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: 房煊梓

学 号: 210010101

专 业: 智能强基-计算机

日 期: 2024 年 6 月 1 日

一、本次实验要求

1. 完成**所有需要补充的**代码，并截图说明。

(1)FGSM 攻击函数:

```
# 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(perturbed_image,0,1)
    # Return the perturbed image
    return perturbed_image
```

首先通过 sign 函数来收集数据梯度的元素符号。然后对每个像素加上 epsilon*sign_data_grad 来调整输入图像的每个像素，从而创建扰动图像。再通过 torch.clamp 函数添加剪切来维持[0,1]范围。最后返回被扰动的图像。

(2)test 函数:

```
# 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 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) )
```

调用(1)中完成的 fgsm_attack 函数来进行攻击，获取被扰动的图像。然后重新分类受到扰动的图像。检查是否成功的同时，仿照保存 0 episilon 样本的方法，对于其他满足 len<5 的样本进行保存，用于后续的可视化。

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。

首先设置精度计算器。

```
# Accuracy counter
correct = 0
adv_examples = []
```

然后使用 for 循环，遍历测试集中的所有样本:

```
# Loop over all examples in test set
for data, target in test_loader:
```

将数据和标签发给设备。

```
# Send the data and label to the device
data, target = data.to(device), target.to(device)
```

设置张量的 required_grad 属性。

```
# Set requires_grad attribute of tensor. Important for Attack
data.requires_grad = True
```

通过模型前向传递数据，获取最大对数概率的索引。

```
# 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
```

若初始预测错误，则不打断攻击，而是继续。

```
# If the initial prediction is wrong, don't bother attacking, just move on
if init_pred.item() != target.item():
    continue
```

计算损失。

```
# Calculate the loss
loss = F.nll_loss(output, target)
```

将所有已经存在的梯度置为 0。

```
# Zero all existing gradients
model.zero_grad()
```

将损失反向回传。

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

收集 datagrad。

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

调用 fgsm_attack 函数来进行攻击，获取被扰动的图像。

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

重新分类受到扰动的图像。

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

检查是否成功的同时，仿照保存 0 episilon 样本的方法，对于其他满足 len<5 的样本进行保存，用于后续的可视化。

```
# 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) )
```

循环结束后，计算该 epsilon 下最终的准确率并打印。

```
# 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))
```

