Adversarial Attack

**Deadline：**

**6 月 6 日**前将报告和源码打包成压缩文件按照**姓名+学号+homework4**

的格式发送到助教邮箱 **[xinfeiliu@mail.ustc.edu.cn](mailto:xinfeiliu@mail.ustc.edu.cn)**

1. Task Description

Attack objective: Non-targeted attack

Attack constraint: L-infinity or L2 norm and Parameter ε

Attack algorithm: FGSM, PGD, BIM, CW, Deepfool etal.

Attack schema: White box attack and Black box attack (perform attack on proxy network)

1. Data Format

Images: CIFAR-10 imagesPretrained Model: <https://github.com/osmr/imgclsmob/tree/master> provides multiple models pretrained on CIFAR-10. You can also train a model yourself.

1. Implementation

# *Load pretrained\_model*

**pretrained\_model** **=** "data/lenet\_mnist\_model.pth"

*# LeNet Model definition*

*# MNIST Test dataset and dataloader declaration*

*# Initialize the network*

**model** **=** **[Net](https://pytorch.org/docs/stable/generated/torch.nn.Module.html" \l "torch.nn.Module" \o "torch.nn.Module)().to([device](https://pytorch.org/docs/stable/tensor_attributes.html" \l "torch.device" \o "torch.device))**

*# Load the pretrained model*

**[model.load\_state\_dict](https://pytorch.org/docs/stable/generated/torch.nn.Module.html" \l "torch.nn.Module.load_state_dict" \o "torch.nn.Module.load_state_dict)([torch.load](https://pytorch.org/docs/stable/generated/torch.load.html" \l "torch.load" \o "torch.load)(pretrained\_model,** **map\_location=[device](https://pytorch.org/docs/stable/tensor_attributes.html" \l "torch.device" \o "torch.device)))**

*# Set the model in evaluation mode. In this case this is for the Dropout layers*

**[model.eval](https://pytorch.org/docs/stable/generated/torch.nn.Module.html" \l "torch.nn.Module.eval" \o "torch.nn.Module.eval)()**

*# FGSM attack code*

**def** **fgsm\_attack(image,** **epsilon,** **data\_grad):**

*# Collect the element-wise sign of the data gradient*

**sign\_data\_grad** **=** **data\_grad.sign()**

*# Create the perturbed image by adjusting each pixel of the input image*

**perturbed\_image** **=** **image** **+** **epsilon\*sign\_data\_grad**

*# Adding clipping to maintain [0,1] range*

**perturbed\_image** **=** **[torch.clamp](https://pytorch.org/docs/stable/generated/torch.clamp.html" \l "torch.clamp" \o "torch.clamp)(perturbed\_image,** 0**,** 1**)**

*# Return the perturbed image*

**return** **perturbed\_image**

**def** **test(** **model,** **[device](https://pytorch.org/docs/stable/tensor_attributes.html" \l "torch.device" \o "torch.device),** **[test\_loader](https://pytorch.org/docs/stable/data.html" \l "torch.utils.data.DataLoader" \o "torch.utils.data.DataLoader),** **epsilon** **):**

*# Accuracy counter*

**correct** **=** 0

*# Loop over all examples in test set*

**for** **data,** **target** **in** **[test\_loader](https://pytorch.org/docs/stable/data.html" \l "torch.utils.data.DataLoader" \o "torch.utils.data.DataLoader):**

*# Send the data and label to the device*

**data,** **target** **=** **data.to([device](https://pytorch.org/docs/stable/tensor_attributes.html" \l "torch.device" \o "torch.device)),** **target.to([device](https://pytorch.org/docs/stable/tensor_attributes.html" \l "torch.device" \o "torch.device))**

*# Set requires\_grad attribute of tensor. Important for Attack*

**data.requires\_grad** **=** **True**

*# Forward pass the data through the model*

**output** **=** **model(data)**

**init\_pred** **=** **output.max(**1**,** **keepdim=True)[**1**]** *# get the index of the max log-probability*

*# If the initial prediction is wrong, don't bother attacking, just move on*

**if** **init\_pred.item()** **!=** **target.item():**

**continue**

*# Calculate the loss*

**loss** **=** **[F.nll\_loss](https://pytorch.org/docs/stable/generated/torch.nn.functional.nll_loss.html" \l "torch.nn.functional.nll_loss" \o "torch.nn.functional.nll_loss)(output,** **target)**

*# Zero all existing gradients*

**[model.zero\_grad](https://pytorch.org/docs/stable/generated/torch.nn.Module.html" \l "torch.nn.Module.zero_grad" \o "torch.nn.Module.zero_grad)()**

*# Calculate gradients of model in backward pass*

**loss.backward()**

*# Collect ``datagrad``*

**data\_grad** **=** **data.grad.data**

*# Restore the data to its original scale*

**data\_denorm** **=** **denorm(data)**

*# Call FGSM Attack*

**perturbed\_data** **=** **fgsm\_attack(data\_denorm,** **epsilon,** **data\_grad)**

*# Re-classify the perturbed image*

**output** **=** **model(perturbed\_data\_normalized)**

*# Check for success*

**final\_pred** **=** **output.max(**1**,** **keepdim=True)[**1**]** *# get the index of the max log-probability*

**if** **final\_pred.item()** **==** **target.item():**

**correct** **+=** 1

*# Calculate final accuracy for this epsilon*

**final\_acc** **=** **correct/**float**(**len**([test\_loader](https://pytorch.org/docs/stable/data.html" \l "torch.utils.data.DataLoader" \o "torch.utils.data.DataLoader)))**

print**(**f"Epsilon: *{***epsilon***}***\t**Test Accuracy = *{***correct***}* / *{*len**([test\_loader](https://pytorch.org/docs/stable/data.html" \l "torch.utils.data.DataLoader" \o "torch.utils.data.DataLoader))***}* = *{***final\_acc***}*"**)**

*# Return the accuracy and an adversarial example*

**return** **final\_acc**

上述代码是FGSM对抗攻击的主要代码，可以仿照上述代码完成其他对抗攻击方法。

1. 评价指标

攻击后神经网络的分类精度，单一类别的识别准确率。对于White-Box attack,我们希望对抗精度在10%以下（CNN:ResNet-20，L-infinity Parameter ε=8/255）,对于Black-Box attack, 我们希望实现一个Surrogate Network进行黑盒攻击。

任务：方法不限，实现一个除FGSM以外的白盒攻击算法，同时实现一个黑盒攻击算法（黑盒攻击只需要实现即可，对效果没有具体要求）。