
实验 1：对抗样本生成

一、实验目的：

掌握面向人工智能数据安全性的典型人工智能数据集构建方法，掌握面向数据安全性分析的典型智能算法构建方法，掌握经典对抗样本生成方法。

二、实验原理：

对抗性攻击有很多类别，每种攻击都有不同的目标和对攻击者知识的假设。一般来说，首要目标是向输入数据添加最少的扰动，以导致所需的错误分类。攻击者的知识假设有多种，其中两种是：白盒和黑盒。白盒攻击假设攻击者完全了解并有权访问模型，包括架构、输入、输出和权重。黑盒攻击假设攻击者只能访问模型的输入和输出，并且对底层架构或权重一无所知。还有多种类型的目标，包括错误分类和源/目标错误分类。错误分类的目标意味着对手只希望输出分类是错误的，而不关心新的分类是什么。源/目标错误分类意味着攻击者想要更改最初属于特定源类的图像，以便将其分类为特定目标类。

三、实验内容：

本次实验从最基本的白盒攻击入手，教授最常用的样本攻击方法快速梯度符号法（FGSM, Fast Gradient-Sign Method）和投影梯度下降法（PGD, Project Gradient Descent）。包括样本攻击模块构建、被攻击模型构建、测试函数构建、结果可视化实现等内容。

四、实验步骤：

本实验具体步骤如下所示，方便大家理解和使用。（代码示例仅

作参考，同学们有更好的思路可以自行编写代码)

(1) 环境构建

安装 **anaconda** 软件，用于环境配置。通过 **anaconda prompt** 打开命令行终端，执行下述命令安装相关包

```
conda create -n exp python=3.8
conda activate exp
conda install pytorch==1.12.1 torchvision==0.13.1 torchaudio==0.12.1 cpuonly -c pytorch
pip install matplotlib
```

(2) Python 包与预训练模型获取

按照下述代码进行 **Python** 包的引用，预训练模型采用提供的模型即可，自定义路径。

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms

pretrained_model = "./weights/lenet_mnist_model.pth" # 预训练模型
use_cuda = False # 是否使用cuda
```

其他可能使用的 **Python** 包：

```
import matplotlib.pyplot as plt
import numpy as np
```

(3) 被攻击模型搭建

按照如下代码实现被攻击模型，本实验采用分类网络 **LeNet** 作为被攻击模型。

```

11  # 定义LeNet网络
12  # LeNet Model definition
    4 usages
13  class Net(nn.Module):
14      def __init__(self):
15          super(Net, self).__init__()
16          self.conv1 = nn.Conv2d(1, 32, 3, 1)
17          self.conv2 = nn.Conv2d(32, 64, 3, 1)
18          self.dropout1 = nn.Dropout(0.25)
19          self.dropout2 = nn.Dropout(0.5)
20          self.fc1 = nn.Linear(9216, 128)
21          self.fc2 = nn.Linear(128, 10)
22
23      def forward(self, x):
24          x = self.conv1(x)
25          x = F.relu(x)
26          x = self.conv2(x)
27          x = F.relu(x)
28          x = F.max_pool2d(x, 2)
29          x = self.dropout1(x)
30          x = torch.flatten(x, 1)
31          x = self.fc1(x)
32          x = F.relu(x)
33          x = self.dropout2(x)
34          x = self.fc2(x)
35          output = F.log_softmax(x, dim=1)
36          return output

```

(4) FGSM 模块搭建

FGSM 模块输入包括原图像、epsilon 参数、梯度三个部分，输出为攻击后的图像。

```

# FGSM攻击代码
2 usages
def fgsm_attack(image, epsilon, data_grad):
    # 收集数据梯度的元素符号
    sign_data_grad = data_grad.sign()
    # 通过调整输入图像的每个像素来创建扰动图像
    perturbed_image = image + epsilon*sign_data_grad
    # 添加剪切以维持[0,1]范围
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # 返回被扰动的图像
    return perturbed_image

```

(5) PGD 模块搭建

PGD 模块输入包括迭代图像、alpha 参数、epsilon 参数、梯度四个部分，输出为攻击后的图像。

