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:　计算机科学与技术学院

姓　名:　刘睿

学　号:　220110720

专　业:　计算机科学与技术

日　期:　　2025 年 4 月

一、实验过程

每个实验步骤（供有 4 个任务）要求有具体截图和说明，并回答相关的问题

1. 完成**所有需要补充的**代码，并截图说明代码的含义和作用。

fgsm_attack()

```python
import torch

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
```

计算损失函数关于图像的梯度的符号，接着按照 FGSM 方法生成扰动的图像，并将像素值裁剪回[0, 1]范围内保证图像合法

test()

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

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

生成扰动后的图像并对其尝试进行分类

```python
        # 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 adv 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))
```

对于确实扰动后预测失败的图像，对每个 epsilon 下的前五张进行保存

2. 分析 **4.4 测试攻击效果函数** 的代码部分，说明每段代码的作用。

test()

```
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:

```

correct 记录正确数量之后计算 Acc

adv_example 列表保存预测失败的例子，之后可视化

对 test_loader 进行迭代

```
    # 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 ``datagrad``
        data_grad = data.grad.data
```

获取这一批 test 数据的梯度，用于之后计算扰动图片

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

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

利用获得的梯度计算扰动图片，并使用现有模型进行预测

```python
        # 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 adv 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))
```

将预测的结果（概率）转变为离散的标记，并判断是否预测正确。若正确，correct 计数器加一。对于 episode=0（即无扰动的情况下），我们展示预测正确的前五张。若预测错误，correct 不用动，记录错误的前五张

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

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

计算 acc 并返回 acc 和用于可视化的几组图

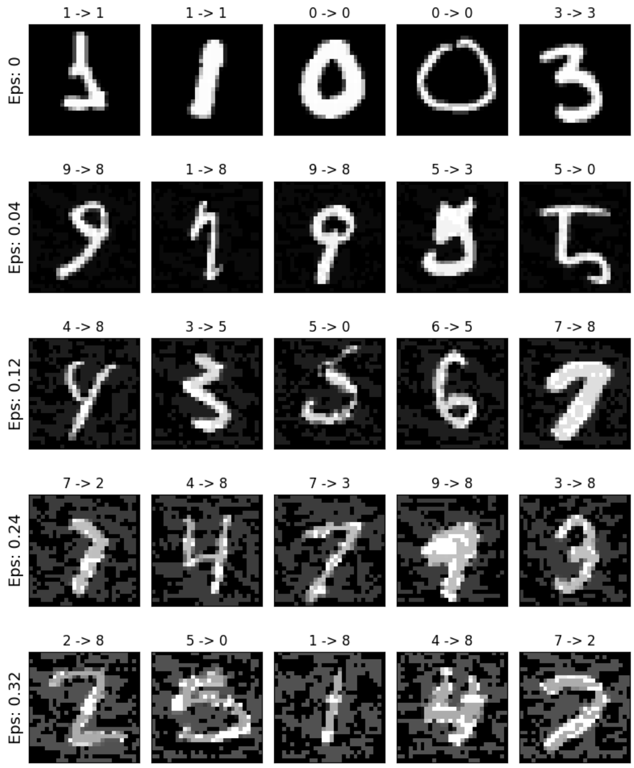
3. 分别对 默认给出的 epsilons = [0, .05, .08,.1, .15, .2, .25] 和自行修改的 epsilons 执行结果进行截图，并做简要说明。

```
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
```

```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.04   Test Accuracy = 9523 / 10000 = 0.9523
Epsilon: 0.12   Test Accuracy = 7943 / 10000 = 0.7943
Epsilon: 0.24   Test Accuracy = 2458 / 10000 = 0.2458
Epsilon: 0.32   Test Accuracy = 603 / 10000 = 0.0603
```

随着 epsilon 的增加，acc 不断下降。尤其是 epsilon 超过 0.15 之后，强烈的扰动让机器难以分辨图像。在 epsilon 达到 0.3 左右的时候，分类器和随机预测的效果并无差别。观察下面的可视化图像可以看出，在如此强烈的扰动下，虽然人眼还能分辨笔迹，但很多数字的重要特征已被噪音淹没或者和别的数字混杂了


Eps: 0
8 -> 8
3 -> 3
9 -> 9
9 -> 9
9 -> 9
Eps: 0.05
1 -> 3
3 -> 8
8 -> 5
8 -> 9
9 -> 4
Eps: 0.08
7 -> 9
1 -> 3
4 -> 1
2 -> 8
9 -> 5
Eps: 0.1
9 -> 4
8 -> 3
3 -> 2
7 -> 2
7 -> 9
Eps: 0.15
6 -> 5
8 -> 6
2 -> 1
3 -> 8
6 -> 5
Eps: 0.2
6 -> 5
6 -> 8
9 -> 8
9 -> 4
7 -> 2
Eps: 0.25
2 -> 6
5 -> 2
3 -> 8
8 -> 2
2 -> 8


Eps: 0
1 -> 1
1 -> 1
0 -> 0
0 -> 0
3 -> 3
Eps: 0.04
9 -> 8
1 -> 8
9 -> 8
5 -> 3
5 -> 0
Eps: 0.12
4 -> 8
3 -> 5
5 -> 0
6 -> 5
7 -> 8
Eps: 0.24
7 -> 2
4 -> 8
7 -> 3
9 -> 8
3 -> 8
Eps: 0.32
2 -> 8
5 -> 0
1 -> 8
4 -> 8
7 -> 2

4. 任意选择一个扩展任务将完成的解决方案、代码和测试结果进行说明。

选择防御对抗样本攻击，这意味着我们需要生成对抗样本和其正确的 label 对，重新投入并训练我们的模型

```
retrained_model_path = 'data/retrained_model.pth'

retrained_model = Net().to(device)

retrained_model.load_state_dict(torch.load(retrained_model_path, map_location='cpu'))
```

重新加载一个网络的参数，这个保存的网络是从同样的 pretrained model 上开始训练的，只是后来我迭代训练了几次，就用新的保存的模型参数了

```
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)
        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)

        combined_data = torch.cat([data.detach(), perturbed_data.detach()], dim=0)
        combined_target = torch.cat([target, target], dim=0)

        output = model(combined_data)
        loss = F.nll_loss(output, combined_target)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch_idx % 100 == 0:
            print(f'Train Batch: {batch_idx}\tLoss: {loss.item():.6f}')
```

类比之前的 test()，我们需要加载 dataloader 中的数据对，首先在模型上得到梯度，计算出对抗样本，并和原本的样本合并，标签即为原本样本的标签自己 concatenate 一次，重新投入训练

```python
mnist_mean = 0.1307
mnist_std = 0.3081

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST(
        './data', train=True, download=True,
        transform=transforms.Compose([
            transforms.ToTensor(),
            transforms.Normalize((mnist_mean,), (mnist_std,))
        ])
    ),
    batch_size=64, shuffle=True
)
```

```python
import torch.optim as optim


optimizer = optim.Adam(retrained_model.parameters(), lr=0.001)
```

```python
for eps in epsilons:
    adversarial_train(retrained_model, device, train_loader, optimizer, eps)
```

准备好训练集和 optimizer，对每个 episode 进行训练。因为我想随时手动停止训练和观察数据就没有设置 epoch，而是自己手动迭代

```python
accuracies = []
examples = []

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

```
Epsilon: 0      Test Accuracy = 9496 / 10000 = 0.9496
Epsilon: 0.05   Test Accuracy = 9215 / 10000 = 0.9215
Epsilon: 0.08   Test Accuracy = 8886 / 10000 = 0.8886
Epsilon: 0.1    Test Accuracy = 8766 / 10000 = 0.8766
Epsilon: 0.15   Test Accuracy = 7949 / 10000 = 0.7949
Epsilon: 0.2    Test Accuracy = 7017 / 10000 = 0.7017
Epsilon: 0.25   Test Accuracy = 5945 / 10000 = 0.5945
```

用同样的方式查看 test 效果，可以看到，对轻扰动和无扰动的分类效果略有下降，但是对于强扰动后的图像分类正确率有明显上升。虽然这个正确率并未达到一个很好的结果，但这告诉我们我们有办法通过进一步的工作，如数据增强等方法来达到更好的分类效果

二、实验中遇到的问题和解决方法

实验比较简单，并没有遇到什么问题

三、本次实验的收获和建议

实验很有趣，虽然接触过很多神经网络图像分类任务，但第一次体验对抗样本攻击。从对抗样本的攻防中我进一步理解的数据的重要性，给我在数据预处理方面的启发