ECE 661: Homework 5 Adversarial Attacks and Defenses

Frankie Willard

November 2022

1 True/False Questions (10 pts)

Problem 1.1 (1 pt) In an evasion attack, the attacker perturbs a subset of training instances which prevents the DNN from learning an accurate model.

False. Evasion attacks are inference time, in which it manipulates/perturbs the user input to fool the DNN System (L16 Slide 10). Thus, it is not affecting the training instances, but rather the testing instances.

Problem 1.2 (3 pts) In general, modern defenses not only improve robustness to adversarial attack, but they also improve accuracy on clean data.

False. A tradeoff between robustness and accuracy always exists, and an accurate and robust model is still hard to achieve (L18 slide 18).

Problem 1.3 (3 pts) In a backdoor attack, the attacker first injects a specific noise trigger to a subset of data points and sets the corresponding labels to a target class. Then, during deployment, the attacker uses a gradient-based perturbation (e.g., Fast Gradient Sign Method) to fool the model into choosing the target class.

False. While the first part is true for step 1, the backdoor triggering during deployment simply places the trigger on the input image, rather than a gradient-based perturbation (L16 slide 31)

Problem 1.4 (3 pts) Outlier exposure is an Out-of-Distribution (OOD) detection technique that uses OOD data during training, unlike the ODIN detector.

True. Outlier exposure uses a variety of OOD data during training and minimizes a 2-part loss (L16 slide 39). ODIN uses temperature scaling in the softmax funcation and small perturbations in the inputs to enlarge the softmax score gap between ID and OOD examples (L16 slide 38).

Problem 1.5 (3 pts) It is likely that an adversarial examples generated on a ResNet-50 model will also fool a VGG-16 model.

True. This is the foundation of the idea of transfer attacks, which are even performed in this lab.

Problem 1.6 (3 pts) The perturbation direction used by the Fast Gradient Sign Method attack is the direction of steepest ascent on the local loss surface, which is the most efficient direction towards the decision boundary

False. While the FGSM does move in the direction of steepest ascent on the local loss surface, the diection of steeps ascent does not guarantee it is the most efficient direction towards the decision boundary (L17 slide 14)

Problem 1.7 (3 pts) The purpose of the projection step of the Projected Gradient Descent (PGD) attack is to prevent a misleading gradient due to gradient masking.

False. Starting from a random point near the data sample is likely to mitigate the gradient masking effect (L17 slide 17). Thus, it is the random point, rather than the projection step, that is preventing the misleading gradient due to gradient masking.

Problem 1.8 (3 pts) Analysis shows that the best layer for generating the most transferable feature space attacks is the final convolutional layer, as it is the convolutional layer that has the most effect on the prediction.

False. Attacks from intermediate layers transfer better than those generated at the output layer (e.g., FGSM) (L17 slide 28)

Problem 1.9 (3 pts) The DVERGE training algorithm promotes a more robust model ensemble, but the individual models within the ensemble still learn non-robust features.

True. Objective can be minimized if sub-model using different set of features, including non-robust features, leading to higher clean accuacy. Great ensemble robustness is achieved against vaious attacks (Lecture 18 slide 24)

Problem 1.10 (3 pts) On a backdoored model, the exact backdoor trigger must be used by the attacker during deployment to cause the proper targeted misclassification.

False. Backdoor triggering occurs during deployment and consists of one trigger to manipulate the model's outputs on all inputs. (L16 slide 31-32). However, the trigger does not have to be exact. According to Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks, ”when the model is trained to recognize the trigger, it may not learn the exact shape and color of the trigger. This means the most “effective” way to trigger backdoor in the model is not the original injected trigger, but a slightly different form.”

2 Lab 1: Environment Setup and Attack Implementation (20 pts)

(a) (4 pts) Train the given NetA and NetB models on the FashionMNIST dataset. Use the provided training parameters and save two checkpoints: "netA_standard.pt" and "netB_standard.pt". What is the final test accuracy of each model? Do both models have the same architecture? (Hint: accuracy should be around 92 percent for both models).

NetA Test Accuracy: 0.92680

NetB Test Accuracy: 0.92190

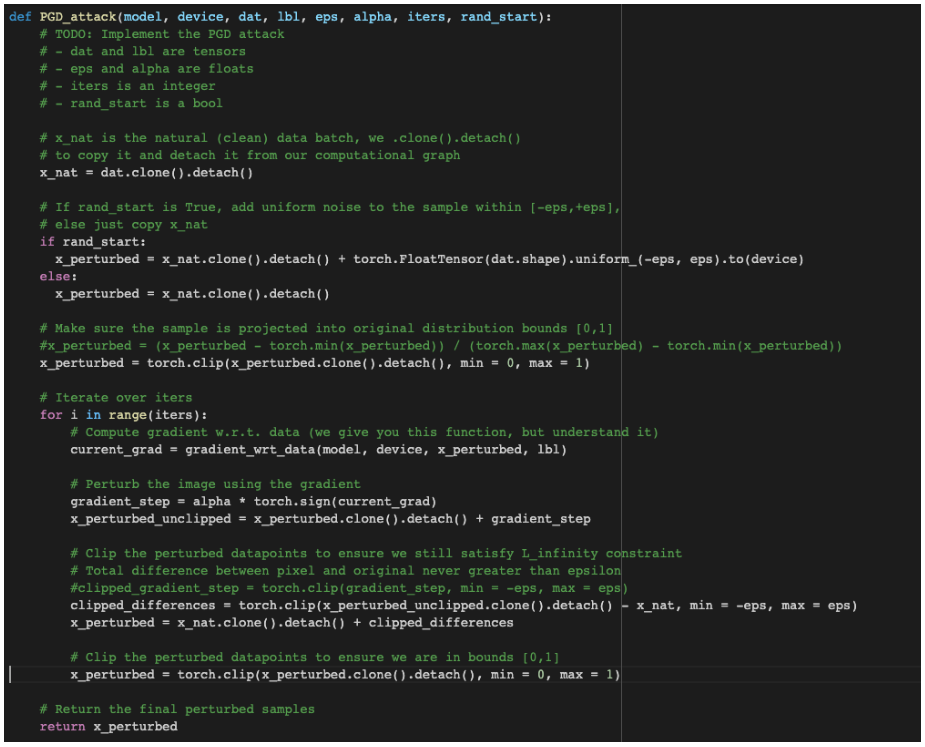
The models have different architectures. NetA consists of 3 Convolutions (followed by ReLU) with the first 2 having Max Pooling layers after, and then 2 linear layers after all the Conv/ReLU/Pooling. NetB consists of 4 Convolutions (followed by ReLU) with the second and fourth Conv layers having Max Pooling layers after (and then 2 linear layers after all the Conv/ReLU/Pooling). NetA has a higher number of feature maps output from each Convolutional layer.

(b) (8 pts) Implement the untargeted L∞-constrained Projected Gradient Descent (PGD) adversarial attack in the attacks.py file. In the report, paste a screenshot of your PGD_attack function and describe what each of the input arguments is controlling. Then, using the "Visualize some perturbed samples" cell in HWK5_main.ipynb, run your PGD attack using NetA as the base classifier and plot some perturbed samples using ϵ values in the range [0.0, 0.2]. At about what ϵ does the noise start to become perceptible/noticeable? Do you think that you (or any human) would still be able to correctly predict samples at this ϵ value? Finally, to test one important edge case, show that at $\epsilon = 0$ the computed adversarial example is identical to the original input image. (HINT: We give you a function to compute input gradient at the top of the attacks.py file)

Code:

Each input argument: model- the PyTorch NN that the attack is being performed on (important for gradient information) device- the CPU, GPU, or TPU that the network and its weights are being stored/run on dat- data to be perturbed

lbl- label of data (that we are trying to model to misclassify data to) eps- epsilon value representing the maximum absolute value pixel perturbation that is allowed to be undergone. also if rand_start true, $(-eps, eps)$ represents the range for which uniform noise is added to the data at the beginning (to prevent misleading gradient from gradient masking) alpha- scalar multiplied by the gradient to determine the pre-clipped gradient tsep iters- number of iterations to undergo the perturbation process rand_start- boolean variable determining whether or not uniform noise between -eps and eps should be added to the data at the beginning (to prevent misleading gradient from gradient masking)

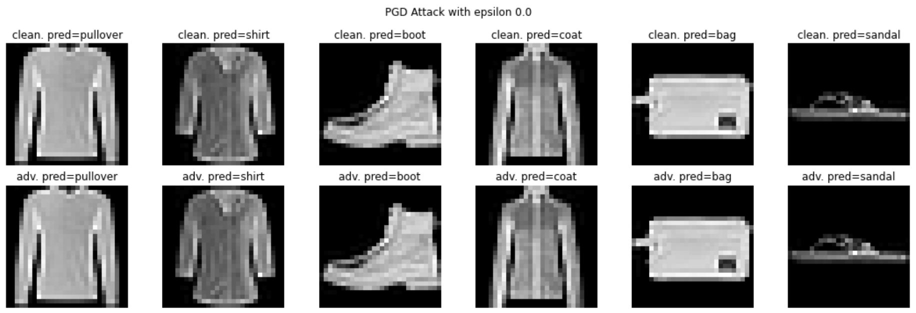


```python
def PGD_attack(model, device, dat, lbl, eps, alpha, iters, rand_start):
    # TODO: Implement the PGD attack
    # - dat and lbl are tensors
    # - eps and alpha are floats
    # - iters is an integer
    # - rand_start is a bool

    # 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()

    # If rand_start is True, add uniform noise to the sample within [-eps,+eps],
    # else just copy x_nat
    if rand_start:
      x_perturbed = x_nat.clone().detach() + torch.FloatTensor(dat.shape).uniform_(-eps, eps).to(device)
    else:
      x_perturbed = x_nat.clone().detach()

    # Make sure the sample is projected into original distribution bounds [0,1]
    #x_perturbed = (x_perturbed - torch.min(x_perturbed)) / (torch.max(x_perturbed) - torch.min(x_perturbed))
    x_perturbed = torch.clip(x_perturbed.clone().detach(), 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)
        current_grad = gradient_wrt_data(model, device, x_perturbed, lbl)

        # Perturb the image using the gradient
        gradient_step = alpha * torch.sign(current_grad)
        x_perturbed_unclipped = x_perturbed.clone().detach() + gradient_step

        # Clip the perturbed datapoints to ensure we still satisfy L_infinity constraint
        # Total difference between pixel and original never greater than epsilon
        #clipped_gradient_step = torch.clip(gradient_step, min = -eps, max = eps)
        clipped_differences = torch.clip(x_perturbed_unclipped.clone().detach() - x_nat, min = -eps, max = eps)
        x_perturbed = x_nat.clone().detach() + clipped_differences

        # Clip the perturbed datapoints to ensure we are in bounds [0,1]
        x_perturbed = torch.clip(x_perturbed.clone().detach(), min = 0, max = 1)

    # Return the final perturbed samples
    return x_perturbed
```

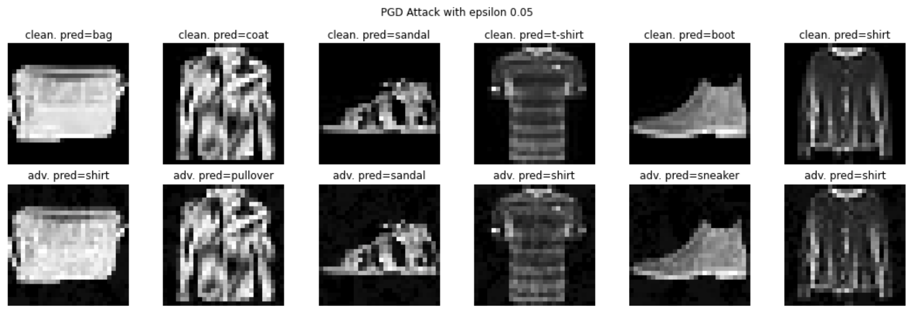
Analysis:

The noise starts to become somewhat perceptible for me around eps = 0.05, and at that point it begins misclassifying about half of the samples. I think humans would still be able to predict samples at this epsilon value. By eps = 0.1, the noise is fairly significant, but would still likely lead to correct classification by a human (although the model is only able to correctly classify one of them).

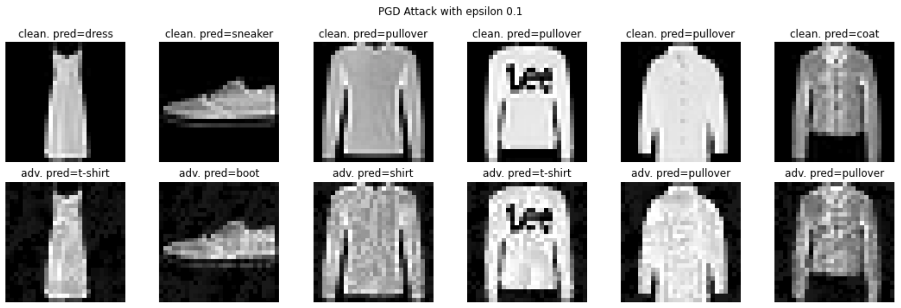
PGD Epsilon 0:



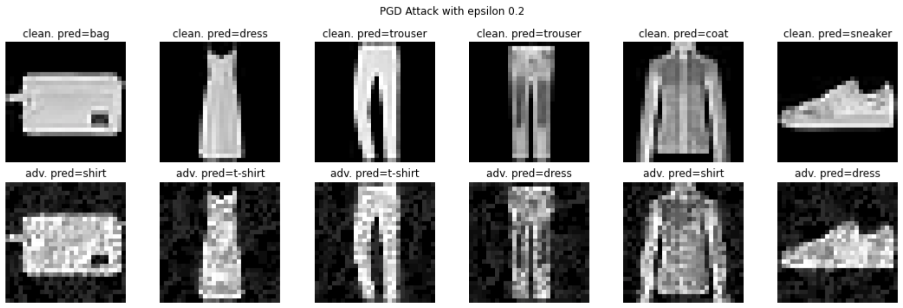
PGD Epsilon 0.05:



PGD Epsilon 0.1:

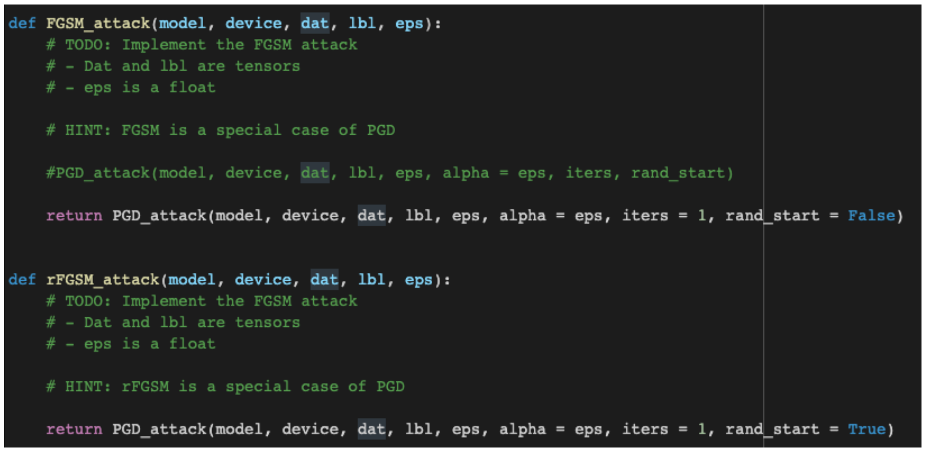


PGD Epsilon 0.2:



(c) (4 pts) Implement the untargeted L∞-constrained Fast Gradient Sign Method (FGSM) attack and random start FGSM (rFGSM) in the attacks.py file. (Hint: you can treat the FGSM and rFGSM functions as wrappers of the PGD function). Please include a screenshot of your FGSM_attack and rFGSM_attack function in the report. Then, plot some perturbed samples using the same ϵ levels from the previous question and comment on the perceptibility of the FGSM noise. Does the FGSM and PGD noise appear visually similar?

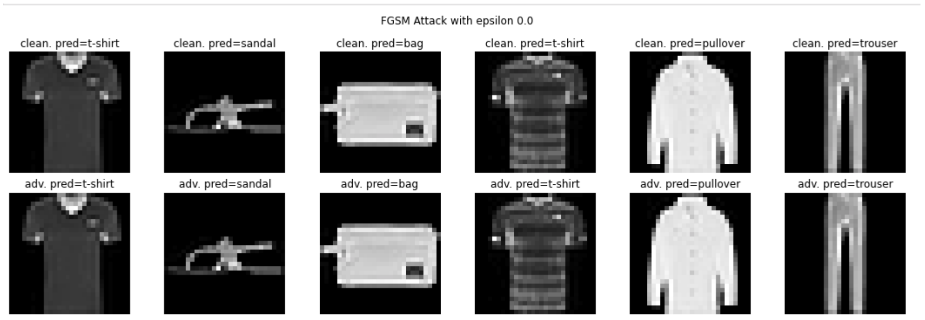
Code:


```
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

    #PGD_attack(model, device, dat, lbl, eps, alpha = eps, iters, rand_start)

    return PGD_attack(model, device, dat, lbl, eps, alpha = eps, iters = 1, rand_start = 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, alpha = eps, iters = 1, rand_start = True)
```

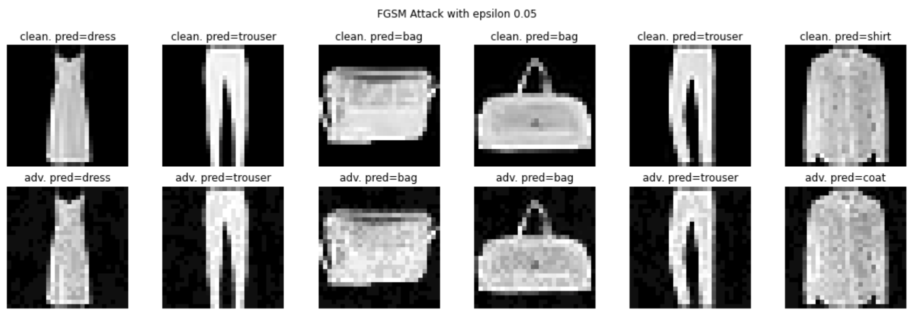
Analysis:

The noise starts to become somewhat perceptible for me earlier for the FGSM models. I can more clearly notice it in the FGSM model at eps= 0.05 in both the FGSM and rFGSM models, with the FGSM appearing marginally noisier to my naked eye. The PGD is much smoother/harder to notice at this epsilon value (as well as greater epsilon values). While the level of noise may appear different at these epsilon values, I can not tell any differences between the structure of the noise itself.

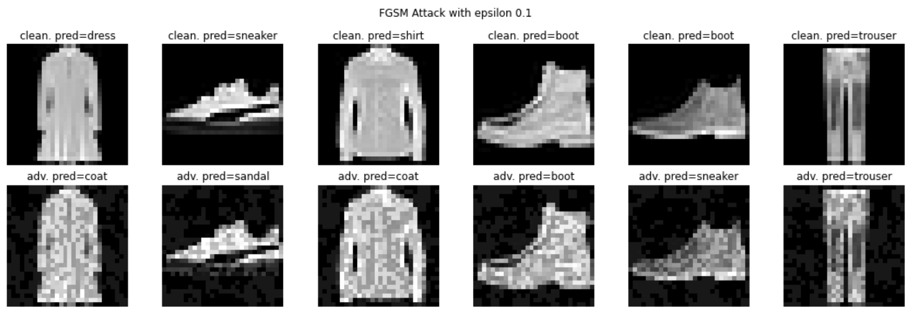
FGSM Epsilon 0:



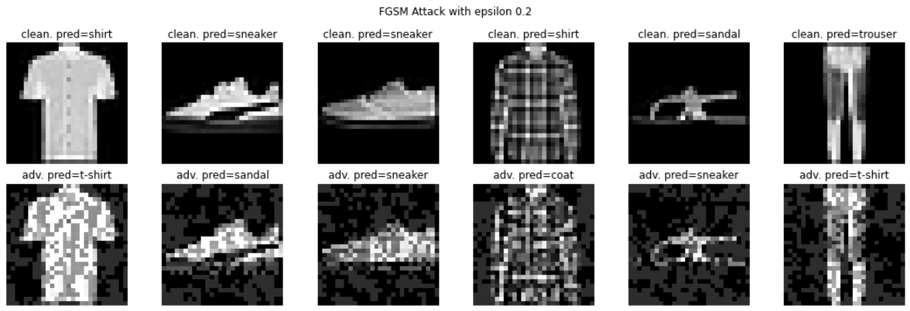
FGSM Epsilon 0.05:



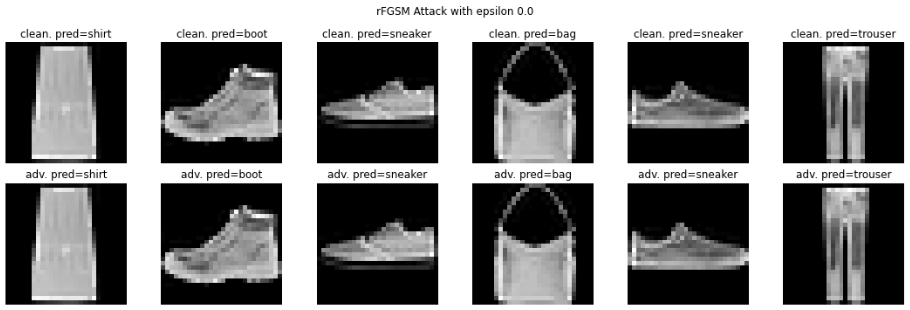
FGSM Epsilon 0.1:



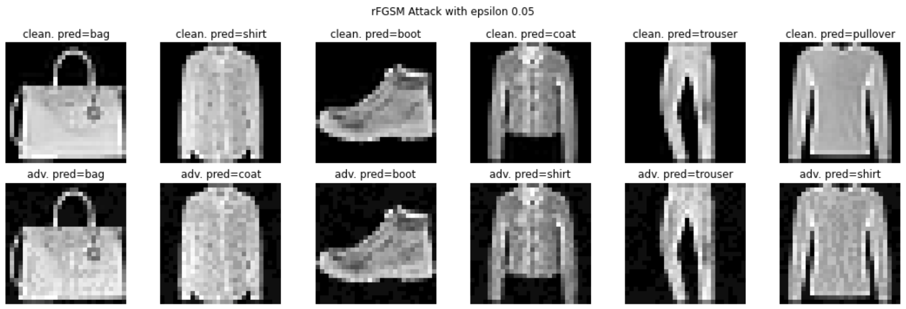
FGSM Epsilon 0.2:



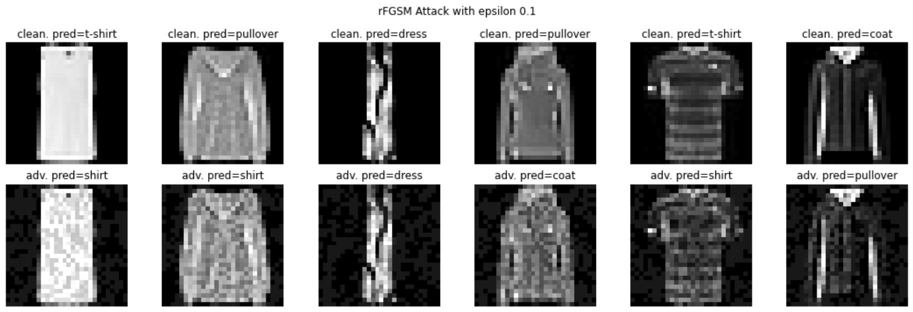
rFGSM Epsilon 0:



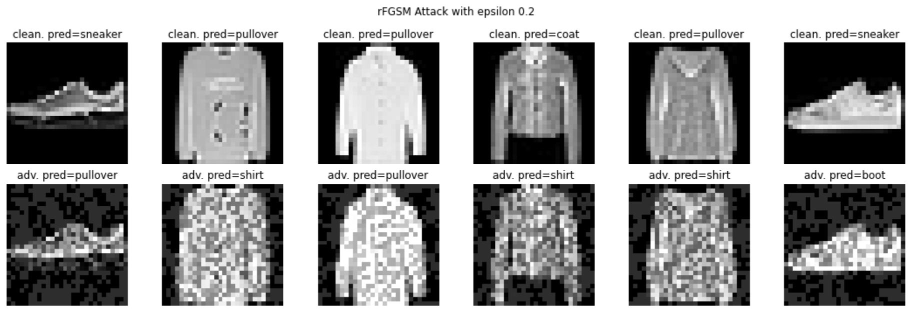
rFGSM Epsilon 0.05:



rFGSM Epsilon 0.1:

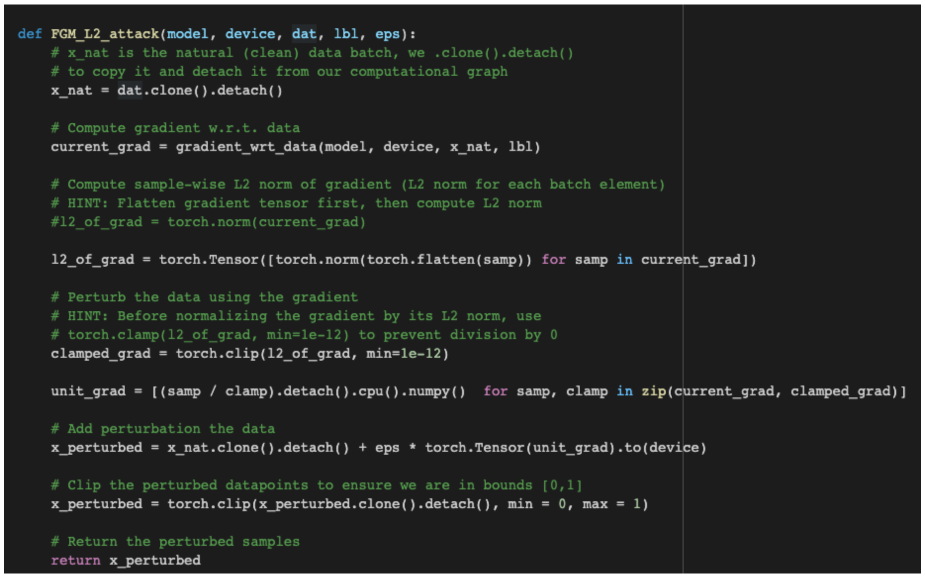


rFGSM Epsilon 0.2:



(d) (4 pts) Implement the untargeted L2-constrained Fast Gradient Method attack in the attacks.py file. Please include a screenshot of your FGM_L2_attack function in the report. Then, plot some perturbed samples using ϵ values in the range of [0.0, 4.0] and comment on the perceptibility of the L2 constrained noise. How does this noise compare to the L∞ constrained FGSM and PGD noise visually? (Note: This attack involves a normalization of the gradient, but since these attack functions take a batch of inputs, the norm must be computed separately for each element of the batch).

Code:

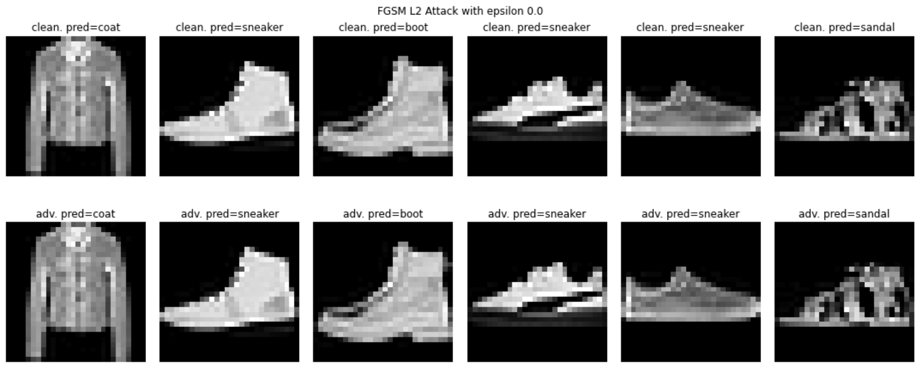


```
def FGM_L2_attack(model, device, dat, lbl, eps):
    # x_nat is the natural (clean) data batch, we .clone().detach()
    # to copy it and detach it from our computational graph
    x_nat = dat.clone().detach()

    # Compute gradient w.r.t. data
    current_grad = 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
    #l2_of_grad = torch.norm(current_grad)

    l2_of_grad = torch.Tensor([torch.norm(torch.flatten(samp)) for samp in current_grad])

    # 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
    clamped_grad = torch.clip(l2_of_grad, min=1e-12)

    unit_grad = [(samp / clamp).detach().cpu().numpy()  for samp, clamp in zip(current_grad, clamped_grad)]

    # Add perturbation the data
    x_perturbed = x_nat.clone().detach() + eps * torch.Tensor(unit_grad).to(device)

    # Clip the perturbed datapoints to ensure we are in bounds [0,1]
    x_perturbed = torch.clip(x_perturbed.clone().detach(), min = 0, max = 1)

    # Return the perturbed samples
    return x_perturbed
```

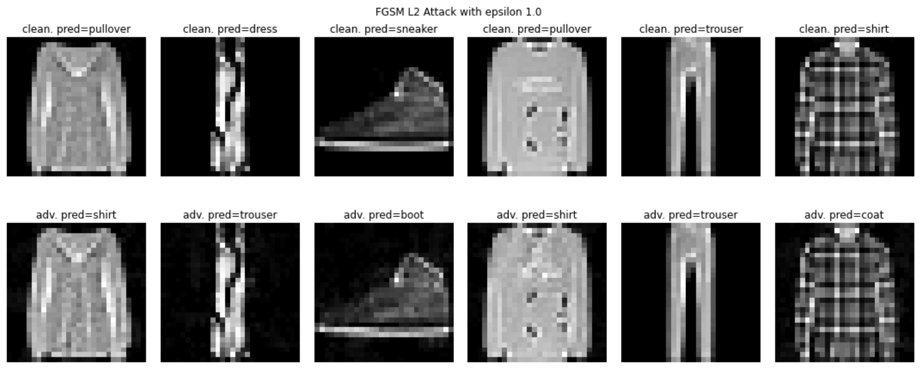
Analysis:

Visually, the noise in the FGM L2 model appears to be the most subtle. It is generally harder to find even for higher epsilon values, and even at high epsilon values of 3 and 4, it does not alter the neighboring pixels of the objects as much as high epsilon values for PGD or FGSM techniques.

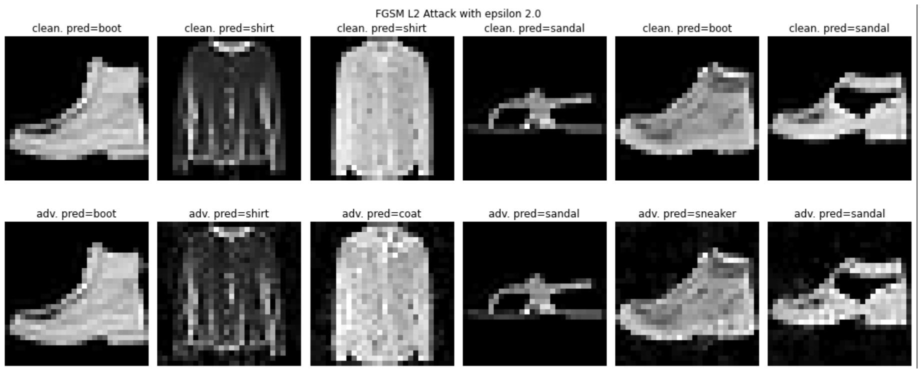
FGSM L2 Epsilon 0:



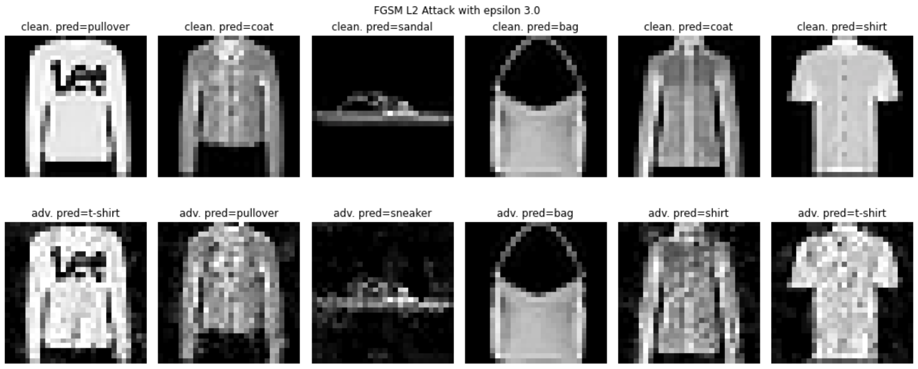
FGSM L2 Epsilon 1.0:



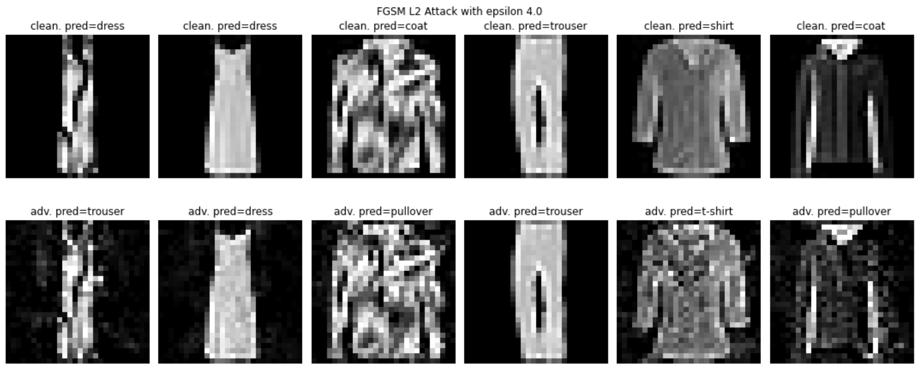
FGSM L2 Epsilon 2.0:



FGSM L2 Epsilon 3.0:



FGSM L2 Epsilon 4.0:



3 Lab 2: Measuring Attack Success Rate (30 pts)

(a) (2 pts) Briefly describe the difference between a whitebox and blackbox adversarial attacks. Also, what is it called when we generate attacks on one model and input them into another model that has been trained on the same dataset?

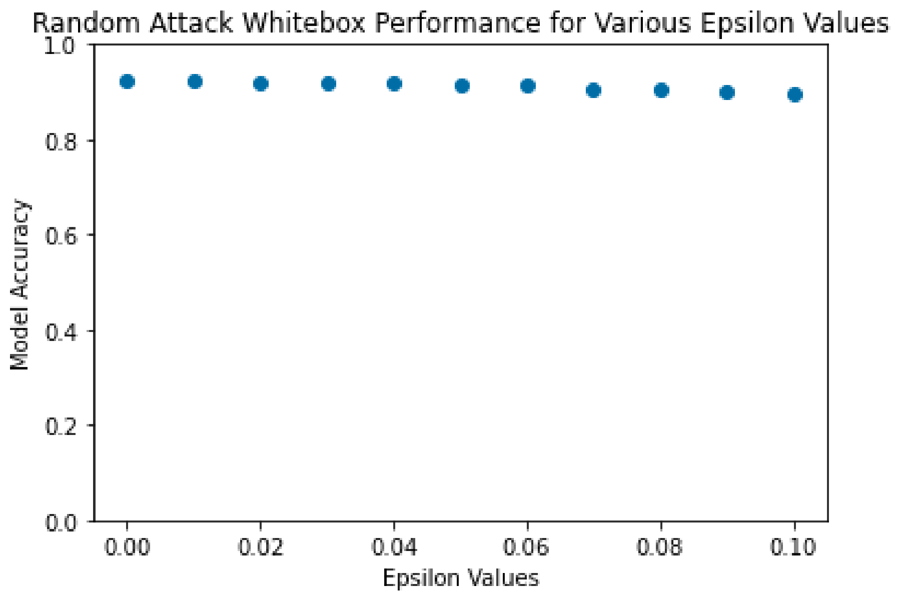
In a white-box attack, the attacker knows the specifications of the model being used including the architecture and weights. The attacker can perform complex optimizations to perform a streamlined attack that is much less limited by computation.

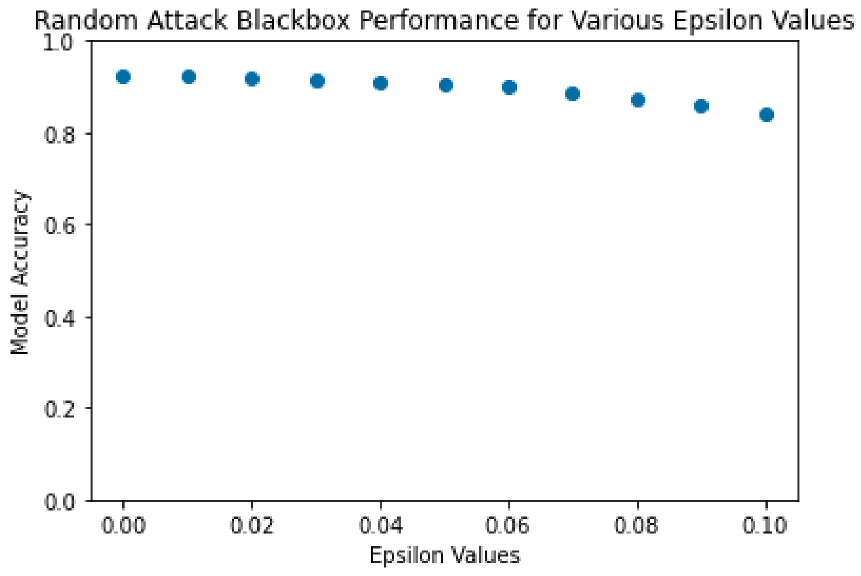
In a black-box attack, the attacker does not have this information relevant to the model being used and its architecture/weights/training. It merely access the target model's output, and it is bottlenecked by the number of successive queries it can make.

Transfer Attacks consists of training a proxy DNN using the target model's public dataset, then generate whitebox-style attacks on the proxy DNN and transfering them to the target model. Thus, this is the desired definition for the latter question.

(b) (3 pts) Random Attack - To get an attack baseline, we use random uniform perturbations in range [-ϵ, ϵ]. We have implemented this for you in the attacks.py file. Test at least eleven ϵ values across the range [0, 0.1] (e.g., np.linspace(0,0.1,11)) and plot two accuracy vs epsilon curves (with y-axis range [0, 1]) on two separate plots: one for the whitebox attacks and one for blackbox attacks. How effective is random noise as an attack? (Note: in the code, whitebox and blackbox accuracy is computed simultaneously)

Random noise is not very effective as an attack, as noticed by the lack of decrease in performance in both the whitebox (which it has information to) and blackbox model. The performance generally stay above 80 percent and often 90 percent for various epsilon values.

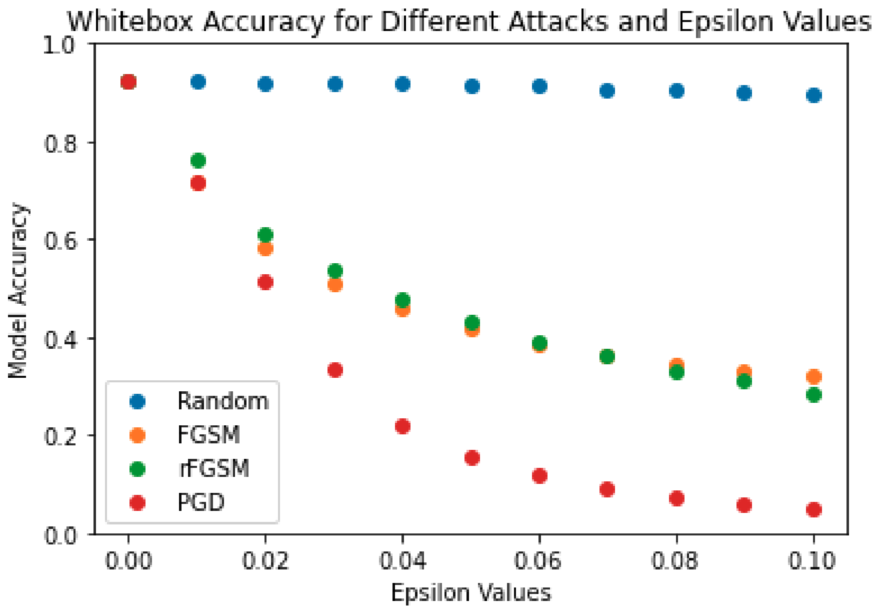

Random Attack Whitebox Performance for Various Epsilon Values
Model Accuracy
Epsilon Values
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.02
0.04
0.06
0.08
0.10


Random Attack Blackbox Performance for Various Epsilon Values
Model Accuracy
Epsilon Values
0.0
0.2
0.4
0.6
0.8
1.0
0.00
0.02
0.04
0.06
0.08
0.10

(c) (10 pts) Whitebox Attack - Using your pre-trained "NetA" as the whitebox model, measure the whitebox classifier's accuracy versus attack epsilon for the FGSM, rFGSM, and PGD attacks. For each attack, test at least eleven ϵ values across the range [0, 0.1] (e.g., np.linspace(0,0.1,11)) and plot the accuracy vs epsilon curve. Please plot these curves on the same axes as the whitebox plot from part (b). For the PGD attacks, use perturb_iters = 10 and α = 1.85 * (ϵ / perturb_iters). Comment on the difference between the attacks. Do either of the attacks induce the equivalent of "random guessing" accuracy? If so, which attack and at what ϵ value? (Note: in the code, whitebox and blackbox accuracy is computed simultaneously)

The PGD was the most effective of the attacks, achieving the lowest accuracy at each epsilon value. This gap seems to increase for each epsilon until PGD approaches the probabilistic baseline at higher epsilons. Meanwhile, the FGSM and rFGSM perform very similarly, with the rFGSM having marginally lower accuracy for higher epsilon (and FGSM marginally lower for lower epsilon). All three techniques are vastly better than the random noise attack. The random noise attack is able to reach an accuracy of about 0.9, the FGSM and rFGSM are able to reach an accuracy of about 0.28, and the PGD is able to reach an accuracy of about 0.05.

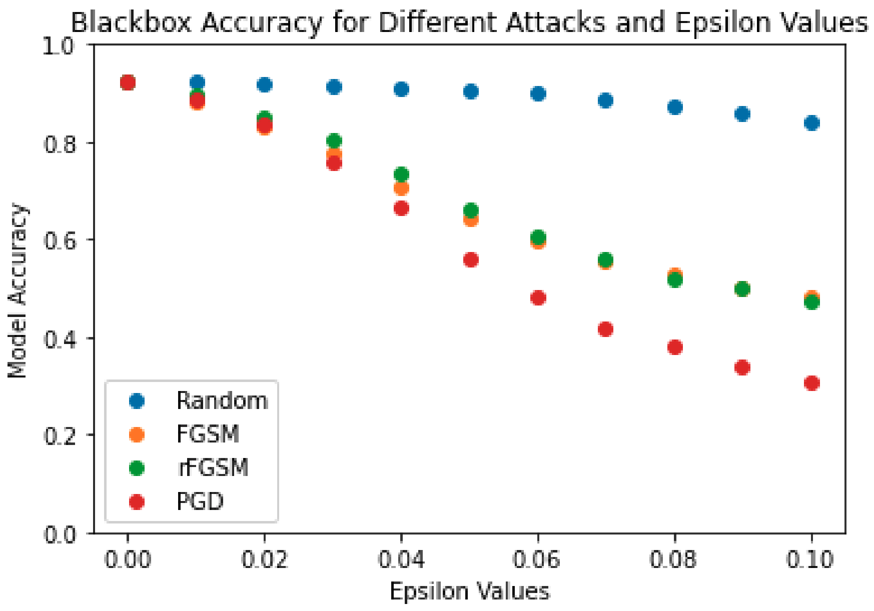
There are 10 classes in the dataset. Thus, the probabilistic baseline for the dataset is 0.1, which represents random guessing. Only the PGD is able to achieve this low (or lower) of an accuracy, for epsilons greater than or equal to 0.07.



(d) (10 pts) Blackbox Attack - Using the pre-trained "NetA" as the whitebox model and the pretrained "NetB" as the blackbox model, measure the ability of adversarial examples generated on the whitebox model to transfer to the blackbox model. Specifically, measure the blackbox classifier's accuracy versus attack epsilon for both FGSM, rFGSM, and PGD attacks. Use the same ϵ values across the range [0, 0.1] and plot the blackbox model's accuracy vs epsilon curve. Please plot these curves on the same axes as the blackbox plot from part (b). For the PGD attacks, use perturb_iters = 10 and α = 1.85 * (ϵ/perturb_iters). Comment on the difference between the blackbox attacks. Do either of the attacks induce the equivalent of "random guessing" accuracy? If so, which attack and at what ϵ value? (Note: in the code, whitebox and blackbox accuracy is computed simultaneously)

The PGD was the most effective of the attacks, achieving the lowest accuracy at each epsilon value. This gap seems to slowly increase for each epsilon. Meanwhile, the FGSM and rFGSM perform very similarly, with the rFGSM having negligibly lower accuracy for higher epsilon (and FGSM marginally lower for lower epsilon). All three techniques are vastly better than the random noise attack. The random noise attack is able to reach an accuracy of about 0.84, the FGSM and rFGSM are able to reach an accuracy of about 0.48, and the PGD is able to reach an accuracy of about 0.3.

There are 10 classes in the dataset. Thus, the probabilistic baseline for the dataset is 0.1, which represents random guessing. None of the models are able to achieve this low of an accuracy.



(e) (5 pts) Comment on the difference between the attack success rate curves (i.e., the accuracy vs. epsilon curves) for the whitebox and blackbox attacks. How do these compare to effectiveness of the naive uniform random noise attack? Which is the more powerful attack and why? Does this make sense? Also, consider the epsilon level you found to be the "perceptibility threshold" in Lab 1.b. What is the attack success rate at this level and do you find the result somewhat concerning?

The attack success rate curves have the same relative performance differences between attacks, with a stronger polynomial decay in the whitebox attacks than the blackbox attacks (and a smaller limit approached). The whitebox attacks generally approach their performance limit, whereas with the blackbox attack, the continue decreasing for longer (more steadily), as without having the information of the model it is harder to decrease the performance (less educated, requires proving). The performance decreases faster for the whitebox attacks, which makes sense, as the attacks can use gradient information based on the whitebox architecture, whereas with the blackbox, it does not have this luxury. All attacks perform much better than the random noise attack, which does not use any gradient information and rather adds noise, not allowing it to decrease the performance to anything lower than 0.84. The PGD attack is the most powerful attack as it achieves the lowest accuracy of all the attacks. This makes sense, as it takes advantage of the gradient as in FGSM, and is able to refine its step over several iterations to perturb it as optimally as possible towards the decision boundary.

At epsilon = 0.05, I find that the blackbox attack starts to create a gap between the attacks. The whitebox models have an accuracy of 0.15 for PGD, 0.45 for FGSM, 0.9 for random noise attack. The blackbox models have an accuracy of 0.55 for PGD, 0.65 for FGSM, 0.9 for random noise attack. This is concerning in terms of performance as this is merely the threshold for which I noticed some of the noise, but it is not hard to classify as a human. The fact that the performance can be decreased by 30 percent using a PGD attack on a blackbox model (NO INFORMATION) on something with small perceivable amounts of noise is concerning. On the whitebox model, it is able to achieve significant decreases at small epsilons for which the noise may not even be very visible, which is definitely concerning.

4 Lab 3: Adversarial Training (40 pts + 10 Bonus)

(a) (5 pts) Starting from the given "Model Training" code, adversarially train a "NetA" model using a FGSM attack with $\epsilon = 0.1$, and save the model checkpoint as "netA_advtrain_fgsm0p1.pt". What is the final accuracy of this model on the clean test data? Is the accuracy less than the standard trained model? Repeat this process for the rFGSM attack with $\epsilon = 0.1$, saving the model checkpoint as "netA_advtrain_rfgsm0p1.pt". Do you notice any differences in training convergence when using these two methods?

FGSM accuracy: 0.72370

rFGSM accuracy: 0.88680

The FGSM model is a significant decrease in performance from the standard model (92 percent accuracy to 72 percent accuracy). Meanwhile, the rFGSM model undergoes a small decrease in performance from the standard model (92 percent accuracy to 89 percent accuracy).

Yes, there appears to be a larger difference in terms of performance between the two methods, especially as it relates to convergence. The rFGSM is able to reach a much higher accuracy due to its more stable convergence, in which it steadily improve from epoch to epoch, whereas FGSM was more unstable and its performance bounced around and was not able to continuously increase accuracy in a smooth manner.

(b) (5 pts) Starting from the given "Model Training" code, adversarially train a "NetA" model using a PGD attack with $\epsilon = 0.1$, perturb_iters = 4, $\alpha = 1.85$ * (ϵ/perturb_iters), and save the model checkpoint as "netA_advtrain_pgd0p1.pt". What is the final accuracy of this model on the clean test data? Is the accuracy less than the standard trained model? Are there any noticeable differences in the training convergence between the FGSM-based and PGD-based AT procedures?

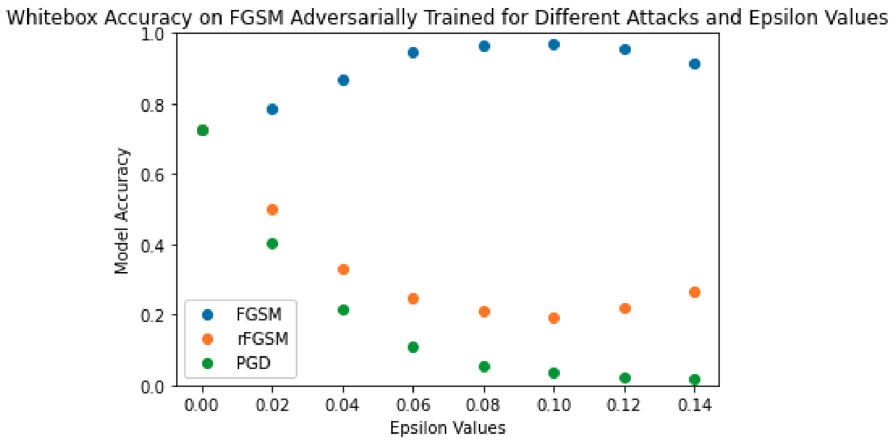
PGD accuracy: 0.87190

The PGD model undergoes a small decrease in performance from the standard model (92 percent accuracy to 87 percent accuracy).

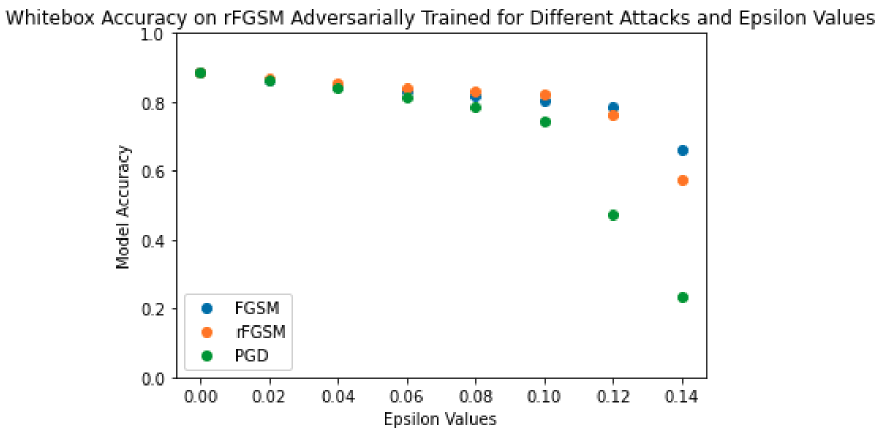
The PGD model differed in convergence from the unstable FGSM technique, and had a convergence similar in structure to the rFGSM in that it followed a steady and stable performance increase.

(c) (15 pts) For the model adversarially trained with FGSM ("netA_advtrain_fgsm0p1.pt") and rFGSM ("netA_advtrain_rfgsm0p1.pt"), compute the accuracy versus attack epsilon curves against both the FGSM, rFGSM, and PGD attacks (as whitebox methods only). Use $\epsilon = [0.0, 0.02, 0.04, . . . , 0.14]$. Please use a different plot for each adversarially trained model (i.e., two plots, three curves each). Is the model robust to all types of attack? If not, explain why one attack might be better than another. (Note: you can run this code in the "Test Robust Models" cell of the HWK5_main.ipynb notebook).

Based on the graph, it appears that the FGSM-AT model is more robust to the FGSM attack, however, it undergoes significant performance decreases for the rFGSM and PGD attacks. This makes sense as the model was trained against FGSM data such that it is used to seeing data with its perturbations and thus is similar to the samples seen while training. Its lack of random start as FGSM does not allow free adversarial training and is empirically known to not be robust to other attacks.

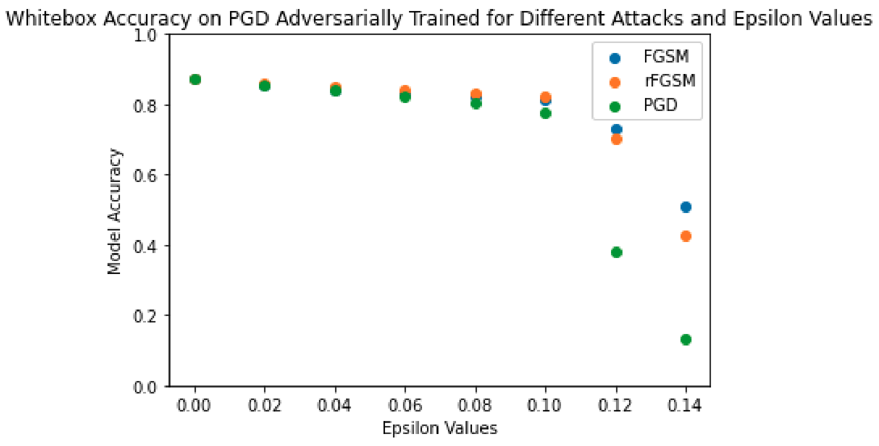


Based on the graph, it appears that the rFGSM-AT model is more robust to all attacks, however, it undergoes significant performance decreases for high epsilon values for the PGD attack (and moderate performance decrease for rFGSM). Once the training epsilon is hit, the FGSM and rFGSM attack accuracy slightly decreases but the PGD massively decreases- this makes sense as the model was trained against rFGSM data (and FGSM is within the evant space of rFGSM perturbations) such that it is used to seeing data with its perturbations and thus is similar to the samples seen while training. Furthermore, it makes sense that the rFGSM is robust to PGD up to the training epsilon, as in its paper ($https : //openreview.net/attachment?id = BJx040EFvHname = original\_pdf$), this was a key motivator of adding the random start in the first place to FGSM to allow it to perform free adversarial training.



(d) (15 pts) For the model adversarially trained with PGD ("netA_advtrain_pgd0p1.pt"), compute the accuracy versus attack epsilon curves against the FGSM, rFGSM and PGD attacks (as whitebox methods only). Use ϵ = [0.0, 0.02, 0.04, . . . , 0.14], perturb_iters = 10, $\alpha = 1.85*(\epsilon/\text{perturb\_iters})$. Please plot the curves for each attack in the same plot to compare against the two from part (c). Is this model robust to all types of attack? Explain why or why not. Can you conclude that one adversarial training method is better than the other? If so, provide an intuitive explanation as to why (this paper may help explain: https://arxiv.org/pdf/2001.03994.pdf). (Note: you can run this code in the "Test Robust Models" cell of the HWK5_main.ipynb notebook).

Based on the graph, it appears that the PGD-AT model is more robust to all attacks, however, it undergoes significant performance decreases for high epsilon values for the PGD attack (and moderate performance decrease for rFGSM). This is because once the epsilon increases beyond what it was trained on, the perturbed samples no longer represent data that could have been in the training distribution, and thus the model can not generalize. Furthermore, the PGD is robust to all forms of attack as FGSM and rFGSM are within the event space of a PGD attack (the PGD is just a more refined perturbation) and thus the model is used to training samples of that type. We can not deduce a best adversarial training method, as both the PGD and rFGSM have similar performance at each epsilon and are robust to the attacks.



(e) (Bonus 5 pts) Using PGD-based AT, train a at least three more models with different ϵ values. Is there a trade-off between clean data accuracy and training ϵ? Is there a trade-off between robustness and training ϵ? What happens when the attack PGD's ϵ is larger than the ϵ used for training? In the report, provide answers to all of these questions along with evidence (e.g., plots and/or tables) to substantiate your claims.

I have decided to train models with epsilon 0.05, 0.1, 0.15, and 0.2

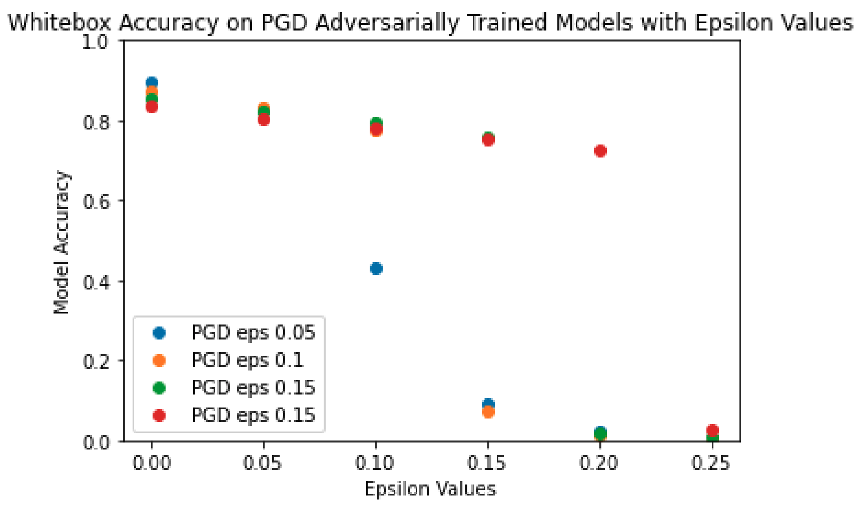
In terms of clean data accuracy, we find in the table that as we increase the training epsilon, we experience a small decrease in accuracy (about 2 percent for every increase of epsilon of 0.05). Thus, there is a small tradeoff between epsilon and clean data accuracy.

In terms of robustness, we find that higher epsilon values are generally more robust, with higher accuracies at each epsilon value (after the training epsilon is hit). For epsilons smaller than the training epsilon for a model, the models perform similarly (smaller epsilon may do marginally better), as the models with lower epsilons trained on start from a higher accuracy but decrease in accuracy more for each epsilon increase. Once the training epsilon is hit for a model, its performance decreases quickly, whereas models with higher epsilons continue to perform fairly well, demonstrating higher robustness. Thus, there is definitely a tradeoff between robustness and training epsilon.