```
16 def pgd_attack(image, alpha, epsilon, data_grad):
17     sign_data_grad = torch.sign(data_grad)
18     perturbed_image = image + alpha * sign_data_grad
19     delta = torch.clamp(perturbed_image - image, min=-epsilon, max=epsilon)
20     perturbed_image = torch.clamp(image + delta, 0, 1)
21     return perturbed_image

34 elif attack == 'pgd':
35     data.requires_grad = True
36     data = data + 0.001 * torch.randn(data.shape)
37     data = torch.clamp(data, 0 - epsilon, 1 + epsilon)
38     for _ in range(step):
39         # 设置张量的requires_grad属性，这对于攻击很关键
40         # data.requires_grad = True
41         # 通过模型前向传递数据
42         output = model(data)
43         init_pred = output.max(1, keepdim=True)[1] # 获取初始预测结果
44         # 如果初始预测是错误的，不中断攻击，继续
45         if init_pred.item() != target.item():
46             continue
47         # 计算损失
48         loss = F.nll_loss(output, target)
49         # 将所有现有的渐变归零，作用是清除上一次的梯度
50         data_grad = torch.autograd.grad(loss, data, create_graph=True)[0]
51         # model.zero_grad()
52         # 计算后向传递模型的梯度，计算出各个参数的梯度
53         # loss.backward()
54         # 收集datagrad 为了攻击
55         # data_grad = data.grad.data
56         data = pgd_attack(data, alpha=0.01, epsilon=epsilon, data_grad=data_grad)
57     perturbed_data = data
```

(6) 测试函数构建

完成被攻击模型构建和攻击模块搭建后，需要通过测试函数将各部分进行整合，具体代码如下所示。

可通过修改 `attack` 变量调整攻击类型。当适用 FGSM 攻击方法时，梯度符号仅计算一次，无需迭代；当适用 PGD 攻击方法时，梯度会迭代计算多次，可通过改变 `step` 取值来改变攻击迭代次数。

```

6 def test(model, device, test_loader, epsilon, attack='pgd', step=1):
7     # 精度计数器
8     corrent = 0 # 正确的数量
9     adv_examples = [] # 存储攻击成功的样本
10    perturbed_data = []
11    # 循环遍历测试集中的所有示例
12    for data, target in test_loader:
13        # 将数据和标签发送到设备
14        data, target = data.to(device), target.to(device)
15        if attack == 'fgsm':
16            # 设置张量的requires_grad属性, 这对于攻击很关键
17            data.requires_grad = True
18            # 通过模型前向传递数据
19            output = model(data)
20            init_pred = output.max(1, keepdim=True)[1] # 获取初始预测结果
21            # 如果初始预测是错误的, 不打破攻击, 继续
22            if init_pred.item() != target.item():
23                continue
24            # 计算损失
25            loss = F.nll_loss(output, target)
26            # 将所有现有的渐变归零, 作用是清除上一次的梯度
27            model.zero_grad()
28            # 计算后向传递模型的梯度, 计算出各个参数的梯度
29            loss.backward()
30            # 收集datagrad 为了攻击
31            data_grad = data.grad.data
32            # 调用FGSM攻击
33            perturbed_data = fgsm_attack(data, epsilon, data_grad)
34        elif attack == 'pgd':
35            data.requires_grad = True
36            # output = model(data)
37            # data = data + 0.001 * torch.randn(data.shape)
38            # data = torch.clamp(data, 0 - epsilon, 1 + epsilon)
39            for _ in range(step):
40                # 设置张量的requires_grad属性, 这对于攻击很关键
41                # data.requires_grad = True
42                # 通过模型前向传递数据
43                output = model(data)
44                init_pred = output.max(1, keepdim=True)[1] # 获取初始预测结果
45                # 如果初始预测是错误的, 不打破攻击, 继续
46                if init_pred.item() != target.item():
47                    continue
48                # 计算损失
49                loss = F.nll_loss(output, target)
50                # 将所有现有的渐变归零, 作用是清除上一次的梯度
51                data_grad = torch.autograd.grad(loss, data, create_graph=True)[0]
52                # model.zero_grad()
53                # 计算后向传递模型的梯度, 计算出各个参数的梯度
54                # loss.backward()
55                # 收集datagrad 为了攻击
56                # data_grad = data.grad.data
57                data = pgd_attack(data, epsilon=epsilon, alpha=1, data_grad=data_grad)
58            perturbed_data = data
59            # perturbed_data = pgd_attack(data, epsilon, data_grad)
60
61    # 重新分类受扰乱的图像
62    output = model(perturbed_data)

```



```

61         # 重新分类受扰乱的图像
62         output = model(perturbed_data)
63
64         # 检查是否成功
65         final_pred = output.max(1, keepdim=True)[1] # 获取最终预测结果
66         if final_pred.item() == target.item():
67             corrent += 1
68             # 保存0 epsilon示例的特例
69             if (epsilon == 0) and (len(adv_examples) < 5):
70                 adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
71                 adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))
72             else: # 保存epsilon>0的样本
73                 if len(adv_examples) < 5:
74                     adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
75                     adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))
76
77         # 计算最终的正确率
78         final_acc = corrent / float(len(test_loader))
79         print("Epsilon:{}/\tTest Accuracy={}/\t={}".format(epsilon, corrent, len(test_loader), final_acc))
80
81         # 返回正确率和对抗样本
82         return final_acc, adv_examples

```

(7) 主程序构建

根据已构建模块，设计主程序代码，具体如下图所示。其中，epsilon 部分需要学生根据实际情况设置参数范围，在程序中用 python 列表表示即可。

```

10     epsilons = []
11     accuracies = []
12     examples = []
13
14     # MNIST数据集测试和加载
15     test_loader = torch.utils.data.DataLoader(
16         datasets.MNIST('./data', train=False, download=True, transform=transforms.Compose([
17             transforms.ToTensor(),
18         ])),
19         batch_size=1, shuffle=True)
20
21     # 查看是否配置GPU，没有就调用CPU
22     print("CUDA Available:", torch.cuda.is_available())
23     device = torch.device("cuda" if (use_cuda and torch.cuda.is_available()) else "cpu")
24     model = Net().to(device)
25     model.load_state_dict(torch.load(pretrained_model, map_location='cpu'))
26
27     # 对于每个epsilon，运行测试
28     for eps in epsilons:
29         acc, ex = test(model, device, test_loader, eps)
30         accuracies.append(acc)
31         examples.append(ex)

```

(8) 实验结果可视化

完成本实验主体代码部分的实现工作后，需要对实验结果进行记录，并绘制曲线和可视化分类结果，具体代码如下所示。

```

32 plt.figure(figsize=(5, 5))
33 plt.plot(epsilons, accuracies, "*-")
34 plt.yticks(np.arange(0, 1.1, step=0.1))
35 plt.xticks(np.arange(0, .35, step=0.05))
36 plt.title("Accuracy vs Epsilon") # 准确率与epsilon的关系
37 plt.xlabel("Epsilon")
38 plt.ylabel("Accuracy")
39 plt.show()
40
41 # 画出几个epsilon的示例
42 cnt = 0 # 计数器
43 plt.figure(figsize=(8, 10)) # 画布大小
44 for i in range(len(epsilons)): # 遍历epsilon
45     for j in range(len(examples[i])):
46         cnt += 1
47         plt.subplot(len(epsilons), len(examples[0]), cnt)
48         plt.xticks([], [])
49         plt.yticks([], [])
50         if j == 0: # 第一行的标题
51             plt.ylabel("Eps:{}".format(epsilons[i]), fontsize=14)
52             orig, adv, ex = examples[i][j] # 获取原始, 对抗, 样本
53             plt.title("{} -> {}".format(orig, adv), color=("green" if orig == adv else "red"), fontsize=14)
54             plt.imshow(ex, cmap="gray")
55 plt.tight_layout() # 自动调整子图参数, 使之填充整个图像区域
56 plt.show()

```

五、验收要求:

1. 分别完成 LeNet 分类模型在被 FGSM 和 PGD 方法攻击前后的分类精度的变化记录(以表格形式, FGSM 和 PGD 攻击参数自选)
2. 完成不同 epsilon 参数下 FGSM 攻击对 LeNet 分类模型影响的记录, 并绘制曲线、结果可视化(程序自带)
3. 完成不同 epsilon、alpha、迭代次数参数下 PGD 攻击对 LeNet 分类模型影响的记录, 并绘制曲线、结果可视化(程序自带)
4. 比较 FGSM 和 PGD 两种攻击方法的差异, 并通过实验结果说明各自对模型的影响程度