advertorch

Aug 12, 2019

User Guide


adver
torch

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
```

{

"cells": [

{ “cell_type”: “markdown”, “metadata”: {}, “source”: [

“# Attack, Defense, and BPDA”

]

}, {

“cell_type”: “code”, “execution_count”: null, “metadata”: {}, “outputs”: [], “source”: [

“# Copyright (c) 2018-present, Royal Bank of Canada and other authors.n” “# See the AUTHORS.txt file for a list of contributors.n” “# All rights reserved.n”, “#n”, “# This source code is licensed under the license found in then”, “# LICENSE file in the root directory of this source tree.n”, “#”

]

}, {

“cell_type”: “code”, “execution_count”: 1, “metadata”: {}, “outputs”: [], “source”: [

“import matplotlib.pyplot as pltn”, “%matplotlib inlinen”, “n”, “import osn”, “import argparsen”, “import torchn”, “import torch.nn as nnn”, “n”, “from advertorch.utils import predict_from_logitsn”, “from advertorch_examples.utils import get_mnist_test_loadern”, “from advertorch_examples.utils import _imshown”, “n”, “torch.manual_seed(0)n”, “use_cuda = torch.cuda.is_available()n”, “device = torch.device(“cuda” if use_cuda else “cpu”)”

]

}, {

“cell_type”: “markdown”, “metadata”: {}, “source”: [

“### Load model that is trained with *tut_train_mnist.py*”

]

}, {

“cell_type”: “code”, “execution_count”: 2, “metadata”: {}, “outputs”: [

{ “name”: “stderr”, “output_type”: “stream”, “text”: [

“/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*.n”, ” from ._conv import register_converters as _register_convertersn”

]

}, {

“data”: {

“text/plain”: [“LeNet5(n”, ” (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))n”, ” (relu1): ReLU(inplace)n”, ” (maxpool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)n”, ” (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))n”, ” (relu2): ReLU(inplace)n”, ” (maxpool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)n”, ” (linear1): Linear(in_features=3136, out_features=200, bias=True)n”, ” (relu3): ReLU(inplace)n”, ” (linear2): Linear(in_features=200, out_features=10, bias=True)n”, “)”

```
            ]
           }, "execution_count": 2, "metadata": {}, "output_type": "execute_result"
          }
       ], "source": [
           "from advertorch.test_utils import LeNet5n", "from advertorch_examples.utils import TRAINED_MODEL_PATHn", "n", "filename = "mnist_lenet5_clntrained.pt"n", "# filename = "mnist_lenet5_advtrained.pt"n", "n", "model = LeNet5()n", "model.load_state_dict(n", " " torch.load(os.path.join(TRAINED_MODEL_PATH, filename)))n", "model.to(device)n", "model.eval()"
       ]
   }, {
       "cell_type": "markdown", "metadata": {}, "source": [
           "### Load data"
       ]
   }, {
       "cell_type": "code", "execution_count": 3, "metadata": {}, "outputs": [], "source": [
           "batch_size = 5n", "loader = get_mnist_test_loader(batch_size=batch_size)n", "for cln_data, true_label in loader:n", " " breakn", "cln_data, true_label = cln_data.to(device), true_label.to(device)"
       ]
   }, {
       "cell_type": "markdown", "metadata": {}, "source": [
           "### Construct a LinfPGDAttack adversary instance"
       ]
   }, {
       "cell_type": "code", "execution_count": 4, "metadata": {}, "outputs": [], "source": [
           "from advertorch.attacks import LinfPGDAttackn", "n", "adversary = LinfPGDAttack(n", " " model, loss_fn=nn.CrossEntropyLoss(reduction="sum"), eps=0.15,n", " " nb_iter=40, eps_iter=0.01, rand_init=True, clip_min=0.0, clip_max=1.0,n", " " targeted=False)"
       ]
   }, {
       "cell_type": "markdown", "metadata": {}, "source": [
           "### Perform untargeted attack"
       ]
   }, {
       "cell_type": "code", "execution_count": 5, "metadata": {}, "outputs": [], "source": [
           "adv_untargeted = adversary.perturb(cln_data, true_label)"
       ]
```

}, {

“cell_type”: “markdown”, “metadata”: {}, “source”: [

“### Perform targeted attack”

]

}, {

“cell_type”: “code”, “execution_count”: 6, “metadata”: {}, “outputs”: [], “source”: [

“target = torch.ones_like(true_label) * 3n”, “adversary.targeted = Truen”, “adv_targeted = adversary.perturb(cln_data, target)”

]

}, {

“cell_type”: “markdown”, “metadata”: {}, “source”: [

“### Visualization of attacks”

]

}, {

“cell_type”: “code”, “execution_count”: 7, “metadata”: {}, “outputs”: [

{

“data”: { “image/png”: “iVBORw0KGgoAAAANSUhEUgAAAsgAAAIsCAYAAAAkIoxHAAAABHNCS

“text/plain”: [

“<Figure size 720x576 with 15 Axes>”

]

}, “metadata”: {}, “output_type”: “display_data”

}

], “source”: [

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

]

}, {

“cell_type”: “markdown”, “metadata”: {}, “source”: [

“### Construct defenses based on preprocessing”

]

}, {

“cell_type”: “code”, “execution_count”: 8, “metadata”: {}, “outputs”: [], “source”: [

“from advertorch.defenses import MedianSmoothing2Dn”, “from advertorch.defenses import BitSqueezingn”, “from advertorch.defenses import JPEGFiltern”, “n”, “bits_squeezing = BitSqueezing(bit_depth=5)n”, “median_filter = MedianSmoothing2D(kernel_size=3)n”, “jpeg_filter = JPEGFilter(10)n”, “n”, “defense = nn.Sequential(n”, ” jpeg_filter,n”, ” bits_squeezing,n”, ” median_filter,n”, “)”

]

}, {

“cell_type”: “markdown”, “metadata”: {}, “source”: [

“### Process the inputs using the defensen”, “here we use the previous untargeted attack as the running example. “

]

}, {

“cell_type”: “code”, “execution_count”: 9, “metadata”: {}, “outputs”: [], “source”: [

“adv = adv_untargetedn”, “adv_defended = defense(adv)n”, “cln_defended = defense(cln_data)”

]

}, {

“cell_type”: “markdown”, “metadata”: {}, “source”: [

“### Visualization of defenses”

]

}, {

“cell_type”: “code”, “execution_count”: 10, “metadata”: {

“scrolled”: false

}, “outputs”: [

{

“data”: { “image/png”: “iVBORw0KGgoAAAANSUhEUgAAAsgAAAK+CAYAAAC/0OiqAAAABHNC

“text/plain”: [

“<Figure size 720x720 with 20 Axes>”

]

}, “metadata”: {}, “output_type”: “display_data”

}

], “source”: [

“pred_cln = predict_from_logits(model(cln_data))n”, “pred_cln_defended = predict_from_logits(model(cln_defended))n”, “pred_adv = predict_from_logits(model(adv))n”, “pred_adv_defended = predict_from_logits(model(adv_defended))n”, “n”, “n”, “import matplotlib.pyplot as pltn”, “plt.figure(figsize=(10, 10))n”, “for ii in range(batch_size):n”, ” plt.subplot(4, batch_size, ii + 1)n”, ” _imshow(cln_data[ii])n”, ” plt.title(“clean \n pred: {}”.format(pred_cln[ii]))n”, ” plt.subplot(4, batch_size, ii + 1 +

batch_size)n", " _imshow(cln_data[ii])n", " plt.title("defended clean \n pred: {}".format(pred_cln_defended[ii]))n", " plt.subplot(4, batch_size, ii + 1 + batch_size * 2)n", " _imshow(adv[ii])n", " plt.title("adv \n pred: {}".format(n", " pred_adv[ii]))n", " plt.subplot(4, batch_size, ii + 1 + batch_size * 3)n", " _imshow(adv_defended[ii])n", " plt.title("defended adv \n pred: {}".format(n", " pred_adv_defended[ii]))n", "n", "plt.tight_layout()n", "plt.show()"

]

}, {

"cell_type": "markdown", "metadata": {}, "source": [

"### BPDA (Backward Pass Differentiable Approximation)n", "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.n", "n", "Here we use BPDA to perform adaptive attack towards the defenses we used above.n", "n", "[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"

]

}, {

"cell_type": "code", "execution_count": 11, "metadata": {}, "outputs": [], "source": [

"from advertorch.bpda import BPDAWrappern", "defense_withbpda = BPDAWrapper(defense, forwardsub=lambda x: x)n", "defended_model = nn.Sequential(defense_withbpda, model)n", "bpda_adversary = LinfPGDAttack(n", " defended_model, loss_fn=nn.CrossEntropyLoss(reduction="sum"), eps=0.15,n", " nb_iter=1000, eps_iter=0.005, rand_init=True, clip_min=0.0, clip_max=1.0,n", " targeted=False)n", "n", "n", "bpda_adv = bpda_adversary.perturb(cln_data, true_label)n", "bpda_adv_defended = defense(bpda_adv)"

]

}, {

"cell_type": "code", "execution_count": 12, "metadata": {

"scrolled": false

}, "outputs": [

{

"data": { "image/png": "iVBORw0KGgoAAAANSUhEUgAAAsgAAAItCAYAAAAkIoxHAAAABHNCS

"<Figure size 720x576 with 15 Axes>"

]

}, "metadata": {}, "output_type": "display_data"

}

], "source": [

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

]

}

], “metadata”: {

“kernelspec”: { “display_name”: “Python 3”, “language”: “python”, “name”: “python3”

}, “language_info”: {

“codemirror_mode”: { “name”: “ipython”, “version”: 3

}, “file_extension”: “.py”, “mimetype”: “text/x-python”, “name”: “python”, “nbconvert_exporter”: “python”, “pygments_lexer”: “ipython3”, “version”: “3.7.3”

}

}, “nbformat”: 4, “nbformat_minor”: 2

}

CHAPTER 2

advertorch.attacks

2.1 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
L1PGDAttack	PGD Attack with order=L1
SparseL1DescentAttack	SparseL1Descent Attack
MomentumIterativeAttack	The Momentum Iterative Attack (Dong et al.
LinfMomentumIterativeAttack	The Linf Momentum Iterative Attack Paper: https://arxiv.org/pdf/1710.06081.pdf
L2MomentumIterativeAttack	The L2 Momentum Iterative Attack Paper: https://arxiv.org/pdf/1710.06081.pdf
CarliniWagnerL2Attack	The Carlini and Wagner L2 Attack, https://arxiv.org/abs/1608.04644
ElasticNetL1Attack	The ElasticNet L1 Attack, https://arxiv.org/abs/1709.04114

Continued on next page

Table 1 – continued from previous page

DDNL2Attack	The decoupled direction and norm attack (Rony et al, 2018).
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	Spatially Transformed Attack (Xiao et al.
JacobianSaliencyMapAttack	Jacobian Saliency Map Attack This includes Algorithm 1 and 3 in v1, https://arxiv.org/abs/1511.07528v1

2.2 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(*self*, *x*, ***kwargs*)

Virtual method for generating the adversarial examples.

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(*self*, *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(*self*, *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(*self*, *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.
- **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='139749959564984'>*, *l1_sparsity=None*, *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(*self*, *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.**L1PGDAttack**(*predict*, *loss_fn=None*, *eps=10.0*, *nb_iter=40*, *eps_iter=0.01*, *rand_init=True*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*)

PGD Attack with order=L1

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.**SparseL1DescentAttack**(*predict*, *loss_fn=None*, *eps=0.3*, *nb_iter=40*, *eps_iter=0.01*, *rand_init=False*, *clip_min=0.0*, *clip_max=1.0*, *l1_sparsity=0.95*, *targeted=False*)

SparseL1Descent Attack

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.
- **l1_sparsity** – proportion of zeros in gradient updates

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*, *ord=<Mock name='mock.inf' id='139749959564984'>*)

The Momentum Iterative 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.
- **ord** – the order of maximum distortion (inf or 2).

perturb(*self*, *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*)

The Carlini and Wagner L2 Attack, https://arxiv.org/abs/1608.04644

Parameters

- **predict** – forward pass function.
- **num_classes** – number of clasess.

- **confidence** – confidence of the adversarial examples.
- **targeted** – if the attack is targeted.
- **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(*self*, *x*, *y=None*)

Virtual method for generating the adversarial examples.

Parameters

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

Returns adversarial examples.

class advertorch.attacks.**ElasticNetL1Attack**(*predict*, *num_classes*, *confidence=0*, *targeted=False*, *learning_rate=0.01*, *binary_search_steps=9*, *max_iterations=10000*, *abort_early=False*, *initial_const=0.001*, *clip_min=0.0*, *clip_max=1.0*, *beta=0.01*, *decision_rule='EN'*, *loss_fn=None*)

The ElasticNet L1 Attack, https://arxiv.org/abs/1709.04114

Parameters

- **predict** – forward pass function.
- **num_classes** – number of clasess.
- **confidence** – confidence of the adversarial examples.
- **targeted** – if the attack is targeted.
- **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.
- **beta** – hyperparameter trading off L2 minimization for L1 minimization
- **decision_rule** – EN or L1. Select final adversarial example from all successful examples based on the least elastic-net or L1 distortion criterion.

- **loss_fn** – loss function

perturb(*self*, *x*, *y=None*)
Virtual method for generating the adversarial examples.

Parameters

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

Returns adversarial examples.

class advertorch.attacks.**DDNL2Attack**(*predict*, *nb_iter=100*, *gamma=0.05*, *init_norm=1.0*, *quantize=True*, *levels=256*, *clip_min=0.0*, *clip_max=1.0*, *targeted=False*, *loss_fn=None*)
The decoupled direction and norm attack (Rony et al, 2018). Paper: https://arxiv.org/abs/1811.09600

Parameters

- **predict** – forward pass function.
- **nb_iter** – number of iterations.
- **gamma** – factor to modify the norm at each iteration.
- **init_norm** – initial norm of the perturbation.
- **quantize** – perform quantization at each iteration.
- **levels** – number of quantization levels (e.g. 256 for 8 bit images).
- **clip_min** – mininum value per input dimension.
- **clip_max** – maximum value per input dimension.
- **targeted** – if the attack is targeted.
- **loss_fn** – loss function.

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

Virtual method for generating the adversarial examples.

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*, *clip_max=1.0*, *loss_fn=None*, *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(*self*, *x*, *y=None*)

Virtual method for generating the adversarial examples.

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

Virtual method for generating the adversarial examples.

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*)

Spatially Transformed Attack (Xiao et al. 2018) https://openreview.net/forum?id=HyydRMZC-

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

Virtual method for generating the adversarial examples.

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, https://arxiv.org/abs/1511.07528v1

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

Virtual method for generating the adversarial examples.

Parameters

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

Returns adversarial examples.

CHAPTER 3

advertorch.defenses

3.1 Defenses

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.

3.2 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 4

advertorch.bpda

4.1 BPDA

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

4.2 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 5

advertorch.context

5.1 Context

5.2 Detailed description

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

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

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index

S