Assignment 3

Mitchell Chatterjee
Carleton University (101141206)

The intent of this assignment was to implement PGD attack on a CNN to observe how this increases the generalization capacity of the model when presented with new data. In particular, unperturbed data. The Pytorch package was used in order to complete the assignment.

CNN Model

```
ConvNet(
  (conv1): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (fc1): Linear(in_features=1568, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=10, bias=True)
)
```

Figure 1: Model structure of the CNN

Above we can see the structure of the CNN used for the PGD attack. This is a CNN with two convolutional layers with a 5x5 kernel and stride of one. The network also includes zero padding to ensure we do not lose information from the original layers. This network also includes two fully connected linear layers. The first layer has an input of 1568 features and an output of 120 features. While the second layer which acts as the classification layer has an input of 120 features and an output of 10, corresponding to the number of classes in the MNIST dataset.

Furthermore, we use RELU for the activations in the network and max pooling after each convolutional layer to reduce the dimensionality of the data.

PGD Implementation

In order to implement PGD we follow the steps outlined in Lecture 5, Slide 10. First we add noise to the image according to a uniform distribution to get x_hat. We then perform the following steps iteratively on the data, according to the number of steps specified as a hyperparameter. First we obtain a tentative value for x_hat called z, by performing gradient ascent according to the formula in slide 10. Next we project the value of z back into the L_inf ball by using the min/max formula. This process is performed on a batch of training examples simultaneously to increase the training speed.

We encase the PGD implementation in a training loop where we train the classifier on batches of the adversarial data at a time, performing mini-batch SGD. According to the steps outlined in Lecture 5, Slide 18.

In order to achieve a targeted PGD attack we use the ‘kthvalue’ method in Pytorch to get the second most likely label while excluding the true label if this was in fact the second most likely value.

Finally, to test the data on the training set, we stored the values of the perturbed training data in an array. Which we tested against the model after training was complete. While also testing the model against the original training data.

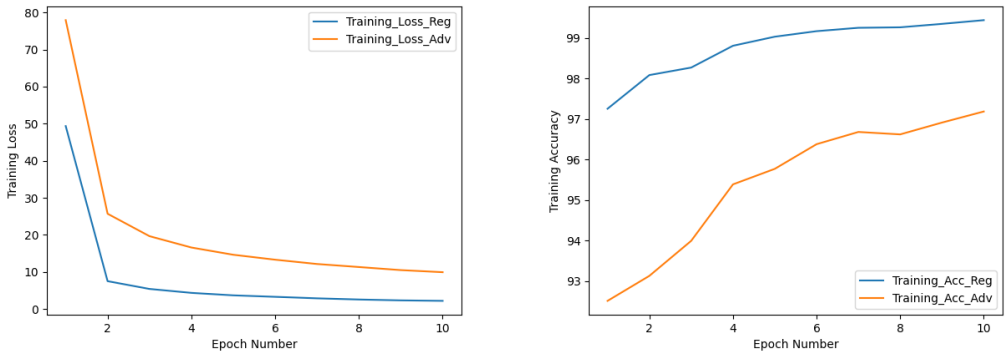

Figure 2: Training Loss and Training Accuracy for 20-step non-targeted PGD with eps=0.3 and step_size=0.02

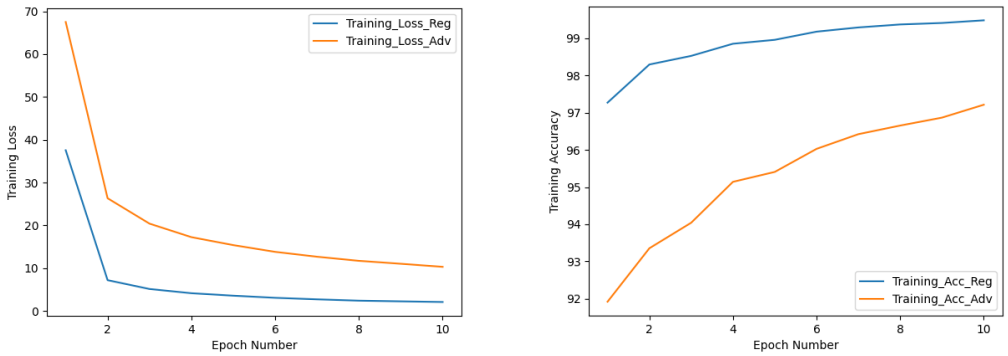

Figure 3: Training Loss and Training Accuracy for 1-step non-targeted PGD with eps=0.3 and step_size=0.5

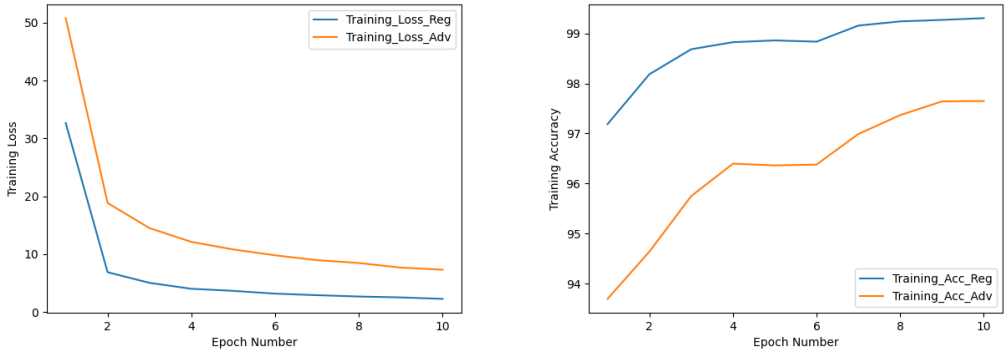

Figure 4: Training Loss and Training Accuracy for 20-step targeted PGD with eps=0.3 and step_size=0.02

As we can see from the results, the training loss on the adversarial data was always greater than the result on the original data. While the training accuracy on the adversarial data was always worse than the accuracy on the original data. There is a direct correlation between the increasing accuracy on the adversarial data and the increasing accuracy on the original data. This is a result of the adversarial data introducing noise to the original data and providing greater generalization capacity to the CNN.

Test Accuracy

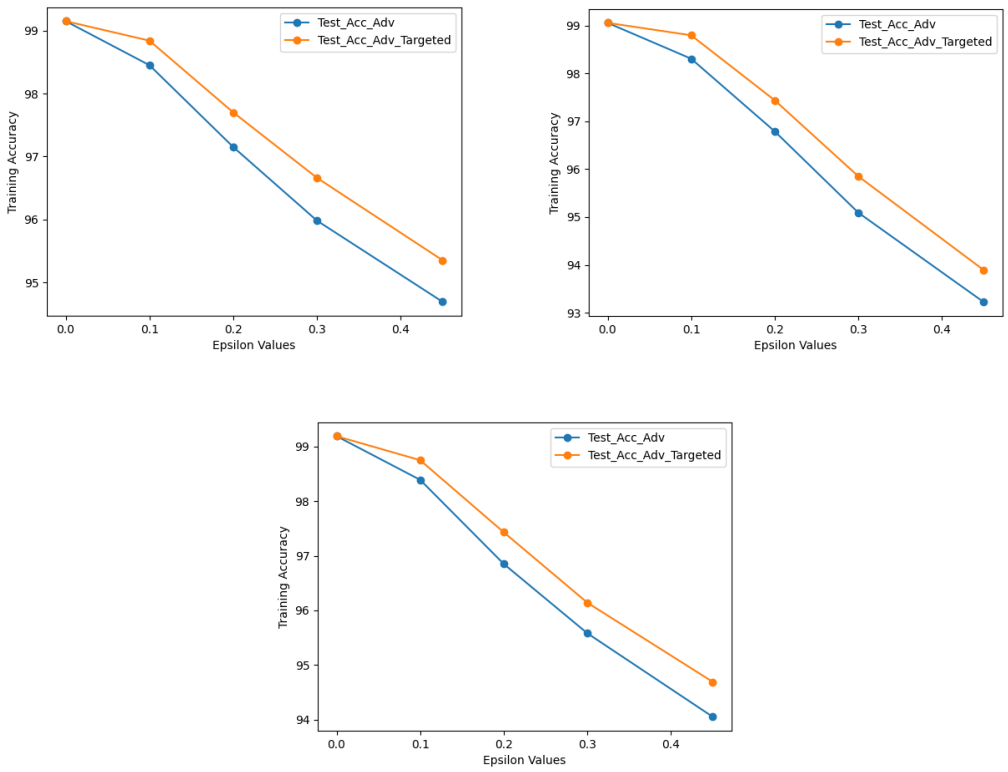

Figure 5: (From left to right) First Model, Second Model, Third Model

The figure above shows the results of the testing accuracy of the first, second and third models with different epsilon values for the PGD attack. As expected, the accuracy of the models decreases with the increasing value of epsilon. This was expected as the value of epsilon bounds the size of the L_inf ball that controls the distance the adversarial examples are from the original data. This allows for more noise in the data as the value of epsilon increases. Consequently the testing accuracy becomes worse.

