ECE 661: Homework #5

Adversarial Attacks and Defenses

NetID: wh162

1. True/False Question
   a. Problem 1.1:

      false, an evasion attack is to manipulate the input to get a wrong output.

   b. Problem 1.2:

      false, modern defenses aren't able to realize robustness and accuracy in the same time.

   c. Problem 1.3:

      True, refer lect16

   d. Problem 1.4:

      true, refer to lec16

   e. Problem 1.5:

      true.

   f. Problem 1.6:

      false, there may occur gradient masking

   g. Problem 1.7:

      false, should be gradient-based approaches.

   h. Problem 1.8:

      false, should be input

   i. Problem 1.9:

      true, refer lec18

j. Problem 1.10:

   true, refer lec 16

2. Lab1: Environment setup and attack implementation
   a. Problem 2.1:

   For netA_standard: the accuracy is 92.48%

```
Epoch: [ 0 / 20 ]; TrainAcc: 0.84570; TrainLoss: 0.42193; TestAcc: 0.88790; TestLoss: 0.30916
Epoch: [ 1 / 20 ]; TrainAcc: 0.90178; TrainLoss: 0.26917; TestAcc: 0.90030; TestLoss: 0.27774
Epoch: [ 2 / 20 ]; TrainAcc: 0.91660; TrainLoss: 0.22825; TestAcc: 0.90910; TestLoss: 0.25362
Epoch: [ 3 / 20 ]; TrainAcc: 0.92720; TrainLoss: 0.19801; TestAcc: 0.90410; TestLoss: 0.26123
Epoch: [ 4 / 20 ]; TrainAcc: 0.93608; TrainLoss: 0.17555; TestAcc: 0.91090; TestLoss: 0.24719
Epoch: [ 5 / 20 ]; TrainAcc: 0.94155; TrainLoss: 0.15628; TestAcc: 0.91260; TestLoss: 0.26112
Epoch: [ 6 / 20 ]; TrainAcc: 0.95092; TrainLoss: 0.13421; TestAcc: 0.91330; TestLoss: 0.28218
Epoch: [ 7 / 20 ]; TrainAcc: 0.95643; TrainLoss: 0.11860; TestAcc: 0.91780; TestLoss: 0.26494
Epoch: [ 8 / 20 ]; TrainAcc: 0.96315; TrainLoss: 0.10230; TestAcc: 0.91590; TestLoss: 0.27438
Epoch: [ 9 / 20 ]; TrainAcc: 0.96700; TrainLoss: 0.09025; TestAcc: 0.91460; TestLoss: 0.30235
Epoch: [ 10 / 20 ]; TrainAcc: 0.97202; TrainLoss: 0.07704; TestAcc: 0.91350; TestLoss: 0.31368
Epoch: [ 11 / 20 ]; TrainAcc: 0.97487; TrainLoss: 0.06749; TestAcc: 0.91410; TestLoss: 0.32474
Epoch: [ 12 / 20 ]; TrainAcc: 0.97683; TrainLoss: 0.06212; TestAcc: 0.91780; TestLoss: 0.35907
Epoch: [ 13 / 20 ]; TrainAcc: 0.98073; TrainLoss: 0.05387; TestAcc: 0.91670; TestLoss: 0.34196
Epoch: [ 14 / 20 ]; TrainAcc: 0.98185; TrainLoss: 0.04925; TestAcc: 0.91000; TestLoss: 0.40046
Epoch: [ 15 / 20 ]; TrainAcc: 0.98153; TrainLoss: 0.04885; TestAcc: 0.91300; TestLoss: 0.42474
Epoch: [ 16 / 20 ]; TrainAcc: 0.99505; TrainLoss: 0.01549; TestAcc: 0.92330; TestLoss: 0.40694
Epoch: [ 17 / 20 ]; TrainAcc: 0.99877; TrainLoss: 0.00629; TestAcc: 0.92370; TestLoss: 0.43649
Epoch: [ 18 / 20 ]; TrainAcc: 0.99957; TrainLoss: 0.00358; TestAcc: 0.92380; TestLoss: 0.46828
Epoch: [ 19 / 20 ]; TrainAcc: 0.99987; TrainLoss: 0.00221; TestAcc: 0.92480; TestLoss: 0.49687
Done!
```

