Assignment 03: Projected Gradient Decent attack
CSI5340 – Introduction to deep learning and reinforcement learning
Cristopher McIntyre Garcia
300025114

Designs

The CNN used during this assignment is a simple model composed of 3 convolution layers with 5x5 kernels and Relu as the activation function, a fully connected layer with batch normalization, and a soft-max layer. With this architecture, the model is able to attain an accuracy of 98.94% on the MNIST testing set after 20 epochs of training (20000 iterations). A pre-trained model is saved in the "models" folder of this assignment and can be reloaded during multiple tests for consistency. To train the model, cross entropy loss is used as the loss function, and mini-batch stochastic gradient descent, with batch size of 64, learning rate of 0.001, and momentum of 0.9, is utilized as the optimizer.

The purpose of a PGD attack is to modify an image slightly so that it is misclassified by a given model. These modified images are called adversarial examples, and the modification (or perturbation) needs to be small enough as to not be perceptible to the human eye. The two variations of this attack are non-targeted and targeted PGD. In both attacks, hyper-parameters ϵ, α, and *steps* are given and utilized in the same way. The hyper-parameter ϵ defines the bounds of a l^∞ norm ball in which every adversarial example must be found. The hyper-parameter α defines the step size, which will be multiplied by either 1 or -1 to change a given pixel's value at every step. Finally, the hyper-parameter *steps* defines the number of iterations the PGD algorithm will perform before returning the adversarial example.

The difference between the two algorithm variants lies in the direction in which they decide to stray the data from the original input. In the non-targeted version, the perturbation added to the image at every step attempts to stray the input data from the real label in any direction. It does so by maximizing the loss between the perturbed data and the true classification. In the targeted version, on the other hand, the perturbation added to the image at every step tries to stray the input data from the real label in the direction of a targeted label. It does so by minimizing the loss between the perturbed data and the targeted classification.

To perform a non-targeted PGD attack, we start by drawing a random perturbation from a uniform distribution. The uniform distribution is defined by the hyper-parameter ϵ which dictates the lower and upper bounds of the distribution l^∞ norm ball. The perturbation is then added to the input data, creating the basis for the adversarial example. Using a loss function (cross entropy loss in this case) and the gradient of the loss function with respect to the input, a new perturbed image can be produced. Again, we need to determine the gradient of the loss function with respect to the input because the goal is to perturb the data in the direction where the loss between it and the real label will be maximized. The sign of this result is taken and multiplied by the α hyper-parameter and the result is added to the perturbed image. The following equation denotes the aforementioned steps:

$$z = \hat{a}^{old} + \alpha \, sign(\nabla_x loss(g(\hat{a}^{old}), y))$$

Here, $\hat{a}^{old}$ is the perturbed data, which is first perturbed by noise from a uniform distribution, x is the input data, and y is the real label. The result is the perturbed image z which is then projected towards the l^∞ norm ball using the following equation:

$$\hat{a}^{new} = min(max(z, x - \epsilon), x + \epsilon)$$

where $\hat{a}^{new}$ is the new perturbed data and x is the original input data. Finding a new z and $\hat{a}^{new}$ is done for a certain number of steps, which is also a hyper-parameter. The reasoning behind this attack is to take the original data and perturb it so as to move its classification away from the right label in any direction. The image can be thought of as being pushed away from the space where the model would classify it as the original label. The sudo code would therefore be as shown below.

```
def Non_Targeted(g, x, y):
        u = U(-ϵ, ϵ)
        â = x + u
        for _ in range(steps):
                score = g(â)
                L = Loss(score, y)
                z =  â + α  * sign(∇ₓL)
                â = min(max(z, x - ϵ), x + ϵ)
        return â
```

To perform a targeted PGD attack, very similar steps are perform. The difference is that instead of using a normal label and maximizing the loss with respect to the input, the loss with respect to the input, and and given a target label, is minimized. In the case of this assignment, the target labels are those corresponding to the largest element in the final output vector of the model excluding the value for the true label, give the original image that will be perturbed.

