adv_cv 实验报告

鲁棒性与攻防 | | CV

本次实验首先做到的是训练一个我们熟悉的训练集--MNIST的分类器， 这里用到了卷积神经网络

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1) #这里用到了两层神经网络
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)#最后一层为输出层， 因为数据集是0-9十个数字所以第二个参数是10

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
class MobileNet(nn.Module):
    def __init__(self):
        super(MobileNet, self).__init__()
        net = models.mobilenet_v3_small(pretrained=False)#这里使用mobileNet训练的分类器

        self.trunk = nn.Sequential(*(list(net.children())[:-2]))


        self.avg_pool = nn.AdaptiveAvgPool2d(output_size=1)
        self.fc = nn.Sequential(
            nn.Linear(576, 10, bias=True),
            nn.LogSoftmax(dim=1)
        )


    def forward(self, x):

        x = self.trunk(x)
```

```
        x = self.avg_pool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

这里我们可以先测试一下我们训练的分类器效果如何

```
def main():
    # Training settings

    no_cuda = False
    seed = 1111
    batch_size = 128
    test_batch_size = 1000
    lr = 0.01
    save_model = True
    epochs = 2 #这里是训练了两轮， 后面可以从实验现象中看出来。

    use_cuda = not no_cuda and torch.cuda.is_available()

    torch.manual_seed(seed)

    device = torch.device("cuda" if use_cuda else "cpu")

    train_kwargs = {'batch_size': batch_size,'shuffle':True}
    test_kwargs = {'batch_size': test_batch_size,'shuffle':True}
    if use_cuda:
        cuda_kwargs = {'num_workers': 1,
                       'pin_memory': True,
                       'shuffle': False}
        train_kwargs.update(cuda_kwargs)
        test_kwargs.update(cuda_kwargs)

    transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])
    dataset1 = datasets.ImageFolder('mnist/training',transform=transform) #将
MNIST分成了训练集和测试集
    dataset2 = datasets.ImageFolder('mnist/testing',transform=transform)
    train_loader = torch.utils.data.DataLoader(dataset1,**train_kwargs,
num_workers=8)
    test_loader = torch.utils.data.DataLoader(dataset2, **test_kwargs)

    model = Net().to(device)# 第一行是用神经网络训练， 第二行是用 MobileNet训练
    #model = MobileNet().to(device)

    optimizer = optim.SGD(model.parameters(), momentum=0.9, lr=lr)

    scheduler = StepLR(optimizer, step_size=3, gamma=0.1)
    for epoch in range(1, epochs + 1):
        train( model, device, train_loader, optimizer, epoch)
        test(model, device, test_loader)
        scheduler.step()

    if save_model:
        torch.save(model.state_dict(), "mnist_cnn.pt")#保存模型
```

```
        #torch.save(model.state_dict(), "mnist_mobile.pt")


    return model
```

可以看一下运行结果:

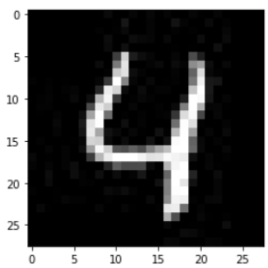
```
In  [4]: model = main()

Train Epoch: 1 [0/60000 (0%)]	Loss: 2.306674
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.408554
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.187409
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.207236
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.153625

Test set: Average loss: 0.0767, Accuracy: 9753/10000 (98%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.065316
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.073221
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.113927
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.138934
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.134654

Test set: Average loss: 0.0450, Accuracy: 9847/10000 (98%)
```

这是神经网络的训练结果， 可以出来正确率达到了98%，

```
In  [4]: model = main()

Train Epoch: 1 [0/60000 (0%)]	Loss: 2.370431
Train Epoch: 1 [12800/60000 (21%)]	Loss: 0.268072
Train Epoch: 1 [25600/60000 (43%)]	Loss: 0.138775
Train Epoch: 1 [38400/60000 (64%)]	Loss: 0.065683
Train Epoch: 1 [51200/60000 (85%)]	Loss: 0.113999

Test set: Average loss: 1.9499, Accuracy: 2251/10000 (23%)

Train Epoch: 2 [0/60000 (0%)]	Loss: 0.050647
Train Epoch: 2 [12800/60000 (21%)]	Loss: 0.070985
Train Epoch: 2 [25600/60000 (43%)]	Loss: 0.073266
Train Epoch: 2 [38400/60000 (64%)]	Loss: 0.073329
Train Epoch: 2 [51200/60000 (85%)]	Loss: 0.123693

Test set: Average loss: 0.0818, Accuracy: 9733/10000 (97%)
```