The first model performs the best in this case as it underwent the most rigorous training regime. Non-targeted 20-step PGD attack. This made its generalization capacity the most robust to underlying noise. The model that was trained with targeted 20-step PGD attack also performed well. However, it is likely that the noise used to train the model was bounded by the fact that we always chose the second most likely class as the target. This in turn limited the type of noise the network was exposed to. As a result the network did not generalize as well as the non-targeted attack. Finally, the model with non-targeted 1-step PGD also performed well. However, it noticeably dropped off faster than the other two models. This is a result of the limited number of iterations used to attain a good value for the PGD attack. As the initial step size was

large compared to the others, it still performed well. However, the limited number of iterations cut the algorithm short from ascending away from the optimal point. In other words, it performed a single large leap based on the initial gradient. While the other models experienced a series of more carefully chosen steps away from the true optimum, based on multiple gradient updates.

Additional Images

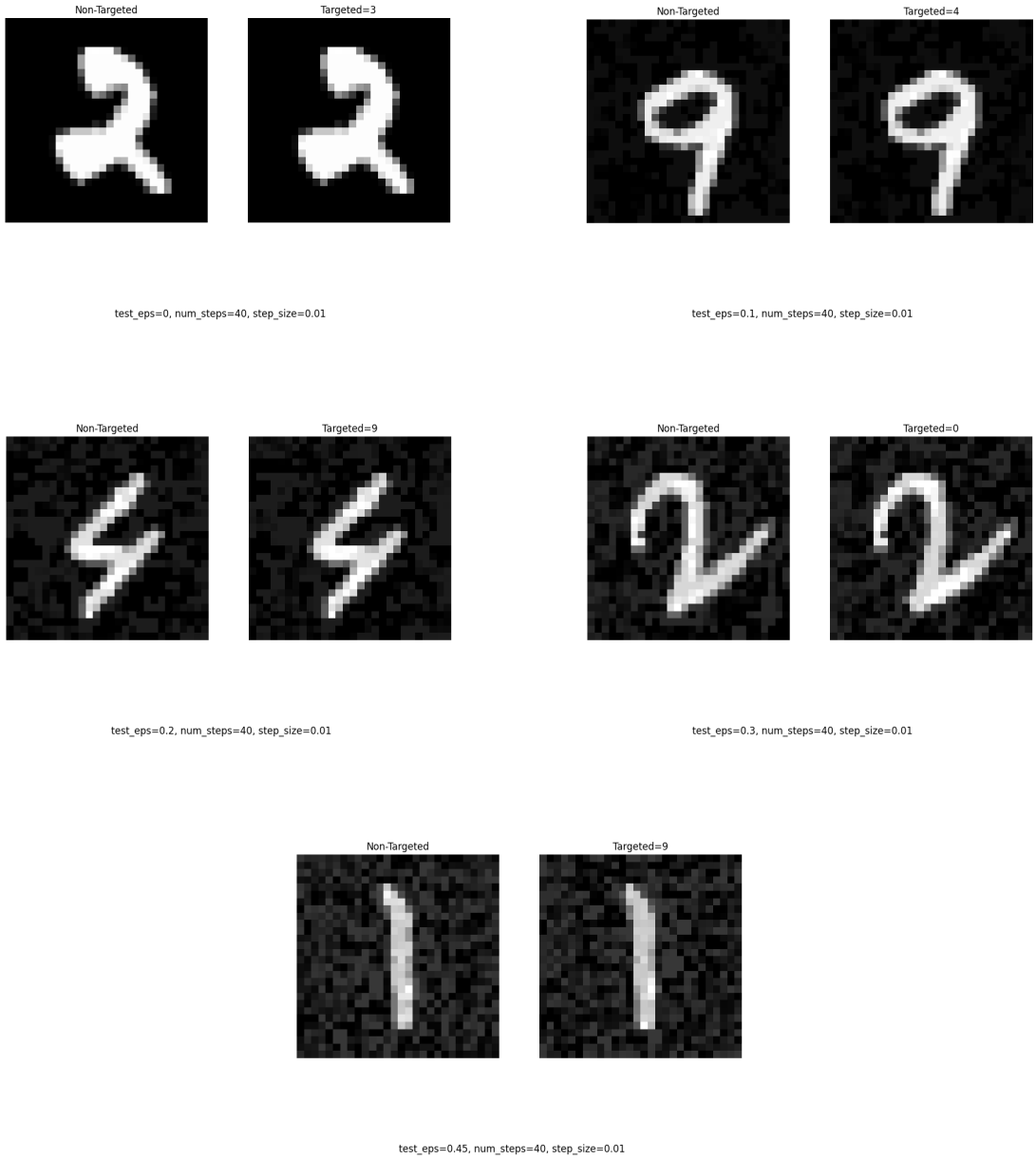

Figure 6: Outputs from the various PGD attacks during the testing phase