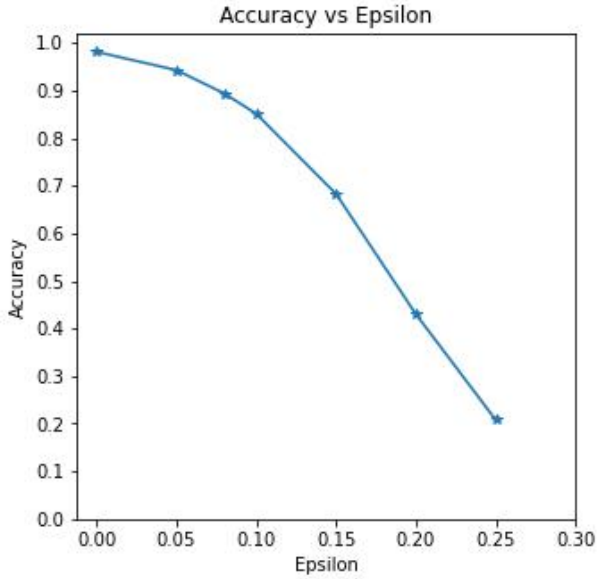
返回准确率和对抗样本。

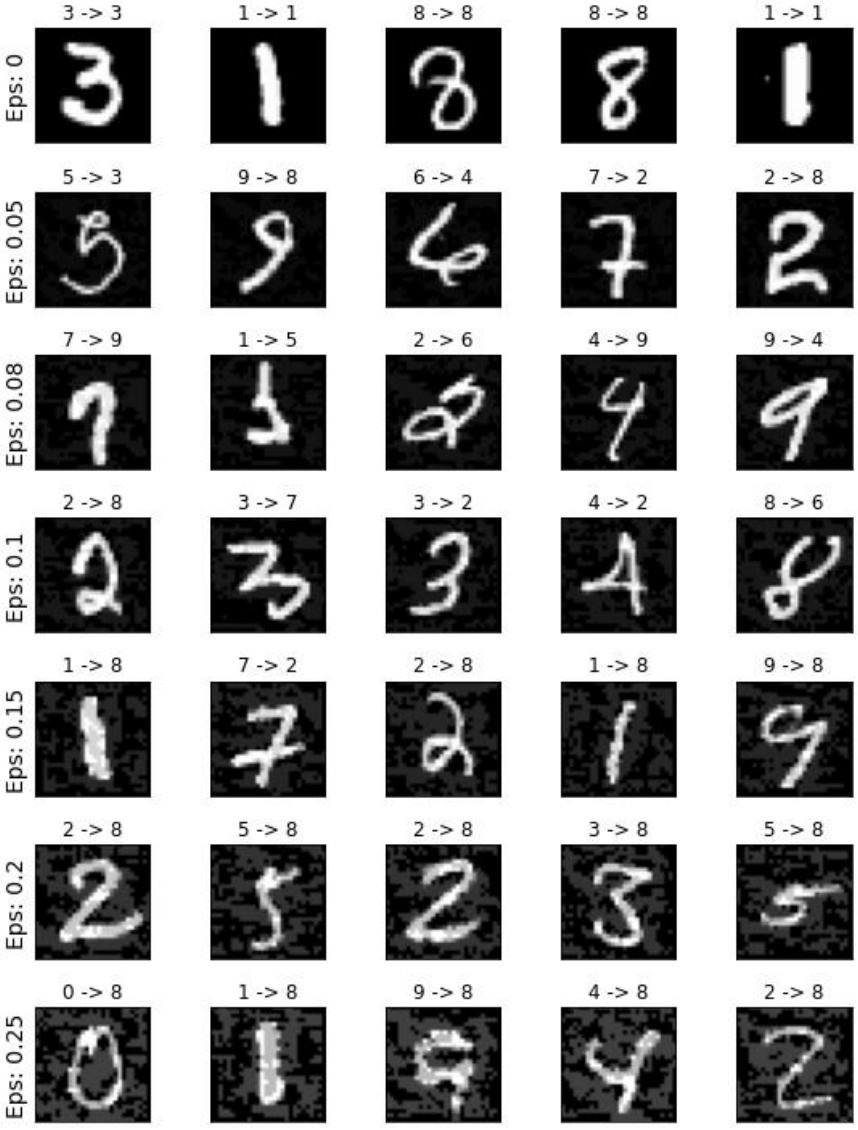
```
# Return the accuracy and an adversarial example
return final_acc, adv_examples
```

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。

(1)默认给出的 epsilons = [0, .05, .08,.1, .15, .2, .25]结果如下：

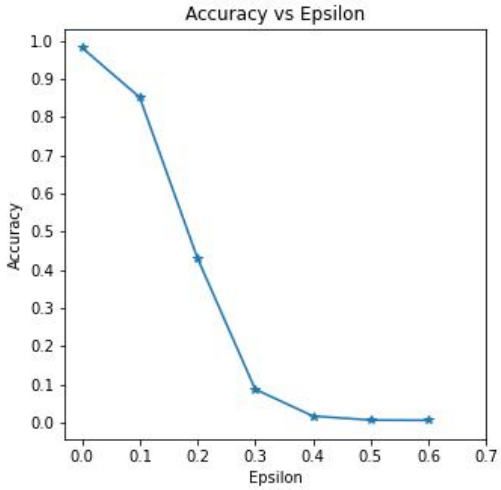
```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.05   Test Accuracy = 9426 / 10000 = 0.9426
Epsilon: 0.08   Test Accuracy = 8936 / 10000 = 0.8936
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.15   Test Accuracy = 6826 / 10000 = 0.6826
Epsilon: 0.2    Test Accuracy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

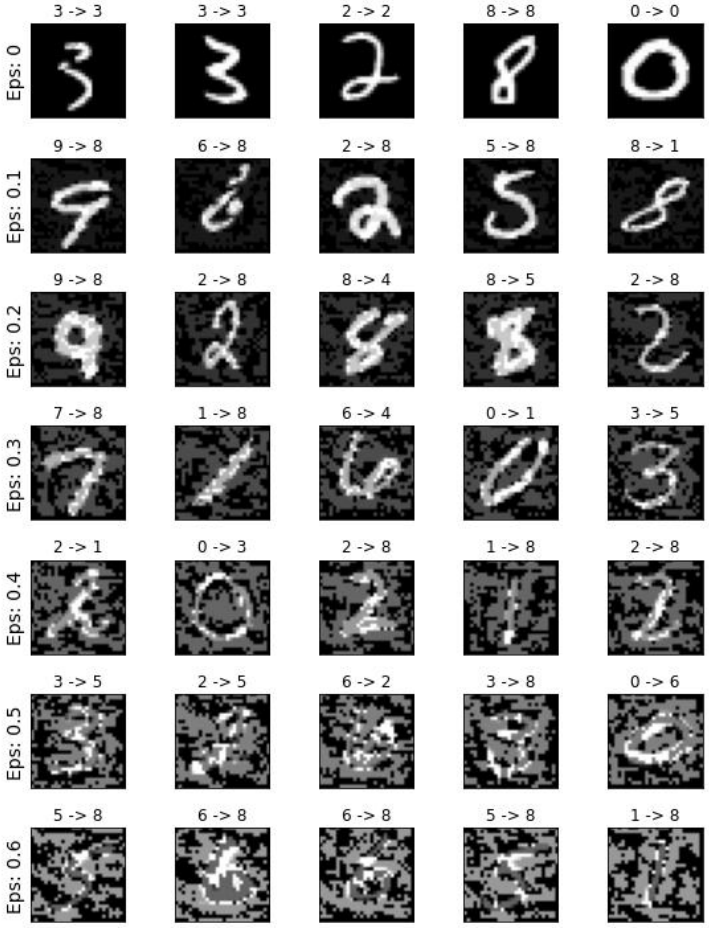

Accuracy vs Epsilon
Accuracy
Epsilon
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.00
0.05
0.10
0.15
0.20
0.25
0.30


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

(2)自行修改的 epsilons = [0, .1, .2, .3, .4, .5, .6]结果如下:

```
Epsilon: 0      Test Accuracy = 9810 / 10000 = 0.981
Epsilon: 0.1    Test Accuracy = 8510 / 10000 = 0.851
Epsilon: 0.2    Test Accuracy = 4301 / 10000 = 0.4301
Epsilon: 0.3    Test Accuracy = 869 / 10000 = 0.0869
Epsilon: 0.4    Test Accuracy = 167 / 10000 = 0.0167
Epsilon: 0.5    Test Accuracy = 63 / 10000 = 0.0063
Epsilon: 0.6    Test Accuracy = 56 / 10000 = 0.0056
```





简要说明：由默认和自行修改的 epsilons 对应的结果可知：

①准确率方面：当 episilon 为 0 时，准确率最高，为 0.981；随着 epsilon 的增大，模型的准确率在降低；epsilon 从 0 到 0.3 时，模型的准确率降低速度较快；当 epsilon 为 0.3 时，模型准确率小于 0.1；epsilon 从 0.3 到 0.4 时，模型的准确率降低速度较慢；epsilon 从 0.4 到 0.6 时，模型的准确率降低速度非常慢；当 epsilon 为 0.6 时，准确率最低，为 0.0056。

②受扰动的图像方面：当 epsilon 从 0 到 0.1 时，受到扰动的图像仍较为清晰；当 epsilon 从 0.2 到 0.4 时，受到扰动的图像有些模糊，但是人眼还能进行辨认；当 epsilon 从 0.5 到 0.6 时，受到扰动的图像非常模糊，人眼辨认难度较大。

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。

选择扩展任务 1.

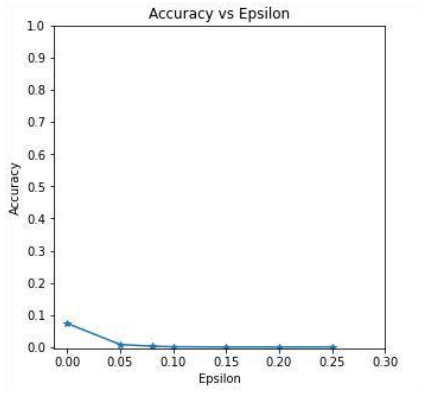
解决方案：用 FashionMNIST 数据集和原始模型进行测试，epsilons 采用默认值。

代码：相对于原来的代码，修改测试集，修改部分的代码截图如下：

```
transform = transforms.Compose([
    transforms.ToTensor()
])
# 下载并加载测试集
testset = datasets.FashionMNIST('~/.pytorch/F_MNIST_data/', download=True, train=False, transform=transform)
test_loader = torch.utils.data.DataLoader(testset, batch_size=1, shuffle=True)
```

结果如下。

```
Epsilon: 0      Test Accuracy = 736 / 10000 = 0.0736
Epsilon: 0.05   Test Accuracy = 80 / 10000 = 0.008
Epsilon: 0.08   Test Accuracy = 28 / 10000 = 0.0028
Epsilon: 0.1    Test Accuracy = 11 / 10000 = 0.0011
Epsilon: 0.15   Test Accuracy = 1 / 10000 = 0.0001
Epsilon: 0.2    Test Accuracy = 3 / 10000 = 0.0003
Epsilon: 0.25   Test Accuracy = 4 / 10000 = 0.0004
```



简要说明：由以上结果可知：

①准确率方面：当 epsilon 为 0 时准确率最高，为 0.0736，没有超过 0.1，其他准确率更是没有超过 0.01，可见整体效果极差。

②受扰动的图像方面：当 epsilon 从 0 到 0.05 时，受到扰动的图像仍较为清晰；当 epsilon 从 0.08 到 0.2 时，受到扰动的图像有些模糊，但是人眼还能进行辨认；当 epsilon 为 0.25 时，受到扰动的图像非常模糊，人眼辨认难度较大。

二、网络与信息安全实验课程的收获和建议（**必填部分**）

*（关于本学期网络与系统实验的三个部分：系统安全，网络安全和 AI 安全，请给出您对于这三部分实验的收获与体会，给出评论以及**改进的建议**。）*

对于系统安全部分，实验指导书中的步骤较为详细，跟着指导书能够一步步完成实验并且在完成的过程中逐步理解相关的知识。对于这部分的建议是可以增加一些正确结果参考，方便同学们检查自己的实验结果。

对于网络安全部分，感觉这部分的实验涉及的知识很广泛，实验原理部分的解释很详细，实验的各个步骤也是循序渐进的，让人实实在在地学到很多相关知识。对于这部分的建议是对于一些新出现的工具进行更详细的说明，比如 PKI 实验的添加证书操作，刚接触的时候不知道从哪里进入相关界面而有点耽误实验流程的推进，如果能进行说明的话可以避免这种情况。还可以增加一些对于大多数同学可能会遇到的错误的说明，这样同学们可以根据说明自行纠错。

对于 AI 安全部分，这部分的实验内容相对前两个部分来说较为简单，实验过程遇到的问题多为实验环境方面的问题，比如提示 torch 未安装等，其他方面的问题不大。对于这部分的建议是可以增加扩展任务的选择，比如选择不同的模型进行对比。