这是MobelNet的训练结果， 可以看出虽然第一轮训练结果很差， 但是第二轮仍然达到了97%的正确率。我们可以更直观地看一下他们的训练结果， 从数据集中选取一张4的图片， 两个分类器都输出了4这个结果。

```
Out[5]: <matplotlib.image.AxesImage at 0x2619c031a48>
```




```
In  [6]:
def cnn_eval(tensor):
    model=Net()
    model.load_state_dict(torch.load("mnist_cnn.pt"))
    model.eval()
    print(torch.argmax(model(tensor)))

def mobile_eval(tensor):
    model=MobileNet()
    model.load_state_dict(torch.load("mnist_mobile.pt"))
    model.eval()
    print(torch.argmax(model(tensor)))

mobile_eval(norm(four_tensor))
cnn_eval(norm(four_tensor))

tensor(4)
tensor(4)
```

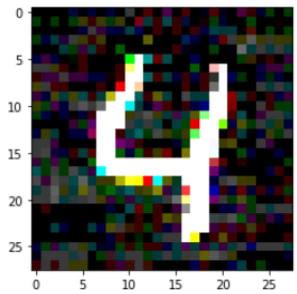
对抗样本:

这里生成了对抗样本

```
delta = torch.zeros_like(four_tensor, requires_grad=True)
opt = optim.SGD([delta], lr=10)
epsilon = 0.2
for t in range(20):
    pred = model(norm(four_tensor + delta))
    #一个样本本来是为了最小化loss以达到正确率
    loss = -nn.CrossEntropyLoss()(pred, torch.LongTensor([gt]))#这里loss取反, 让loss越来越大


    opt.zero_grad()
    loss.backward()
    opt.step()
    delta.data.clamp_(-epsilon, epsilon)
```

所以这里可能他不会被辨认为4,

```
tensor(-19.0494, grad_fn=<NegBackward0>)
tensor(4)
None

<matplotlib.image.AxesImage at 0x1bc9302ddc8>
```



虽然我这里还是被辨认为4了（可能是因为神经网络比较强大）， 但是肉眼很容易看出这个图片加入了扰动。

target attack

除了加入扰动让数字本身不被辨认以外， 甚至可以让数字被定向辨认为一个错误的数字， 甚至将一个不相关的图片辨认成一个数字。

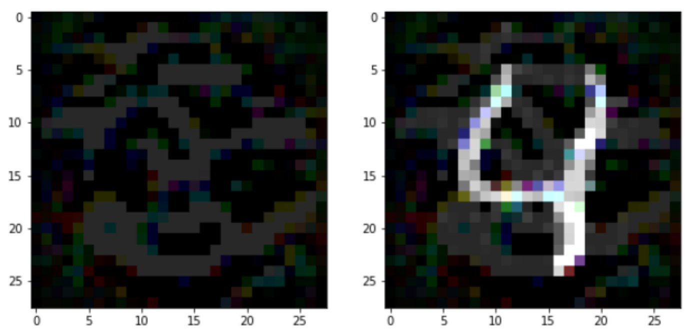
```
import torch.optim as optim

model.eval()
def l_infinity_pgd(model, tensor, gt,epsilon=40./255, target=None,iteration=500, show=True):#target是所定向的目标结果
    delta = torch.zeros_like(tensor, requires_grad=True)
    opt = optim.SGD([delta], lr=0.1)
    print(target)
    for t in range(iteration):
        pred = model(norm(tensor + delta))
        if target is None:
            loss = -nn.CrossEntropyLoss()(pred, torch.LongTensor([gt]))#没有目标就是普通的对抗样本
        else:
```

```
            loss = - 0.5 * nn.CrossEntropyLoss()(pred, torch.LongTensor([4])) + nn.CrossEntropyLoss()(pred, torch.LongTensor([target]))#有目标则会向目标的损失函数变小的方向优化。
        if t % 50 == 0:
            print(t, loss.item())

        opt.zero_grad()
        loss.backward()
        opt.step()
        delta.data.clamp_(-epsilon, epsilon)

    print("True class probability:", nn.Softmax(dim=1)(pred))
    cnn_eval(norm(tensor+delta))

    if show:
        f,ax = plt.subplots(1,2, figsize=(10,5))
        ax[0].imshow((delta)[0].detach().numpy().transpose(1,2,0))
        ax[1].imshow((tensor + delta)[0].detach().numpy().transpose(1,2,0))

    return tensor + delta

x= l_infinity_pgd(model,four_tensor,4,target=8)#这里运用的是pgd方法， 定向攻击， 我们可以将一个数字4辨认成数字8
```

```
8
0 14.67684555053711
50 0.37476295232772827
100 -0.4254000782966614
150 -0.7032363414764404
200 -0.937480092048645
250 -1.0612775087356567
300 -1.1266465187072754
350 -1.1796194314956665
400 -1.2356950044631958
450 -1.301567792892456
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
True class probability: tensor([[0.0037, 0.0102, 0.1287, 0.0081, 0.0376, 0.0009, 0.0021, 0.0019, 0.7480,
         0.0587]], grad_fn=<SoftmaxBackward0>)
tensor(8)
```



可以看到这里图像被辨认成我们所设定的数字8了，但明显对肉眼而言它是4， 攻击效果还是非常显著的。实际上这种攻击方式可以不针对数字， 我们随机选择一个图片都可以定向攻击：

```
import torch.optim as optim

rem_img = Image.open("mnist/pic/tienan.jpeg")
rem_img = rem_img.convert('RGB')
transform=transforms.Compose([
    transforms.Resize((28,28))   ,
        transforms.ToTensor()
        ])

norm=transforms.Compose([
```

```
        transforms.Normalize((0.1307,), (0.3081,))
        ])


rem_tensor = transform(rem_img)[None,:,:,:]
cnn_eval(norm(rem_tensor))
plt.imshow(rem_tensor[0].numpy().transpose(1,2,0))
import torch.optim as optim
# 注意修改gt为输出值
pred = 7
l_infinity_pgd(model,rem_tensor,pred,20./255,6,150)
```

这里钢铁侠可以被辨认成数字6，（取决于我们的目标设置的多少）

```
6
0 2.306993007659912
50 -1.2600481510162354
100 -1.4680516719818115
True class probability:
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
tensor([[0.0748, 0.0358, 0.0666, 0.0181, 0.0125, 0.0825, 0.5136, 0.0114, 0.1816,
         0.0031]], grad_fn=<SoftmaxBackward0>)
tensor(6)
```

小小的修改一下测试结果

```
import torch.optim as optim
# 注意修改gt为输出值
pred = 7
l_infinity_pgd(model,rem_tensor,pred,20./255,8,500)
```

```
8
0 2.8578264713287354
50 -7.608111381530762
100 -10.510968208312988
150 -11.766988754272461
200 -12.201788902282715
250 -12.477649688720703
300 -12.529745101928711
350 -12.286069869995117
400 -12.703282356262207
450 -12.747361183166504
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
True class probability: tensor([[4.4705e-09, 1.9020e-07, 3.0866e-09, 2.7272e-06, 1.3342e-11, 2.5854e-05,
         2.4373e-06, 3.7964e-11, 9.9997e-01, 5.8779e-09]],
       grad_fn=<SoftmaxBackward0>)
tensor(8)
```

```
8]: tensor([[[[ 0.8556,  1.1379,  0.9282,  ...,  1.0948,  0.8431,  0.8392],
          [ 1.1569,  0.9005,  0.8453,  ...,  1.1569,  1.1569,  1.1569],
          [ 0.8468,  0.8431,  0.8431,  ...,  0.8431,  1.1064,  1.0185],
          ...,
          [ 1.0629,  1.1230,  0.9286,  ...,  0.6551,  1.1437,  0.9006],
          [ 0.8461,  0.8242,  0.1407,  ...,  0.3004,  0.8550,  1.0161],
          [ 1.1358,  0.8180,  0.2069,  ...,  0.4527,  1.0863,  1.0273]],

         [[ 0.5697,  0.5865,  0.6235,  ...,  0.3844,  0.4528,  0.3176],
          [ 0.6185,  0.3098,  0.6195,  ...,  0.3146,  0.6235,  0.4031],
          [ 0.3697,  0.3257,  0.3098,  ...,  0.3134,  0.4314,  0.4285],
          ...,
          [ 0.6227,  0.6196,  0.5294,  ...,  0.3913,  0.5525,  0.4433],
          [ 0.3149,  0.2991,  0.0890,  ...,  0.3973,  0.4792,  0.5687],
```

个人感觉其实命中target的概率不算特别大，我自己尝试了很多次，最后增大了训练的轮数才达到了这个结果。

这种针对cv的攻击有一个更严重的地方，在于他的对抗样本可以迁移，

```
# create dataset
import os
from torchvision.utils import save_image
def create_adv_dataset():
    transform=transforms.Compose([
```

```
        transforms.ToTensor()
        ])
    dataset1 = datasets.ImageFolder('mnist/training',transform=transform)
    train_loader = torch.utils.data.DataLoader(dataset1,batch_size=1,
shuffle=True, num_workers=8)

    attack_target = 0
    # 每个数字生成100个对抗样本
    for batch_idx, (data, target) in enumerate(train_loader):
        attack_target = batch_idx//100
        if target== attack_target:
            continue
        if attack_target>9:
            break
        model=Net()
        model.load_state_dict(torch.load("mnist_cnn.pt"))
        model.eval()
        adv_img =
l_infinity_pgd(model,data,target,35./255,attack_target,50,False)
        image_dir_1 = os.path.join('mnist/adv_ori_label',str(target.item()))
        image_dir_2 = os.path.join('mnist/adv_adv_label',str(attack_target))
        if not os.path.exists(image_dir_1):
            os.makedirs(image_dir_1)
        if not os.path.exists(image_dir_2):
            os.makedirs(image_dir_2)

        save_image(adv_img, os.path.join(image_dir_1,str(batch_idx)+'.jpg'))
        save_image(adv_img, os.path.join(image_dir_2,str(batch_idx)+'.jpg'))

create_adv_dataset()
```

这里的代码是我们针对神经网络训练生成了一些对抗样本， 下面我们进行测试

```

test_transform=transforms.Compose([
        #transforms.GaussianBlur(3, sigma=(0.1, 1.0)),
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
        ])

dataset1 = datasets.ImageFolder('mnist/testing',transform=test_transform)#正常数据
集
dataset2 = datasets.ImageFolder('mnist/adv_ori_label',transform=test_transform)#
对抗样本
test_loader1 = torch.utils.data.DataLoader(dataset1,
shuffle=False,batch_size=100)
test_loader2 = torch.utils.data.DataLoader(dataset2,
shuffle=False,batch_size=100)

model=Net()#首先对它相对应的神经网络进行攻击
model.load_state_dict(torch.load("mnist_cnn.pt"))
model.eval()

test(model, 'cpu', test_loader1)
test(model, 'cpu', test_loader2)

model=MobileNet()#其次对不太相干的MobileNet进行攻击
```

```
model.load_state_dict(torch.load("mnist_mobile.pt"))
model.eval()

test(model, 'cpu', test_loader1)
test(model, 'cpu', test_loader2)
```

```
Test set: Average loss: 0.0450, Accuracy: 9847/10000 (98%)

Test set: Average loss: 0.2001, Accuracy: 2849/3069 (93%)

Test set: Average loss: 0.0818, Accuracy: 9733/10000 (97%)

Test set: Average loss: 0.2960, Accuracy: 2745/3069 (89%)
```

可以看到这类数据集下， 两种训练方法的正确率都有所下降， 即使这是我们针对某一种方法训练的对抗样本。 这足以说明对抗样本攻击的危害。