Because the loss is now being minimized, the formula for finding the perturbed data z is as follows:

$$z = \hat{a}^{old} - \alpha \, sign(\nabla_x loss(g(\hat{a}^{old}), \hat{y}))$$

Where $\hat{y}$ is the targeted label and all other variables are the same as in the previous method. The sudo code would therefore be as shown below.

```
def Targeted(g, x, ŷ):
        u = U(-ϵ, ϵ)
        â = x + u
        for _ in range(steps):
                score = g(â)
                L = Loss(score, ŷ)
                z =  â - α * sign(∇ₓL)
                â = min(max(z, x - ϵ), x + ϵ)
        return â
```

The difference is minor, but the targeted attack can allow an attacker to choose the label of the misclassified data. The reasoning behind this attack is to take the original data and perturb it so as to move its classification towards that of the target label, and in doing so, move it away from the original label. The image can be thought of as being pulled towards the space where the model would classify it as the target label.

Training

One technique that can be used to counter PGD attacks is adversarial training. In this style of training, at least in our case, a variation of the PGD attack is used to produce adversarial examples that can then be used as training data for the model. Below we demonstrate the difference between the regular training algorithm and the PGD attack adversarial training algorithm. The PGD attack can be either targeted or non-targeted, and everything else is kept the same.

```
def train(g, x, y):
        for _ in range(epochs):
                for _ in range(iterations)
                        score = g(x)
                        L = Loss(score, y)
                        L.backward()
                        optimizer.step()
        return g
```

```
def PGD_train(g, x, y):
        for _ in range(epochs):
                for _ in range(iterations)
                        â = PGD(x)
                        score = g(â)
                        L = Loss(score, y)
                        L.backward()
                        optimizer.step()
        return g
```

In this experiment, we train three classifier models and demonstrate their behaviours during training. The first classifier is trained using a non-targeted 20-step PGD attack adversarial training algorithm with ϵ set to 0.3 and α set to 0.02. The second classifier is trained using a non-targeted 1-step PGD attack adversarial training algorithm with ϵ set to 0.3 and α set to 0.5. The final classifier is trained using a targeted 20-step PGD attack adversarial training algorithm with ϵ set to 0.3 and α set to 0.02.

The results of training over 20 epochs are shown bellow.

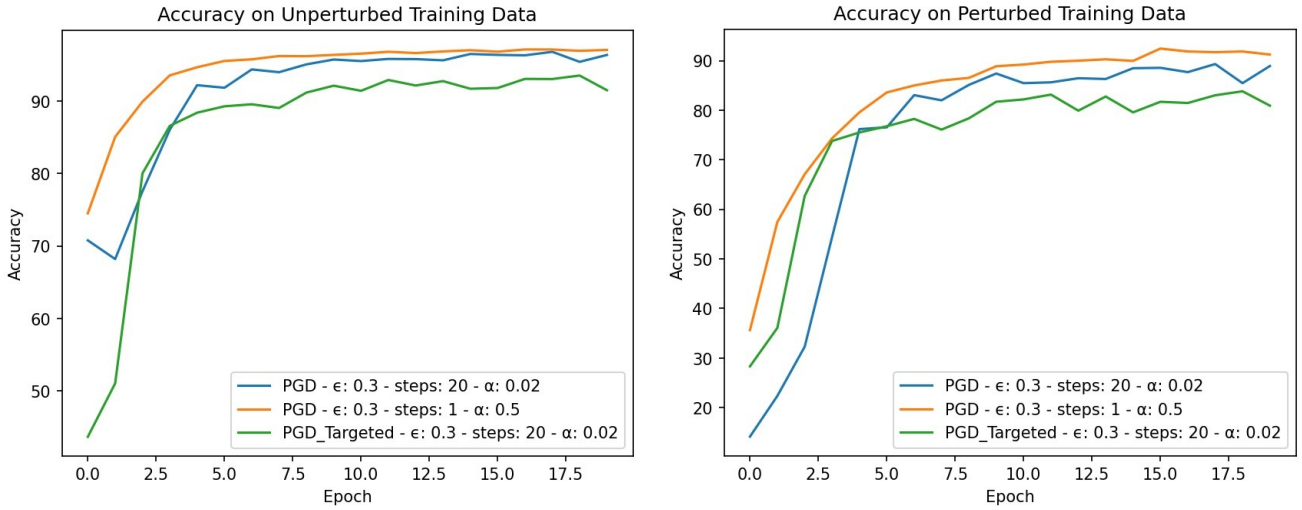

Figure 1. (a) Accuracy of unperturbed training data, (b) Accuracy of unperturbed training data

In figure 1 (a), we see that the training accuracy on the unperturbed data augments rapidly and plateaus at over 90% regardless of the PGD usage during training. All the models start with a lower accuracy when perturbed data is used, as seen in figure 1 (b), but seem to match the accuracy of the unperturbed data later on. The model using only one step PGD ends up performing better than the others. This may be due to the adversarial examples not having been optimized with enough steps to generate properly-adversarial images. The model is therefore having an easier time properly classifying the perturbed

data. If this model were to be attacked with perturbed data from a PGD attack with a larger step count, the model might not perform very well.

When comparing the targeted and non-targeted PGD attack training algorithms of the same step count, we note that the non-targeted attack performs better overall. This leads to the belief that the targeted PGD attack produces better adversarial examples, which makes the model perform worse. Because we are training on the perturbed data, the model loses a bit of its capacity to generalize and performs worse on the unperturbed data. The model also has difficulty with perturbed data because its adversarial examples are better at fooling the model. Intuitively, this makes sense because, in our case, we are using the second most probable classification from the model's output when given the non-perturbed data as the target label. This makes the model more certain about the misclassified label than it otherwise would be if it was trained with a non-targeted PGD algorithm.

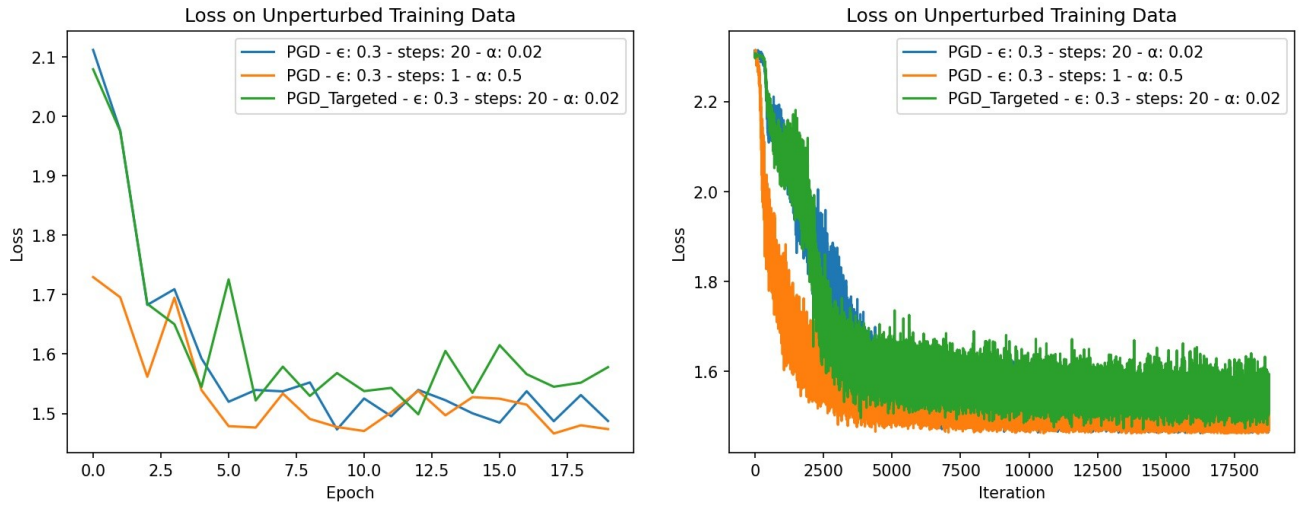

Figure 2. Loss on unperturbed training data

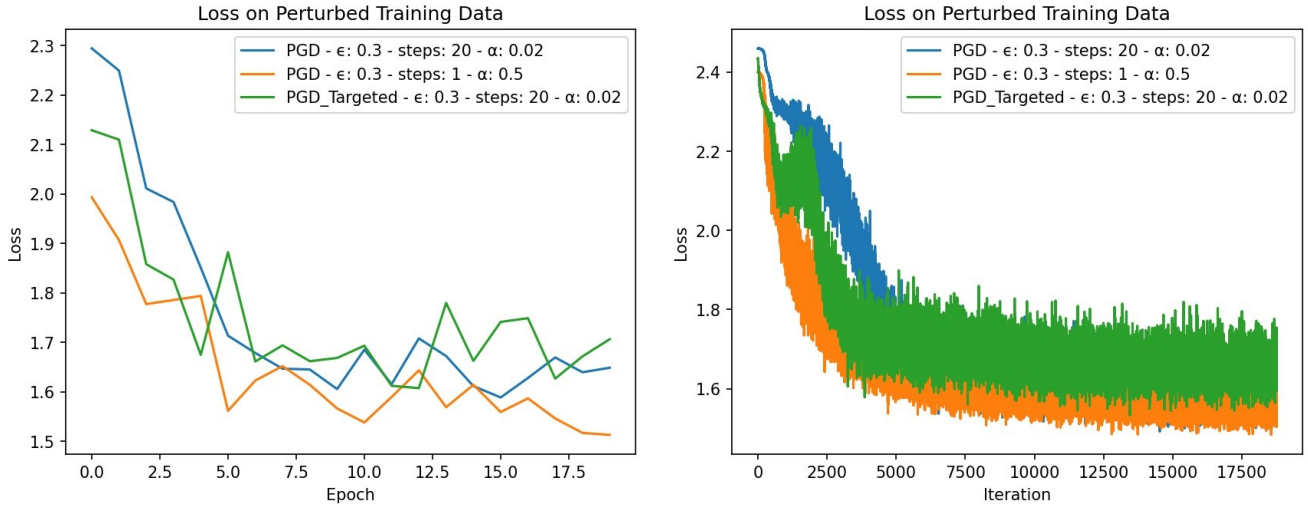

Figure 3. Loss on perturbed training data

In figures 2 and 3, we see the losses of the models over training. These curves are representative of what was discussed above and only strengthen the ideas mentioned. The single step PGD seems to perform the best over all, the non-targeted 20-step PGD model is next, and the worst performer is the