   For netB_standard: the accuracy is 92.37%

```
Epoch: [ 0 / 20 ]; TrainAcc: 0.84213; TrainLoss: 0.43457; TestAcc: 0.87960; TestLoss: 0.32804
Epoch: [ 1 / 20 ]; TrainAcc: 0.90115; TrainLoss: 0.27087; TestAcc: 0.89550; TestLoss: 0.27884
Epoch: [ 2 / 20 ]; TrainAcc: 0.91627; TrainLoss: 0.23079; TestAcc: 0.90740; TestLoss: 0.26146
Epoch: [ 3 / 20 ]; TrainAcc: 0.92722; TrainLoss: 0.20109; TestAcc: 0.90890; TestLoss: 0.25029
Epoch: [ 4 / 20 ]; TrainAcc: 0.93363; TrainLoss: 0.18255; TestAcc: 0.91650; TestLoss: 0.23956
Epoch: [ 5 / 20 ]; TrainAcc: 0.94180; TrainLoss: 0.16140; TestAcc: 0.91940; TestLoss: 0.24100
Epoch: [ 6 / 20 ]; TrainAcc: 0.94707; TrainLoss: 0.14480; TestAcc: 0.91320; TestLoss: 0.25672
Epoch: [ 7 / 20 ]; TrainAcc: 0.95168; TrainLoss: 0.13078; TestAcc: 0.90910; TestLoss: 0.28265
Epoch: [ 8 / 20 ]; TrainAcc: 0.95807; TrainLoss: 0.11186; TestAcc: 0.91890; TestLoss: 0.25888
Epoch: [ 9 / 20 ]; TrainAcc: 0.96290; TrainLoss: 0.10009; TestAcc: 0.91630; TestLoss: 0.27774
Epoch: [ 10 / 20 ]; TrainAcc: 0.96765; TrainLoss: 0.08677; TestAcc: 0.91940; TestLoss: 0.31172
Epoch: [ 11 / 20 ]; TrainAcc: 0.97100; TrainLoss: 0.07859; TestAcc: 0.91540; TestLoss: 0.31969
Epoch: [ 12 / 20 ]; TrainAcc: 0.97455; TrainLoss: 0.06845; TestAcc: 0.91620; TestLoss: 0.34560
Epoch: [ 13 / 20 ]; TrainAcc: 0.97782; TrainLoss: 0.06080; TestAcc: 0.91390; TestLoss: 0.36866
Epoch: [ 14 / 20 ]; TrainAcc: 0.97850; TrainLoss: 0.05718; TestAcc: 0.91470; TestLoss: 0.37720
Epoch: [ 15 / 20 ]; TrainAcc: 0.98130; TrainLoss: 0.05135; TestAcc: 0.91380; TestLoss: 0.38506
Epoch: [ 16 / 20 ]; TrainAcc: 0.99410; TrainLoss: 0.01829; TestAcc: 0.92290; TestLoss: 0.40765
Epoch: [ 17 / 20 ]; TrainAcc: 0.99810; TrainLoss: 0.00807; TestAcc: 0.92330; TestLoss: 0.44487
Epoch: [ 18 / 20 ]; TrainAcc: 0.99930; TrainLoss: 0.00456; TestAcc: 0.92270; TestLoss: 0.48032
Epoch: [ 19 / 20 ]; TrainAcc: 0.99982; TrainLoss: 0.00260; TestAcc: 0.92370; TestLoss: 0.52252
Done!
```

   No, the two models doesn't have the same architecture, model A has one 28x28 and one 14x14 conv2d layer, whereas model B has two 28x28 and two 14x14 conv2d layer.

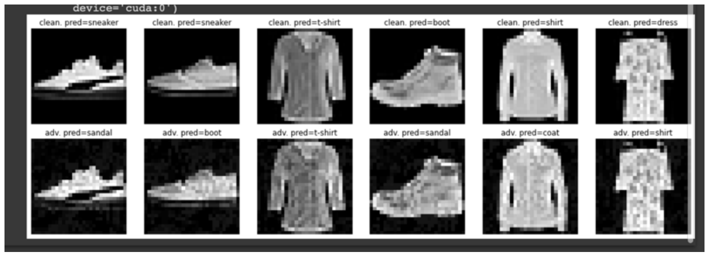
   b. Problem 2.2:

```
x_nat = dat.clone().detach()

# If rand_start is True, add uniform noise to the sample within [-eps,+eps],
# else just copy x_nat
if rand_start:
  x_adv = x_nat.clone().detach() + torch.FloatTensor(x_nat.shape).uniform_(-eps, eps).to(device)
else:
  x_adv = x_nat.clone().detach()
# Make sure the sample is projected into original distribution bounds [0,1]
x_adv = torch.clamp(x_adv, min=0., max=1.)

# Iterate over iters
for i in range(iters):
    # Compute gradient w.r.t  . data (we give you this function, but understand it)
  gradient = gradient_wrt_data(model, device, x_adv , lbl)
    # Perturb the image using the gradient
  x_adv += alpha * torch.sign(gradient)
    # Clip the perturbed datapoints to ensure we still satisfy L_infinity constraint
  x_adv = torch.clamp(x_adv, min=x_nat - eps, max=x_nat + eps)
    # Clip the perturbed datapoints to ensure we are in bounds [0,1]
  x_adv = torch.clamp(x_adv, min=0., max=1.)
# Return the final perturbed samples
return x_adv
```

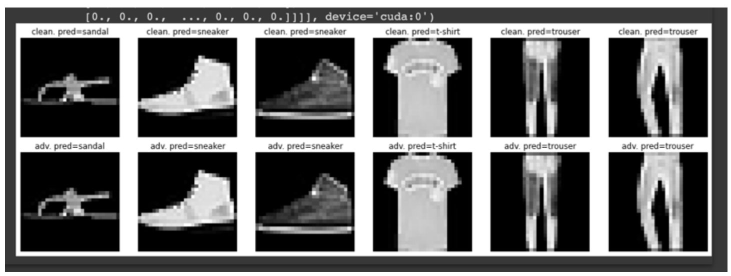
✓ 3m 12s   completed at 2:23 PM

When the value of EPS reaches 0.08, I can start to notice that the difference between the two images. For me, it would be hard to predict or to tell the correctness of the image.

EPS = 0.08



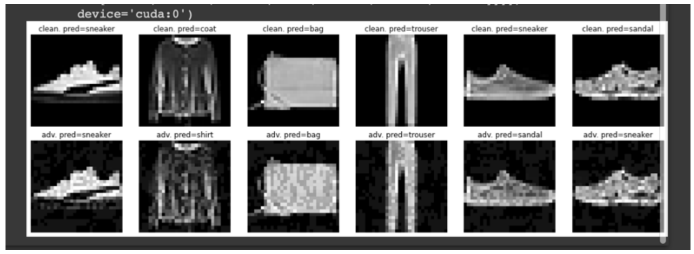
EPS = 0.0



c. Problem 2.3:

```python
def FGSM_attack(model, device, dat, lbl, eps):
    # TODO: Implement the FGSM attack
    # - Dat and lbl are tensors
    # - eps is a float

    # HINT: FGSM is a special case of PGD

    return PGD_attack(model, device, dat, lbl, eps, eps, 1, False)


def rFGSM_attack(model, device, dat, lbl, eps):
    # TODO: Implement the FGSM attack
    # - Dat and lbl are tensors
    # - eps is a float

    # HINT: rFGSM is a special case of PGD

    return PGD_attack(model, device, dat, lbl, eps, eps, 1, True)
```

We can see that under the same EPS = 0.08, the noise was much noticeable, we can easily spot the difference between the images. As a result, under the same EPS, the noise of PGD and FGSM are visually different.



d. Problem 2.4:

```
def FGM_L2_attack(model, device, dat, lbl, eps):
    # x_nat is the natural (clean) data batch, we .clone().detach()
    # to copy it and detach it from our computational graph
    x_nat = dat.clone().detach()

    # Compute gradient w.r.t. data
    gradient = gradient_wrt_data(model, device, x_nat, lbl)
    # Compute sample-wise L2 norm of gradient (L2 norm for each batch element)
    # HINT: Flatten gradient tensor first, then compute L2 norm
    gradient_l2 = gradient.flatten()

    # Perturb the data using the gradient
    # HINT: Before normalizing the gradient by its L2 norm, use
    # torch.clamp(l2_of_grad, min=1e-12) to prevent division by 0
    gradient_l2 = torch.clamp(gradient_l2, min=1e-12)
    gradient_l2 = gradient / torch.norm(gradient_l2, keepdim = True)
    # Add perturbation the data
    x_adv = x_nat + eps * torch.sign(gradient_l2)
    # Clip the perturbed datapoints to ensure we are in bounds [0,1]
    x_adv = torch.clamp(x_adv, min=0., max=1.)
    # Return the perturbed samples
    return x_adv
```

After several experiment on the value of EPS, we can see that the noise of the image grew dramatically, no sooner after the value of 0.2 can't we see the origin image. Unlike the slowing increasing noise in FGSM and PGD, which we can still see the origin image even after larger value of EPS.

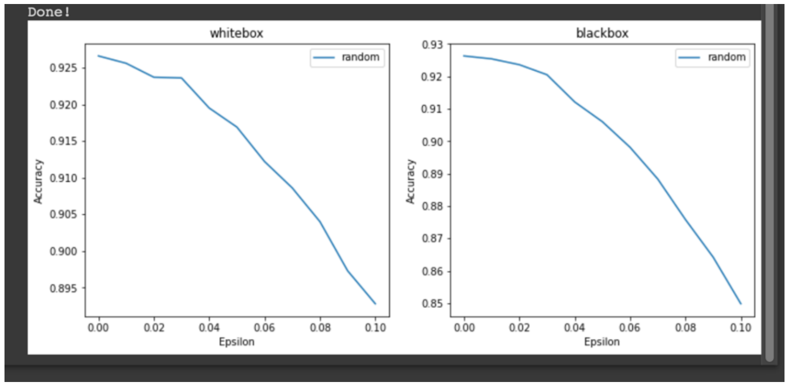
3. Lab2: Measuring attack success rate
    a. Problem 3.1:

    A whitebox attack is when the hacker has the access to the model's weight, whereas a blackbox attack is when the hacker only has access to the input and the output of the model.

    Vulnerability

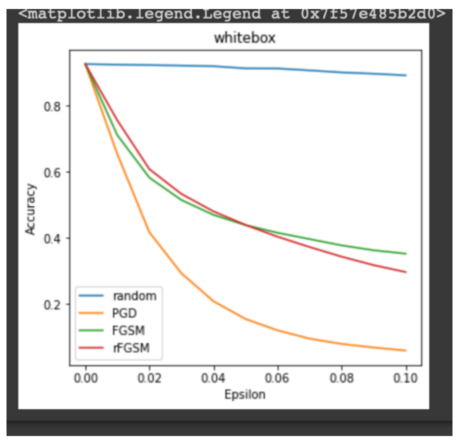
    b. Problem 3.2:

    We can see from the plot that as the eps grows, the accuracy decreases.



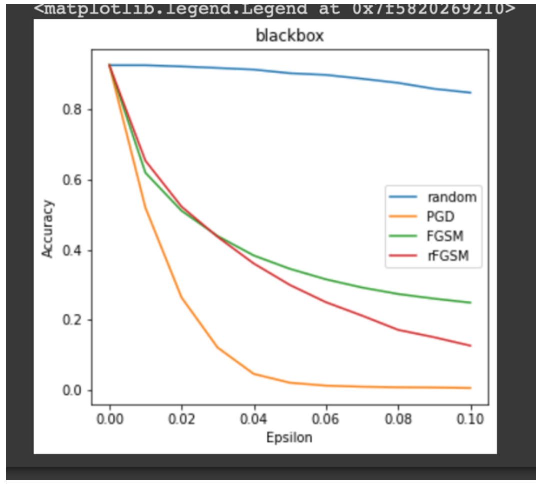
c. Problem 3.3:

From the plot, it is clearly that PGD has the deepest accuracy drop, whereas random attack maintained the quality of the accuracy. Also, from the plot, FGSM and rFGSM merely different from each other. Though the trigger of the drop of random attack is not obvious, I would guess that around eps=0.05, the decrease of the random attack accuracy starts to get noticeable.



d. Problem 3.4:

From the testing of blackbox, the accuracy starts dropping when eps=0.03, which is different than whitebox testing. The curve remain the same, where random attack has the highest accuracy and PDG drops the fastest.



e. Problem 3.5:

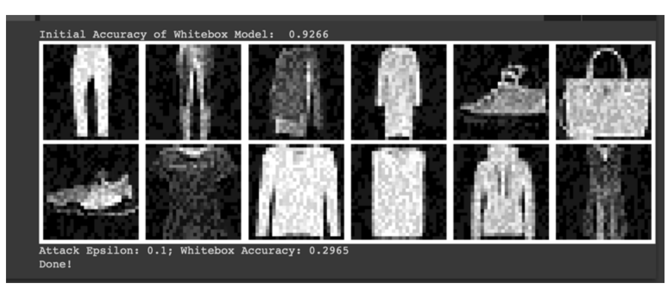
   The whitebox attack is more powerful. We can direct access the weight of the model and thus makes it easier and powerful to break down a model. It makes, sense because the main point of a model is the weight, once we can control the weights, we can control the model.

4. Lab3: Adversarial training
   a. Problem 4.1:

      The final accuracy of the clean test data is 35.3%. Comparing to the beginning, the accuracy is actually a bit higher than the origin model. We can see from the output below, the whitebox accuracy of FGSM is higher than rFGSM.

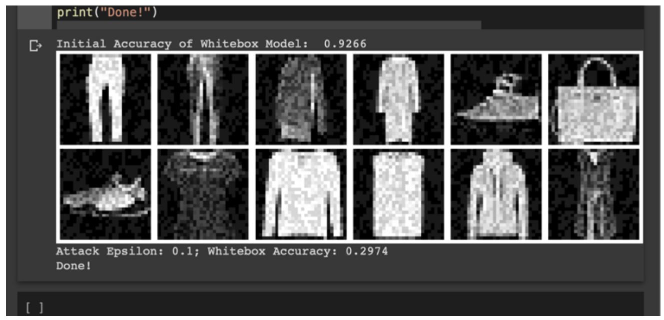
      FGSM:

      rFGSM:



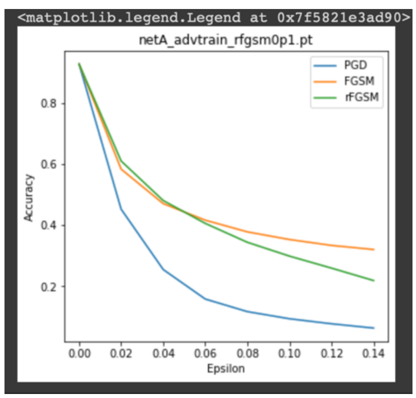
b. Problem 4.2:

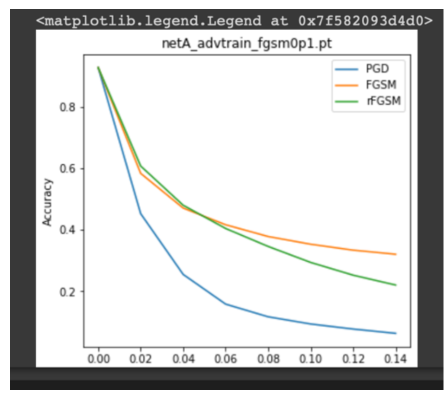
The final accuracy of the clean data is 29.74%, the accuracy is less than the origin model. Also from the results of FGSM and PGD, we can see that FGSM has a higher accuracy than PGD



c. Problem 4.3:

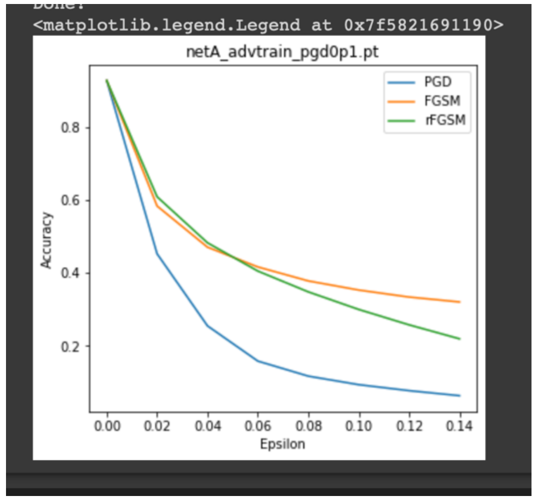
We can see that the model is not robust for all type of attacks.





d. Problem 4.4:

From the three plots, it is hard to tell which one is better



e. Problem 4.5:
f. Problem 4.6: