number of clasess.
- **confidence** – confidence of the adversarial examples.
- **initial_const** – initial value of the constant c
- **max_iterations** – the maximum number of iterations
- **search_steps** – number of search times to find the optimum
- **loss_fn** – loss function
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **abort_early** – if set to true, abort early if getting stuck in local min
- **targeted** – if the attack is targeted

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

Generate the adversarial examples. This method should be overriden in any child class that implements an actual attack.

**Parameters**

- **x** – the model's input tensor.
- ****kwargs** – optional parameters used by child classes.

**Returns** adversarial examples.

**class** advertorch.attacks.**JacobianSaliencyMapAttack**(*predict*, *num_classes*, *clip_min=0.0*, *clip_max=1.0*, *loss_fn=None*, *theta=1.0*, *gamma=1.0*, *comply_cleverhans=False*)

Jacobian Saliency Map Attack This includes Algorithm 1 and 3 in v1

**Parameters**

- **predict** – forward pass function.
- **num_classes** – number of clasess.
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.

- **gamma** – highest percentage of pixels can be modified
- **theta** – perturb length, range is either [theta, 0], [0, theta]

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

Generate the adversarial examples. This method should be overriden in any child class that implements an actual attack.

**Parameters**

- **x** – the model's input tensor.
- ****kwargs** – optional parameters used by child classes.

**Returns** adversarial examples.

CHAPTER 4

# advertorch.defenses

## 4.1 Defenses

| | |
|---|---|
| *Processor* | |
| *ConvSmoothing2D* | Conv Smoothing 2D. |
| *AverageSmoothing2D* | Average Smoothing 2D. |
| *GaussianSmoothing2D* | Gaussian Smoothing 2D. |
| *MedianSmoothing2D* | Median Smoothing 2D. |
| *JPEGFilter* | JPEG Filter. |
| *BitSqueezing* | Bit Squeezing. |
| *BinaryFilter* | Binary Filter. |

## 4.2 Detailed description

**class** advertorch.defenses.**Processor**

**class** advertorch.defenses.**ConvSmoothing2D**(*kernel*)

Conv Smoothing 2D.

**Parameters** **kernel_size** – size of the convolving kernel.

**class** advertorch.defenses.**AverageSmoothing2D**(*channels*, *kernel_size*)

Average Smoothing 2D.

**Parameters**

- **channels** – number of channels in the output.
- **kernel_size** – aperture size.

**class** advertorch.defenses.**GaussianSmoothing2D**(*sigma*, *channels*, *kernel_size=None*)

Gaussian Smoothing 2D.

**Parameters**

- **sigma** – sigma of the Gaussian.
- **channels** – number of channels in the output.
- **kernel_size** – aperture size.

**class** advertorch.defenses.**MedianSmoothing2D**(*kernel_size=3*, *stride=1*)
Median Smoothing 2D.

**Parameters**

- **kernel_size** – aperture linear size; must be odd and greater than 1.
- **stride** – stride of the convolution.

**class** advertorch.defenses.**JPEGFilter**(*quality=75*)
JPEG Filter.

**Parameters** **quality** – quality of the output.

**class** advertorch.defenses.**BitSqueezing**(*bit_depth*, *vmin=0.0*, *vmax=1.0*)
Bit Squeezing.

**Parameters**

- **bit_depth** – bit depth.
- **vmin** – min value.
- **vmax** – max value.

**class** advertorch.defenses.**BinaryFilter**(*vmin=0.0*, *vmax=1.0*)
Binary Filter.

**Parameters**

- **vmin** – min value.
- **vmax** – max value.

# CHAPTER 5

## advertorch.bpda

### 5.1 BPDA

| | |
|---|---|
| *BPDAWrapper* | Wrap forward module with BPDA backward path If forwardsub is not None, then ignore backward |

### 5.2 Detailed description

**class** advertorch.bpda.**BPDAWrapper**(*forward*, *forwardsub=None*, *backward=None*)
Wrap forward module with BPDA backward path If forwardsub is not None, then ignore backward

**Parameters**

- **forwardsub** – substitute forward function for BPDA
- **backward** – substitute backward function for BPDA

# CHAPTER 6

## advertorch.context

### 6.1 Context

| |
|---|
| *ctx_noparamgrad* |
| *ctx_eval* |

### 6.2 Detailed description

**class** advertorch.context.**ctx_noparamgrad**(*module*)

**class** advertorch.context.**ctx_eval**(*module*)

CHAPTER 7

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## a

# Index

# S