Homework 2: Adversarial Robustness

助教：莫易川

邮箱：mo666666@stu.pku.edu.cn

CONTENTS

01/ Brief Introduction

02/ Coding completion

03/ Exploration

CONTENTS

01/ Brief Introduction

Brief Introduction:

Points

- Code completion (14 points: (4+3)*2)
- Tricks for AT (11 points: 7+1.5+2.5)

Requirements

- Word/pdf is both ok.
- Write a report (at most **8** pages).
- Send your report and code to

  trustworthy_ai@163.com

  Theme: Homework2-name-ID
- In Chinese/ English

  Due: 4/27 24:00

Language and wheel

- Python
- PyTorch

Contents included by the *.zip

- homework_attack.ipynb
- homework_defense.py (Other *.py)
- report No checkpoint

02 IDE

jupyter in ANACONDA

Visual Studio Code or PC

Reference:

Amaconda Installation:
https://blog.csdn.net/qq_42257666/article/details/121383450

The usage of Jupyter:
https://zhuanlan.zhihu.com/p/33105153

The documents of Pytorch:
https://pytorch.org/docs/stable/index.html

Vscode Installation:
https://blog.csdn.net/weixin_44950987/article/details/128129613

PyCharm Installation:
https://blog.csdn.net/qq_44809707/article/details/122501118

02 Use GPU to accelerate training

60 free GPU hours

URL: https://tianchi.aliyun.com/

Reference: https://www.bilibili.com/video/BV1Ze411T7JV/?vd_source=b567c1334daded39d41b9b7b5f711f22

CONTENTS

01/ Brief Introduction

02/ Coding completion

02 Preparation

Import package

```python
import torch
from PIL import Image
import matplotlib.pyplot as plt
import torch.nn.functional as F
import json
import torch.nn as nn
import tqdm
from torchvision.models.resnet import resnet50
import torchvision.transforms as transforms
img = Image.open('./test.JPEG')
img = img.convert('RGB')
```

Load image

```python
transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
])
img = transform(img)
img_hwc = img.permute(1, 2, 0)
plt.imshow(img_hwc)
plt.axis("off")
plt.show()
img = img.unsqueeze(dim=0)
```

02 Preparation

Load model

```
class Normalize(nn.Module):
    def __init__(self, mean, std):
        super(Normalize, self).__init__()
        self.mean = torch.tensor(mean)
        self.std = torch.tensor(std)

    def forward(self, x):
        return (x - self.mean.type_as(x)[None,:,None,None]) / self.std.type_as(x)[None,:,None,None]
imagenet_mean = (0.485, 0.456, 0.406)
imagenet_std = (0.229, 0.224, 0.225)
net = resnet50(num_classes=1000, pretrained=True)
model = nn.Sequential(Normalize(mean=imagenet_mean, std=imagenet_std), net)
model.eval()
```

Show the result of the natural image

```
model.eval()
prob = model(img)
label = torch.argmax(prob.squeeze())
target_class = json.load(open("imagenet_class_index.json"))[str(label.item())]
print("Natural Image:", target_class[1])
```

```
Natural Image: church
```

02 Coding completion

PGD attack:

```python
def clamp(X, lower_limit, upper_limit):
    return torch.max(torch.min(X, upper_limit), lower_limit)

def attack_pgd(model, X, y, epsilon=8/255, alpha=2/255, attack_iters=10, restarts=1, lower_limit=torch.tensor([0]), upper_limit=torch.tensor([1])):
    """
    model: Model to attack.
    X: Input image
    y: Class of input image
    epsilon: Budget of PGD attack
    alpha: Step size for PGD attack
    attack_iters: Iterations for PGD attack
    restarts: Restarts for PGD attack
    lower limits: Lower limits for Input Images
    upper limits: Upper limits for Input Images
    """
    ###################################
    # write the code here
    ###################################
    # return max_delta
    pass
```

- Report: 4 point (Tell me how your code works)
- The correctness of the code: 3 point

02 Coding completion

The prediction of the adversarial sample crafted by PGD:

```
delta = attack_pgd(model,img,label.unsqueeze(dim=0))
adv_img = img+delta
adv_img = adv_img.squeeze()
img_hwc = adv_img.permute(1, 2, 0)
plt.imshow(img_hwc)
plt.axis("off")
plt.show()
prob = model(adv_img)
label = torch.argmax(prob.squeeze())
target_class = json.load(open("imagenet_class_index.json"))[str(label.item())]
print("Adversarial Image:", target_class[1])
```

```
Adversarial Image: cab
```

- Report: 4 point (Tell me how your code works)
- The correctness of the code: 3 point

02 Coding completion

C&W attack:

```python
def attack_cw(model, X, y, targeted=False, cw_kappa=0, cw_iters=10000, cw_c=1e-4, binary_search_steps = 9 , cw_lr= 0.01):
    """
    model: Model to attack
    X: Input image
    y: Class of input image
    targeted: Whether to apply targeted attack
    cw_kappa: kappa for C&W attack
    cw_iters: iteration for the C&W attack
    cw_c: constants for C&W attack
    binary_search_steps: steps for binary search
    cw_lr: learning rate for optimizer
    """
    def arctanh(imgs):
        scaling = torch.clamp(imgs, max=1, min=-1)
        x = 0.999999 * scaling
        return 0.5*torch.log((1+x)/(1-x))

    def scaler(x_atanh):
        return ((torch.tanh(x_atanh))+1) * 0.5

    def _f(adv_imgs, labels):
        ###################################
        # write the code here
        ###################################
        # return loss
        pass
    model.eval()
    X = X.detach().clone()
    x_arctanh = arctanh(X)
```

```python
    for _ in tqdm.tqdm(range(binary_search_steps)):
        delta = torch.zeros_like(X)
        delta.detach_()
        delta.requires_grad = True
        optimizer = torch.optim.Adam([delta], lr=cw_lr)
        prev_loss = 1e6
        for step in range(cw_iters):
            optimizer.zero_grad()
            adv_examples = scaler(x_arctanh + delta)
            ###################################
            # write the code here
            ###################################
            if step % (cw_iters // 10) == 0:
                if loss > prev_loss:
                    break
                prev_loss = loss
        adv_imgs = scaler(x_arctanh + delta).detach()
    return adv_imgs
```

- Report: 4 point (Tell me how your code works)
- The correctness of the code: 3 point

The prediction of the adversarial sample attacked by C&W:

```
adv_imgs = attack_cw(model,img,label.unsqueeze(dim=0),cw_iters=10)
adv_img = adv_img.squeeze()
img_hwc = adv_img.permute(1, 2, 0)
plt.imshow(img_hwc)
plt.axis("off")
plt.show()
prob = model(adv_img)
label = torch.argmax(prob.squeeze())
target_class = json.load(open("imagenet_class_index.json"))[str(label.item())]
print("Adversarial Image:", target_class[1])
```

```
Adversarial Image: cab
```

- Report: 4 point (Tell me how your code works)
- The correctness of the code: 3 point

CONTENTS

01/ Brief Introduction

02/ Coding completion

03/ Exploration

03 Exploration

What tricks help 2-epochs AT?

- Report 7 point
- Superior to the baseline 1.5point
- Ranking 2.5point (ACC/2+ROB)

  - 0 ~ 20% ; 2.5point

  - 20% ~ 40%; 2point

  - 40% ~ 60%; 1.5point

  - 60% ~ 80%; 1point

  - 80% ~ 100%; 0.5point

Metrics: PGD-3; budget=8/255; step_size=4/255

Baseline: ACC: 38.12. ROB: 25.30 (test set)

You could:

1 Change AT to other variants (MART, TRADES …)

2 Change hyperparameter (e.g. learning rate schedule)

3 Use a smaller network (Parameter<11.17M)

……

You could not:

1 Use a robust model to finetune

2 Use extra data/ generated data

3 Use a larger model or change the number of epochs.

03 Exploration

resnet.py

```python
class ResNet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3,
                               stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block, 64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512*block.expansion, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out


def ResNet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])
```

homework_defense.py

Hyperparameter setting:

```python
from __future__ import print_function
import os
import argparse
import torchvision
from torch.autograd import Variable
import torch.optim as optim
from torchvision import transforms
from models.resnet import *

parser = argparse.ArgumentParser(description='PyTorch CIFAR TRADES Adversarial Training')
parser.add_argument('--batch-size', type=int, default=128, metavar='N',
                    help='input batch size for training (default: 128)')
parser.add_argument('--test-batch-size', type=int, default=128, metavar='N',
                    help='input batch size for testing (default: 128)')
parser.add_argument('--epochs', type=int, default=2, metavar='N',
                    help='number of epochs to train')
parser.add_argument('--weight-decay', '--wd', default=2e-4,
                    type=float, metavar='W')
parser.add_argument('--lr', type=float, default=0.1, metavar='LR',
                    help='learning rate')
parser.add_argument('--momentum', type=float, default=0.9, metavar='M',
                    help='SGD momentum')
parser.add_argument('--no-cuda', action='store_true', default=False,
                    help='disables CUDA training')
parser.add_argument('--epsilon', default=0.031,
                    help='perturbation')
parser.add_argument('--num-steps', default=10,
                    help='perturb number of steps')
parser.add_argument('--step-size', default=0.007,
                    help='perturb step size')
parser.add_argument('--seed', type=int, default=1, metavar='S',
                    help='random seed (default: 1)')
parser.add_argument('--log-interval', type=int, default=100, metavar='N',
                    help='how many batches to wait before logging training status')
parser.add_argument('--model-dir', default='./model-cifar-wideResNet',
                    help='directory of model for saving checkpoint')
parser.add_argument('--save-freq', '-s', default=1, type=int, metavar='N',
                    help='save frequency')
```

03 Exploration

homework_defense.py

Load model and dataset:

```python
# settings
model_dir = args.model_dir
if not os.path.exists(model_dir):
    os.makedirs(model_dir)
use_cuda = not args.no_cuda and torch.cuda.is_available()
torch.manual_seed(args.seed)
device = torch.device("cuda" if use_cuda else "cpu")
kwargs = {'num_workers': 1, 'pin_memory': True} if use_cuda else {}

# setup data loader
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
])
transform_test = transforms.Compose([
    transforms.ToTensor(),
])
trainset = torchvision.datasets.CIFAR10(root='../data', train=True, download=True, transform=transform_train)
train_loader = torch.utils.data.DataLoader(trainset, batch_size=args.batch_size, shuffle=True, **kwargs)
testset = torchvision.datasets.CIFAR10(root='../data', train=False, download=True, transform=transform_test)
test_loader = torch.utils.data.DataLoader(testset, batch_size=args.test_batch_size, shuffle=False, **kwargs)
```

PGD attack to craft adversarial samples

```python
def PGD(model,
        x_natural,
        y,
        optimizer,
        step_size=0.003,
        epsilon=0.031,
        perturb_steps=10,
        device= torch.device("cuda")):
    # define KL-loss
    criterion = nn.CrossEntropyLoss(size_average=False)
    model.eval()
    x_adv = x_natural+0.001 * torch.randn(x_natural.shape).to(device).detach()
    for _ in range(perturb_steps):
        x_adv.requires_grad_()
        with torch.enable_grad():
            loss = criterion(model(x_adv),y)
        grad = torch.autograd.grad(loss, [x_adv])[0]
        x_adv = x_adv.detach() + step_size * torch.sign(grad.detach())
        x_adv = torch.min(torch.max(x_adv, x_natural - epsilon), x_natural + epsilon)
        x_adv = torch.clamp(x_adv, 0.0, 1.0)
    model.train()
    x_adv = Variable(torch.clamp(x_adv, 0.0, 1.0), requires_grad=False)
    # zero gradient
    optimizer.zero_grad()
    # calculate robust loss
    logits = model(x_adv)
    loss = F.cross_entropy(logits, y)
    return loss
```

homework_defense.py

Function to adversarially train a network.

```
def train(args, model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)

        optimizer.zero_grad()

        # calculate robust loss
        loss = PGD(model=model,
                   x_natural=data,
                   y=target,
                   optimizer=optimizer,
                   step_size=args.step_size,
                   epsilon=args.epsilon,
                   perturb_steps=args.num_steps,
                   device = device)
        loss.backward()
        optimizer.step()

        # print progress
        if batch_idx % args.log_interval == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                       100. * batch_idx / len(train_loader), loss.item()))
```

Function to evaluate the accuracy on the training set.

```
def eval_train(model, device, train_loader):
    model.eval()
    train_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in train_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            train_loss += F.cross_entropy(output, target, size_average=False).item()
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()
    train_loss /= len(train_loader.dataset)
    print('Training: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)'.format(
        train_loss, correct, len(train_loader.dataset),
        100. * correct / len(train_loader.dataset)))
    training_accuracy = correct / len(train_loader.dataset)
    return train_loss, training_accuracy
```

03 Exploration

homework_defense.py

Function to evaluate the accuracy on the test set

```
def eval_test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.cross_entropy(output, target, size_average=False).item()
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()
    test_loss /= len(test_loader.dataset)
    print('Test: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
    test_accuracy = correct / len(test_loader.dataset)
    return test_loss, test_accuracy
```

Main function (Training partition)

```
def main():
    # init model, ResNet18() can be also used here for training
    import time
    start_time = time.time()
    model = ResNet18().to(device)
    optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum, weight_decay=args.weight_decay)

    for epoch in range(1, args.epochs + 1):
        # adjust learning rate for SGD
        adjust_learning_rate(optimizer, epoch)

        # adversarial training
        train(args, model, device, train_loader, optimizer, epoch)

        # evaluation on natural examples
        print('================================================================')
        eval_train(model, device, train_loader)
        eval_test(model, device, test_loader)
        print('================================================================')
        # save checkpoint
        if epoch % args.save_freq == 0:
            torch.save(model.state_dict(),
                       os.path.join(model_dir, 'model-wideres-epoch{}.pt'.format(epoch)))
            torch.save(optimizer.state_dict(),
                       os.path.join(model_dir, 'opt-wideres-checkpoint_epoch{}.tar'.format(epoch)))
    end_time = time.time()
```

03 Exploration

homework_defense.py

Main function (Test partition)

```
def _pgd_whitebox(model,
                  X,
                  y,
                  epsilon=0.031,
                  num_steps=3,
                  step_size=0.0157):
    out = model(X)
    err = (out.data.max(1)[1] != y.data).float().sum()
    X_pgd = Variable(X.data, requires_grad=True)
    for _ in range(num_steps):
        opt = optim.SGD([X_pgd], lr=1e-3)
        opt.zero_grad()
        with torch.enable_grad():
            loss = nn.CrossEntropyLoss()(model(X_pgd), y)
        loss.backward()
        eta = step_size * X_pgd.grad.data.sign()
        X_pgd = Variable(X_pgd.data + eta, requires_grad=True)
        eta = torch.clamp(X_pgd.data - X.data, -epsilon, epsilon)
        X_pgd = Variable(X.data + eta, requires_grad=True)
        X_pgd = Variable(torch.clamp(X_pgd, 0, 1.0), requires_grad=True)
    err_pgd = (model(X_pgd).data.max(1)[1] != y.data).float().sum()
    print('err pgd (white-box): ', err_pgd)
    return err, err_pgd
```

```
def eval_adv_test_whitebox(model, device, test_loader):
    """
    evaluate model by white-box attack
    """
    model.eval()
    robust_err_total = 0
    natural_err_total = 0

    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        # pgd attack
        X, y = Variable(data, requires_grad=True), Variable(target)
        err_natural, err_robust = _pgd_whitebox(model, X, y)
        robust_err_total += err_robust
        natural_err_total += err_natural
    robust_acc = (len(testset)-robust_err_total)/len(testset)
    clean_acc = (len(testset)-natural_err_total)/len(testset)
    print('natural_accuracy: ', clean_acc)
    print('robustness: ', robust_acc)
eval_adv_test_whitebox(model, device, test_loader)
print(end_time - start_time)
```

论文列表

[1] Theoretically principled trade-off between robustness and accuracy

[2] Improving adversarial robustness requires revisiting misclassified examples

[3] Adversarial weight perturbation helps robust generalization

[4] Bag of tricks for adversarial training

[5] Overfitting in adversarially robust deep learning

[6] Improving adversarial robustness via channel-wise activation suppressing

[7] On the Convergence and Robustness of Adversarial Training

[8] Robustbench: a standardized adversarial robustness benchmark

Q&A

Thanks