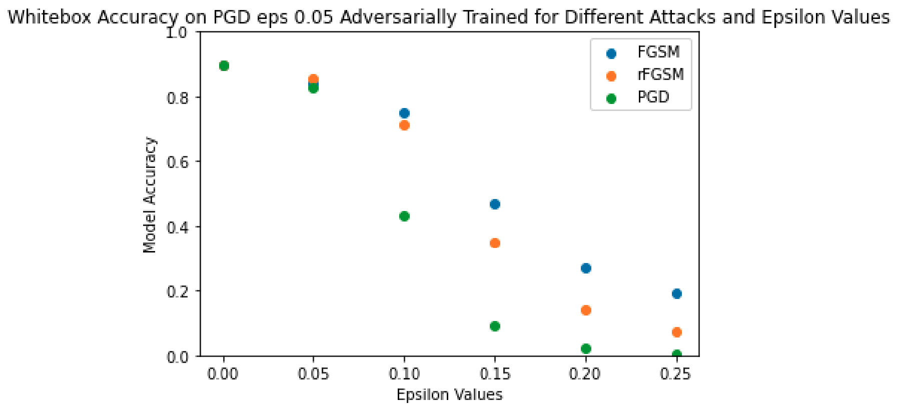
Once the PGD's epsilon is large than the epsilon used for training, we notice a sharp drop in accuracy in the model, as it loses its robustness to the attack, given that the epsilon is so large that the model has not seen training samples similar to that of the highly perturbed data. Thus, it will then perform like models with smaller epsilons and lack robustness to the perturbed data.

(f)

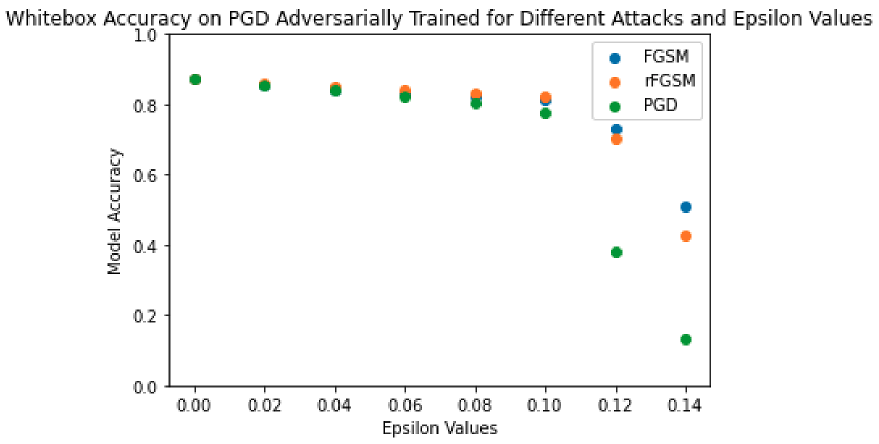
Attacks	EPS	Test Accuracy
PGD	0.05	0.89520
PGD	0.10	0.87190
PGD	0.15	0.85550
PGD	0.20	0.83740



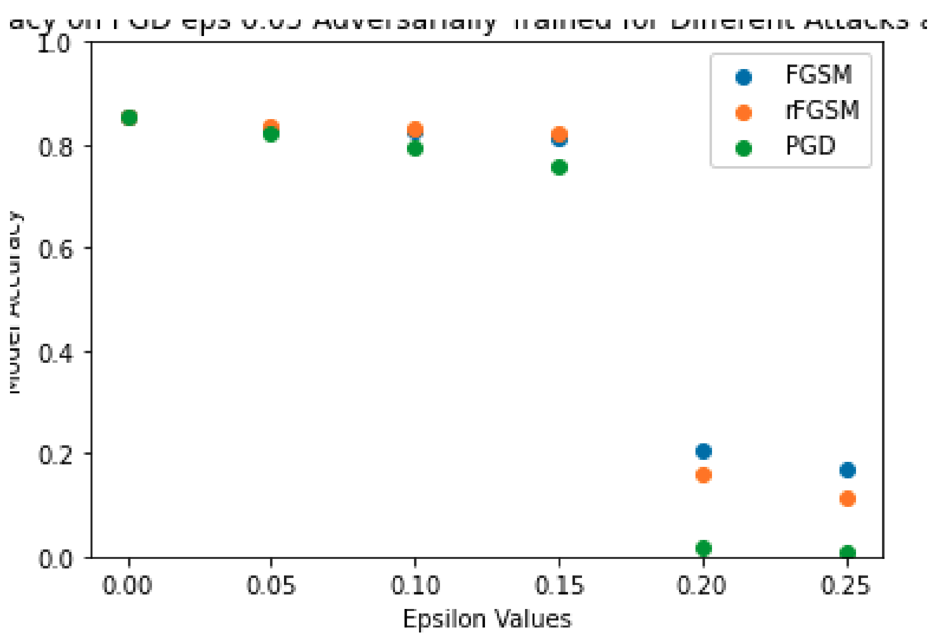
Eps 0.05:



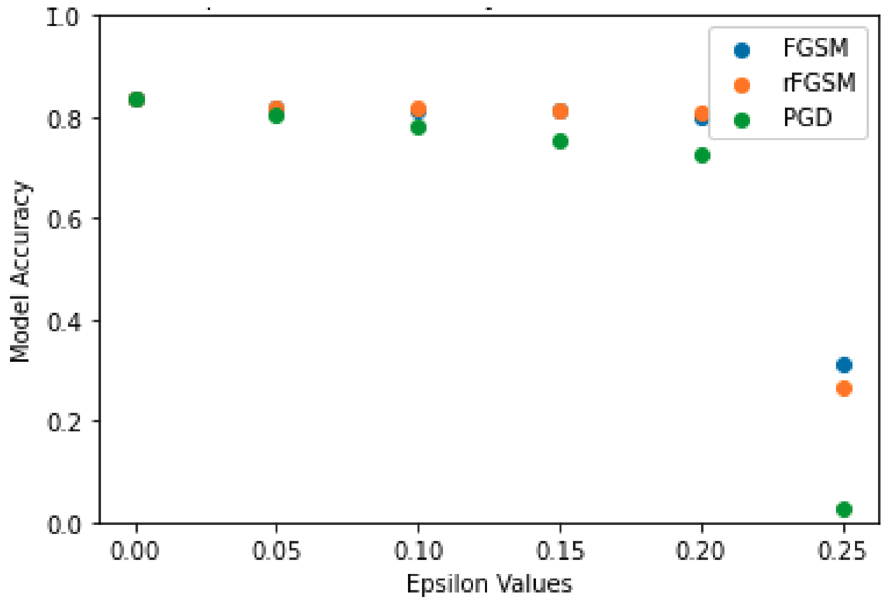
Eps 0.1:


Whitebox Accuracy on PGD Adversarially Trained for Different Attacks and Epsilon Values
Model Accuracy
Epsilon Values
FGSM
rFGSM
PGD
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

Eps 0.15:


acy on PGD eps 0.05 Adversarially Trained for Different Attacks a
Model Accuracy
Epsilon Values
FGSM
rFGSM
PGD
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.05
0.10
0.15
0.20
0.25

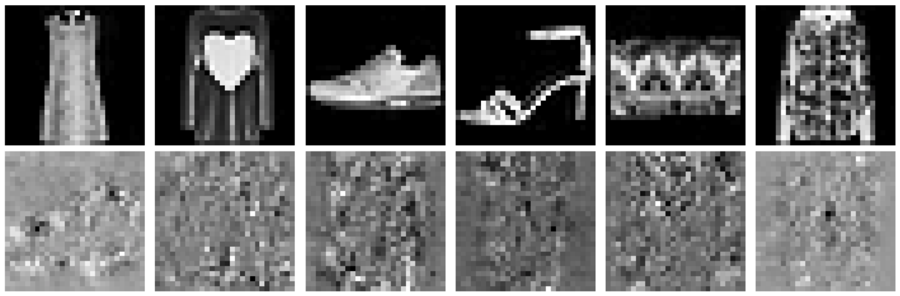
Eps 0.2:



(g) (Bonus 5 pts) Plot the saliency maps for a few samples from the Fashion-MNIST test set as measured on both the standard (non-AT) and PGD-AT models. Do you notice any difference in saliency? What does this difference tell us about the representation that has been learned? (Hint: plotting the gradient w.r.t. the data is often considered a version of saliency, see https://arxiv.org/pdf/1706.03825.pdf)

There is a massive difference in the saliency. The standard model has a higher gradient throughout the image, and focuses on the larger picture, taking in information from everywhere to perform its classification. Meanwhile, the PGD attack trained model generally has a small gradient in most of the image and then has a gradient around an outline of a key feature/edge of the object. This indicates that the model is focusing on extracting the key features of the object in making its clasification decision, and is more robust to small perturbations as introducing noise/small transformations generally won't change the collection of key features/outlined object.

Standard Saliency:



PGD Saliency: