哈尔滨工业大学(深圳)

《网络与系统安全》
实验报告

实验七

对抗样本攻击 实验

学　院: 计算机科学与技术

姓　名: 覃煜淮

学　号: 220110803

专　业: 计算机类

日　期: 2025 年 4 月

PS：因为之后期末周任务较多，本人看着 lab7 不用依赖机房环境，于是便在本地提前完成了，采取了实验六的报告文档进行改正，保证包含了所有实现详细的内容，往老师对格式不一致的问题见谅

FGSM 实现（关于模型和攻击函数的实现）

1. 定义扰动数值，预训练模型参数路径以及使用设备情况

```
epsilons = [0, .05, .08, .1, .15, .2, .25]
pretrained_model = "data/lenet_mnist_model.pth"
use_cuda=False
```

2. 定义模型架构，初始化模型参数，初始化数据加载器

In [10]:
```
# LeNet Model definition
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

# MNIST Test dataset and dataloader declaration
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=False, download=True, transform=transforms.Compose([
            transforms.ToTensor(),
            ])),
        batch_size=1, shuffle=True)

# Define what device we are using
print("CUDA Available: ",torch.cuda.is_available())
device = torch.device("cuda" if (torch.cuda.is_available()) else "cpu")

# Initialize the network
model = Net().to(device)

# Load the pretrained model
model.load_state_dict(torch.load(pretrained_model, map_location='cpu'))

# Set the model in evaluation mode. In this case this is for the Dropout layers
model.eval()
```

```
CUDA Available:  False
```

Out[10]:
```
Net(
  (conv1): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
  (conv2_drop): Dropout2d(p=0.5, inplace=False)
  (fc1): Linear(in_features=320, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=10, bias=True)
)
```

3. 实现攻击函数（**要补全代码**）

In [11]:
```python
def fgsm_attack(image, epsilon, data_grad):
    # 收集数据梯度的元素级符号
    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
    # 添加剪切以保持 [0,1] 范围
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    # 返回扰动后的图像
    return perturbed_image
```

4. 结合模型，攻击函数以及基本配置，对模型攻击效果测试（**要补全代码**）：

In [12]:
```python
def test(model, device, test_loader, epsilon):
    # Accuracy counter
    correct = 0
    adv_examples = []

    # Loop over all examples in test set
    for data, target in test_loader:
        # Send the data and label to the device
        data, target = data.to(device), target.to(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(output, target)

        # Zero all existing gradients
        model.zero_grad()

        # Calculate gradients of model in backward pass
        loss.backward()

        # Collect data_grad
        data_grad = data.grad.data

        # Call FGSM Attack
        perturbed_data = fgsm_attack(data, epsilon, data_grad)

        # Re-classify the perturbed image
        output = model(perturbed_data)

        # 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
            # Special case for saving 0 epsilon examples
            if (epsilon == 0) and (len(adv_examples) < 5):
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))
        else:
            # Save some adversarial examples for visualization later
            if len(adv_examples) < 5:
                adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
                adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))

    # Calculate final accuracy for this epsilon
    final_acc = correct / float(len(test_loader.dataset))
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader.dataset), final_acc))

    # Return the accuracy and an adversarial example
    return final_acc, adv_examples
```

补全部分:

```
# Call FGSM Attack
perturbed_data = fgsm_attack(data, epsilon, data_grad)

# Re-classify the perturbed image
output = model(perturbed_data)

# 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
    # Special case for saving 0 epsilon examples
    if (epsilon == 0) and (len(adv_examples) < 5):
        adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
        adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))
else:
    # Save some adversarial examples for visualization later
    if len(adv_examples) < 5:
        adv_ex = perturbed_data.squeeze().detach().cpu().numpy()
        adv_examples.append((init_pred.item(), final_pred.item(), adv_ex))
```

5. 实施攻击并获得结果

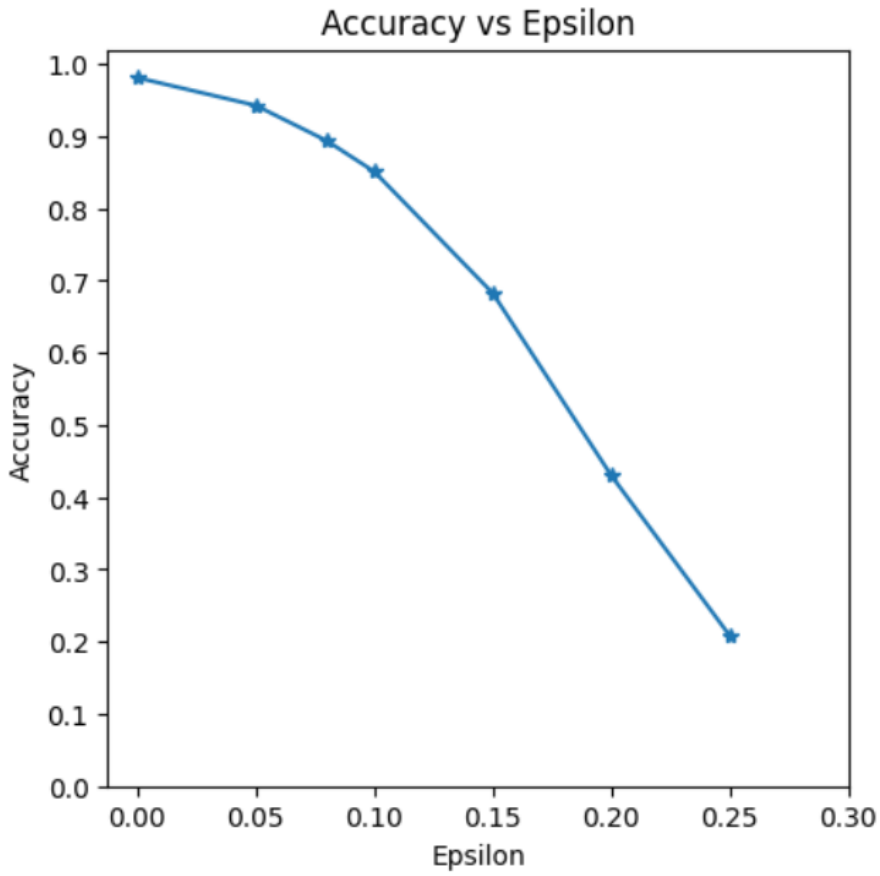
```
accuracies = []
examples = []

# Run test for each epsilon
for eps in epsilons:
    acc, ex = test(model, device, test_loader, eps)
    accuracies.append(acc)
    examples.append(ex)
```

```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.05   Test Accuracy = 9426 / 10000 = 0.9426
Epsilon: 0.08   Test Accuracy = 8936 / 10000 = 0.8936
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.15   Test Accuracy = 6826 / 10000 = 0.6826
Epsilon: 0.2    Test Accuracy = 4301 / 10000 = 0.4301
Epsilon: 0.25   Test Accuracy = 2082 / 10000 = 0.2082
```

结果分析

1. 准确性-扰动大小图

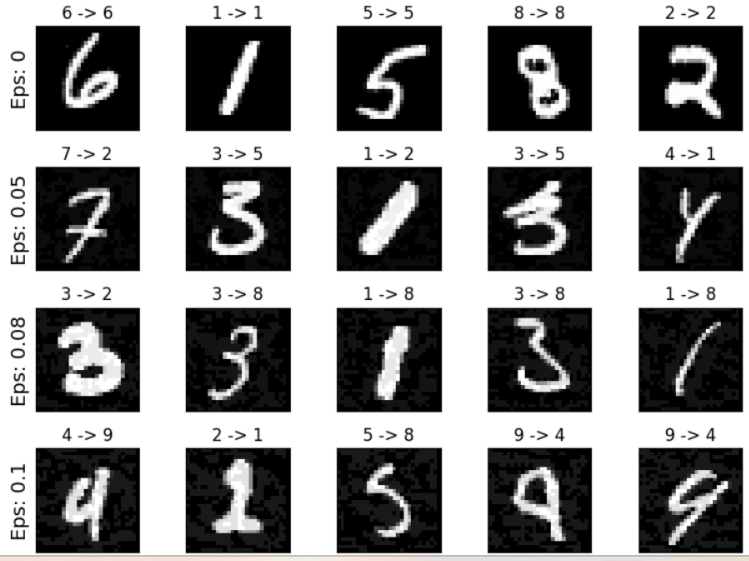
```
plt.figure(figsize=(5,5))
plt.plot(epsilons, accuracies, "*-")
plt.yticks(np.arange(0, 1.1, step=0.1))
plt.xticks(np.arange(0, .35, step=0.05))
plt.title("Accuracy vs Epsilon")
plt.xlabel("Epsilon")
plt.ylabel("Accuracy")
plt.show()
```

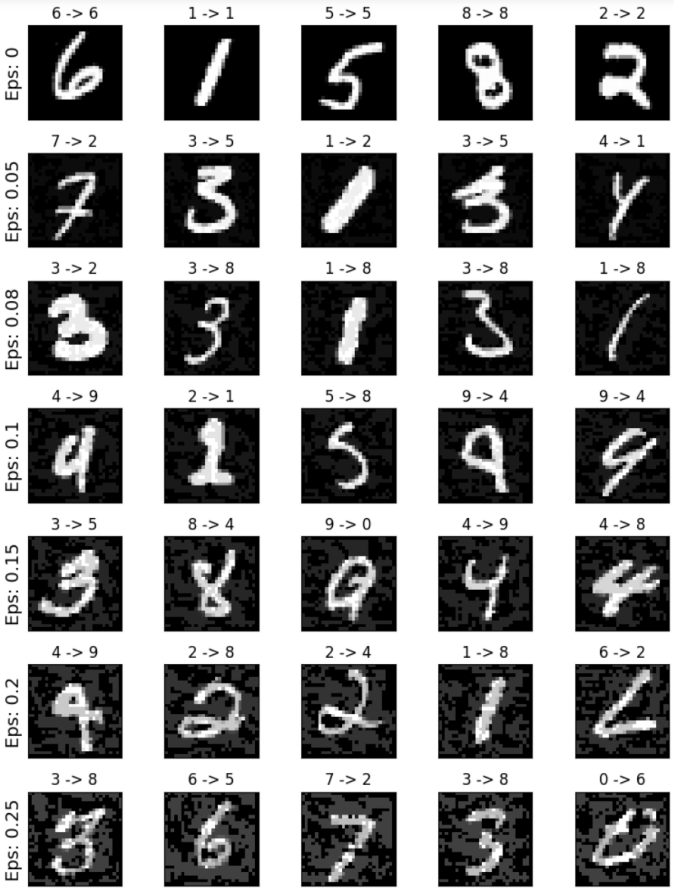


由图观察到，扰动越大，对图像的改变越大，模型识别的准确率越低，可以观察到，再没有进行对抗样本训练的时候，扰动大于 0.1 后模型准确率急剧下降

2. 对抗样本实例：

In [15]:
```
# Plot several examples of adversarial samples at each epsilon
cnt = 0
plt.figure(figsize=(8,10))
for i in range(len(epsilons)):
    for j in range(len(examples[i])):
        cnt += 1
        plt.subplot(len(epsilons),len(examples[0]),cnt)
        plt.xticks([], [])
        plt.yticks([], [])
        if j == 0:
            plt.ylabel("Eps: {}".format(epsilons[i]), fontsize=14)
        orig,adv,ex = examples[i][j]
        plt.title("{} -> {}".format(orig, adv))
        plt.imshow(ex, cmap="gray")
plt.tight_layout()
plt.show()
```





任选一个扩展任务完成

本人选择了任务 2:

> 任务2：防御对抗样本攻击的一个方法是将对抗样本加入数据集中再进行训练，增强训练模型的鲁棒性，请给出你的解决方案、代码和测试结果。

实现代码截图如下:

库的导入以及一些基本参数设置:

```
from __future__ import print_function
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import numpy as np
import matplotlib.pyplot as plt

# NOTE: This is a hack to get around "User-agent" limitations when downloading MNIST datasets
from six.moves import urllib
opener = urllib.request.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0')]
urllib.request.install_opener(opener)

epsilons = [0, .05, .08, .1, .15, .2, .25]
pretrained_model = "data/lenet_mnist_model.pth"
use_cuda = False
```

模型架构，模型权重，模型存储设备以及数据加载器设置:

```
# LeNet Model definition
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)

# MNIST Train and Test dataset and dataloader declaration
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=True, download=True, transform=transforms.Compose([
            transforms.ToTensor(),
            ])),
        batch_size=64, shuffle=True)

test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('./data', train=False, transform=transforms.Compose([
            transforms.ToTensor(),
            ])),
        batch_size=1000, shuffle=True)

# Define what device we are using
print("CUDA Available: ", torch.cuda.is_available())
device = torch.device("cuda" if (torch.cuda.is_available()) else "cpu")

# Initialize the network
model = Net().to(device)

# Load the pretrained model if available
try:
    model.load_state_dict(torch.load(pretrained_model, map_location='cpu'))
except FileNotFoundError:
    print("Pretrained model not found. Starting from scratch.")
```

优化器，损失函数，训练函数（在训练中实时生成对抗样本参与训练），测试效果函数定义：

```python
# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# FGSM attack function
def fgsm_attack(image, epsilon, data_grad):
    sign_data_grad = data_grad.sign()
    perturbed_image = image + epsilon * sign_data_grad
    perturbed_image = torch.clamp(perturbed_image, 0, 1)
    return perturbed_image

# Adversarial training function
def adversarial_train(model, device, train_loader, optimizer, epsilon):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()

        # Forward pass on original data
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        # Generate adversarial examples
        data.requires_grad = True
        output = model(data)
        loss = F.nll_loss(output, target)
        model.zero_grad()
        loss.backward()
        data_grad = data.grad.data
        perturbed_data = fgsm_attack(data, epsilon, data_grad)

        # Forward pass on perturbed data
        optimizer.zero_grad()
        output = model(perturbed_data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

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

```python
# Test function
def test(model, device, test_loader, epsilon):
    model.eval()
    test_loss = 0
    correct = 0
    adv_examples = []

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

    # Test adversarial examples
    model.eval()
    correct = 0
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        data.requires_grad = True
        output = model(data)
        loss = F.nll_loss(output, target)
        model.zero_grad()
        loss.backward()
        data_grad = data.grad.data
        perturbed_data = fgsm_attack(data, epsilon, data_grad)
        output = model(perturbed_data)
        pred = output.max(1, keepdim=True)[1]
        correct += pred.eq(target.view_as(pred)).sum().item()

    final_acc = correct / float(len(test_loader.dataset))
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader.dataset), final_acc))

    return final_acc, adv_examples
```

指定训练超参数，保存模型权重，可视化训练结果：

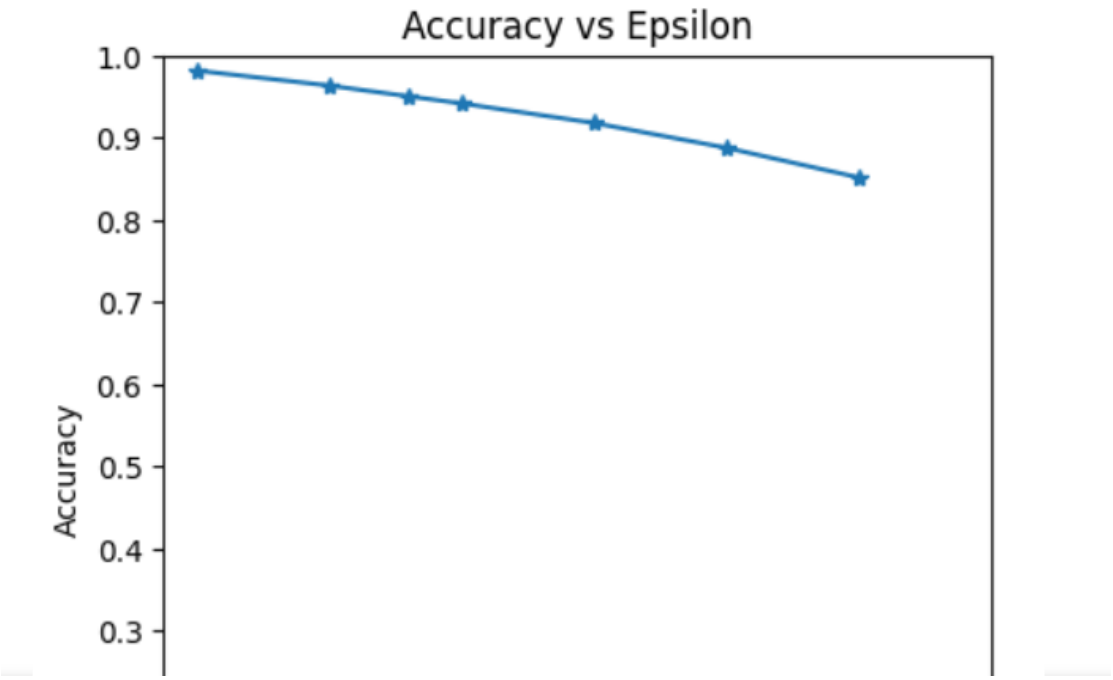
```python
# Test function
def test(model, device, test_loader, epsilon):
    model.eval()
    test_loss = 0
    correct = 0
    adv_examples = []

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
            output = model(data)
            test_loss += F.nll_loss(output, target, reduction='sum').item()
            pred = output.max(1, keepdim=True)[1]
            correct += pred.eq(target.view_as(pred)).sum().item()

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

    # Test adversarial examples
    model.eval()
    correct = 0
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        data.requires_grad = True
        output = model(data)
        loss = F.nll_loss(output, target)
        model.zero_grad()
        loss.backward()
        data_grad = data.grad.data
        perturbed_data = fgsm_attack(data, epsilon, data_grad)
        output = model(perturbed_data)
        pred = output.max(1, keepdim=True)[1]
        correct += pred.eq(target.view_as(pred)).sum().item()

    final_acc = correct / float(len(test_loader.dataset))
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(epsilon, correct, len(test_loader.dataset), final_acc))

    return final_acc, adv_examples
```

训练部分过程展示:

```
Epoch: 1
Train Batch: 0 [0/60000 (0%)]   Loss: 2.240336
Train Batch: 100 [6400/60000 (11%)]     Loss: 1.455251
Train Batch: 200 [12800/60000 (21%)]    Loss: 1.085634
Train Batch: 300 [19200/60000 (32%)]    Loss: 1.416857
Train Batch: 400 [25600/60000 (43%)]    Loss: 1.050454
Train Batch: 500 [32000/60000 (53%)]    Loss: 1.169601
Train Batch: 600 [38400/60000 (64%)]    Loss: 1.093140
Train Batch: 700 [44800/60000 (75%)]    Loss: 1.303844
Train Batch: 800 [51200/60000 (85%)]    Loss: 1.044737
Train Batch: 900 [57600/60000 (96%)]    Loss: 1.050473

Test set: Average loss: 0.0805, Accuracy: 9767/10000 (98%)

Epsilon: 0.3    Test Accuracy = 6336 / 10000 = 0.6336
```

```
Epoch: 4
Train Batch: 0 [0/60000 (0%)]   Loss: 0.742749
Train Batch: 100 [6400/60000 (11%)]     Loss: 0.738295
Train Batch: 200 [12800/60000 (21%)]    Loss: 0.885134
Train Batch: 300 [19200/60000 (32%)]    Loss: 0.779379
Train Batch: 400 [25600/60000 (43%)]    Loss: 0.750184
Train Batch: 500 [32000/60000 (53%)]    Loss: 0.571048
Train Batch: 600 [38400/60000 (64%)]    Loss: 0.676215
Train Batch: 700 [44800/60000 (75%)]    Loss: 0.605601
Train Batch: 800 [51200/60000 (85%)]    Loss: 0.429917
Train Batch: 900 [57600/60000 (96%)]    Loss: 0.696044

Test set: Average loss: 0.0648, Accuracy: 9816/10000 (98%)

Epsilon: 0.3    Test Accuracy = 7798 / 10000 = 0.7798
```

多个扰动训练后在测试上的可视化结果:



观察可知，加入对抗样本进行训练能有效提升模型对于对抗样本的识别能力，能有效提高模型对于对抗攻击的防御能力

二、遇到问题及解决方法

问题：对于对抗样本训练不知道如何产生对抗样本，何时用对抗样本作为数据参与训练

解决方法：查阅相关资料，咨询 AI 确立训练框架，两者结合解决对抗样本训练问题

三、对本次实验的建议

本次实验是唯一一个不用依赖于机房环境能够在本地环境很简单上手操作的实验，并且每一个实验也都只需要花课内时间便能完成，同时防火墙，SQL 注入 XSS 蠕虫攻击等实验都很能增长编码以及相关知识的兴趣，是一门很好的课程