targeted model. That said, the accuracies are not representative of how the models would perform overall on different attacks. For example, the non-targeted PGD models perform well on their own perturbed data, but they may perform terribly against a targeted PGD attack.

Testing

First, we show how the models perform during training on the testing dataset.

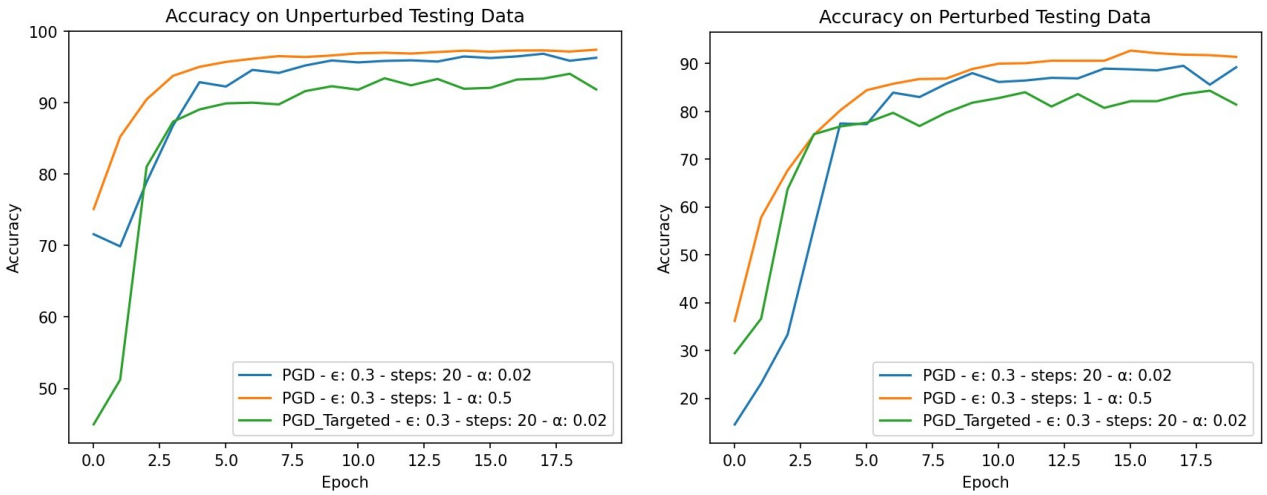

Figure 4. (a) Model accuracy of unperturbed test set while training,
(b) Model accuracy of perturbed test set while training

As seen in figure 4, the models perform similarly on the perturbed testing data as they did on the perturbed training data. The same can be said about the unperturbed testing and training data results. This means that the model is learning how to generalize properly and is not being over or under fitted to the perturbed data. That said, the model is not able to achieve the same accuracy when trained on regular data. The model trained on regular data was consistently performing with 98% accuracy or better, whereas the models trained using adversarial training never attained results over 97% on the non-perturbed testing set.

One question that I had when comparing the accuracy of the testing data and the training data was whether I had made a mistake and used the same data loader. They seemed really similar, almost too similar. It turns out they are different and the accuracy of the testing data is normally around 1% less accurate than that of the training data set accuracy. This is shown below with examples from the 1-step non-targeted PGD adversarial training with ϵ as 0.3 and α as 0.5.

Example of Accuracy on training set: [28.13, 38.41, 46.14, 51.09, 56.56, 60.87, 64.72]
Example of Accuracy on testing set: [27.35, 37.64, 45.29, 49.81, 56.23, 59.87, 64.07]

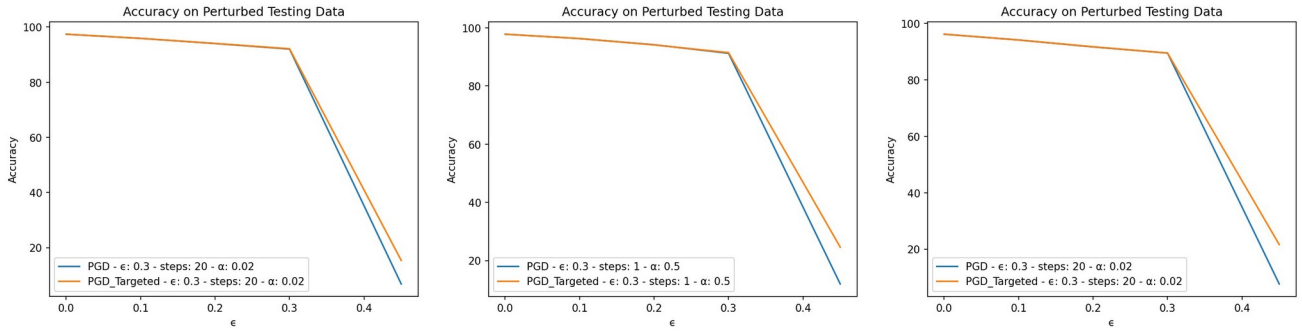

Figure 5. (a) Model: PGD - ϵ: 0.3 - steps: 20 - α: 0.02,
(b) Model: PGD - ϵ: 0.3 - steps: 1 - α: 0.5,
(c) Model: PGD_Targeted - ϵ: 0.3 - steps: 20 - α: 0.02

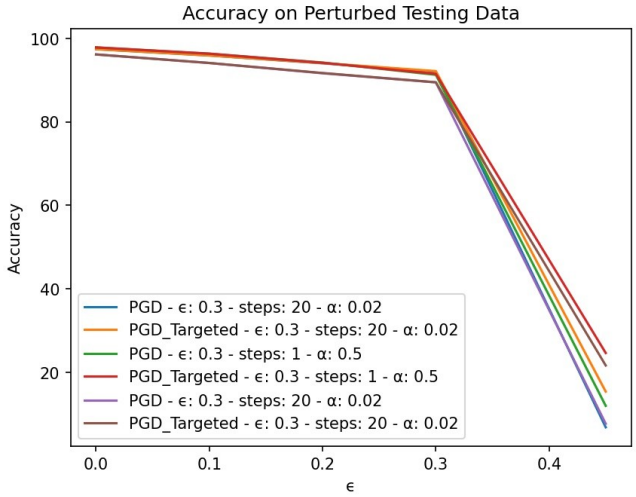

Figure 5.2 All accuracies on testing set with different ϵ values

In figure 5, we see that every classifier performs well, regardless of the type of attack, when the attacks are being made in the l^∞ norm ball used during training. The accuracy does decline as ϵ augments, however never seems to fall bellow 85%. Once the attack happens outside of the l^∞ norm ball bounds, the accuracy drops drastically. This is due to the model being fed perturbed inputs with properties that it has never seen before. In figure 5.2, the models are shown together and it is clear that those that were trained with a targeted PGD are outperforming those that were not.

Below we show some examples of adversarial examples that were used in the testing set. It is clear to see that the higher the ϵ value, the more noise is added to the image. There is not a large perceptible difference between using a non-targeted and a targeted approach. That being said, testing the model on the targeted labels with the perturbed images does yield 100% accuracy, meaning that the model classifies the perturbed data as the target classification every time. This only happens when performing attacks on an untrained classifier. When using a trained classifier, the accuracy attained can usually reach over 80%.

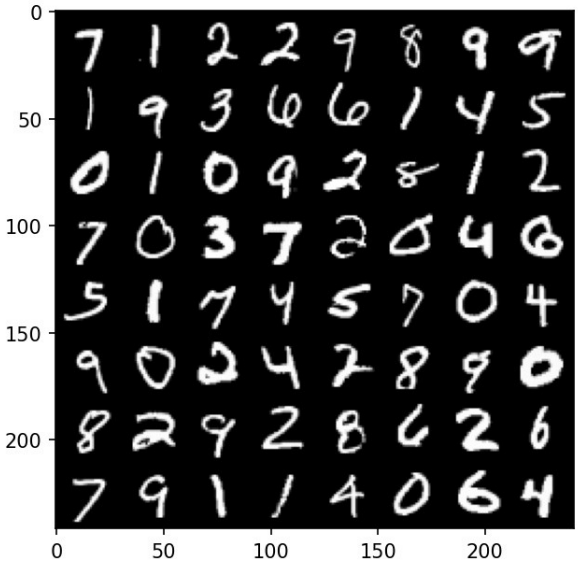

Figure 6. Examples of non-targeted perturbed data using $\epsilon = 0$

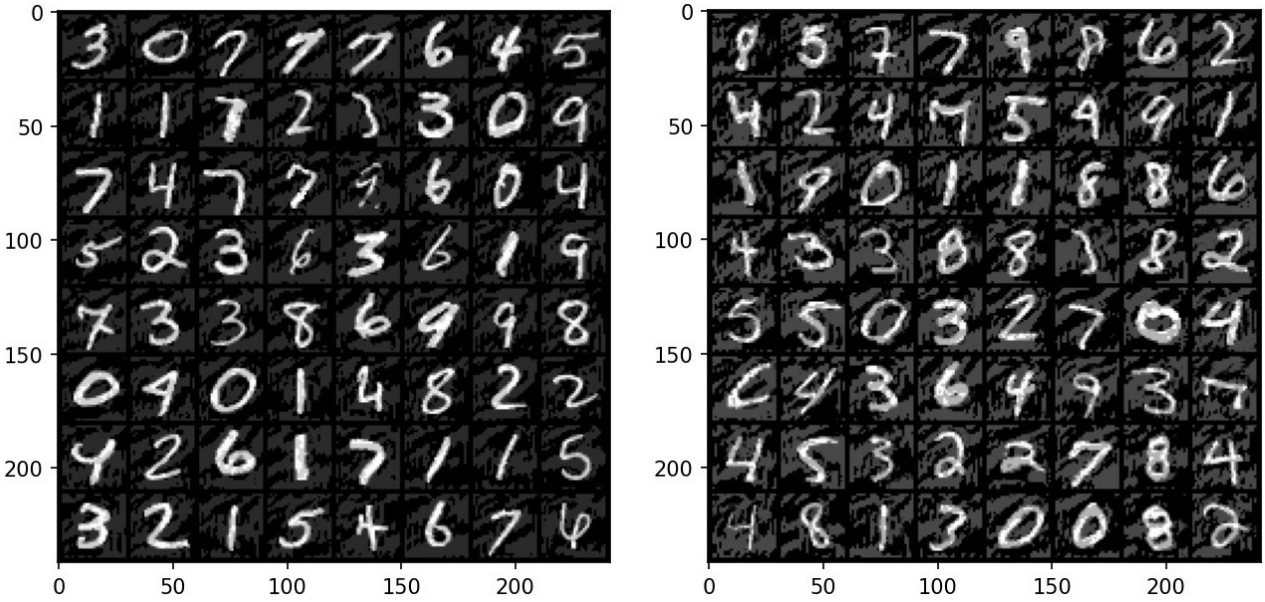

Figure 7. (a) Examples of non-targeted perturbed data using $\epsilon = 0.1$,
(b) Examples of non-targeted perturbed data using $\epsilon = 0.2$

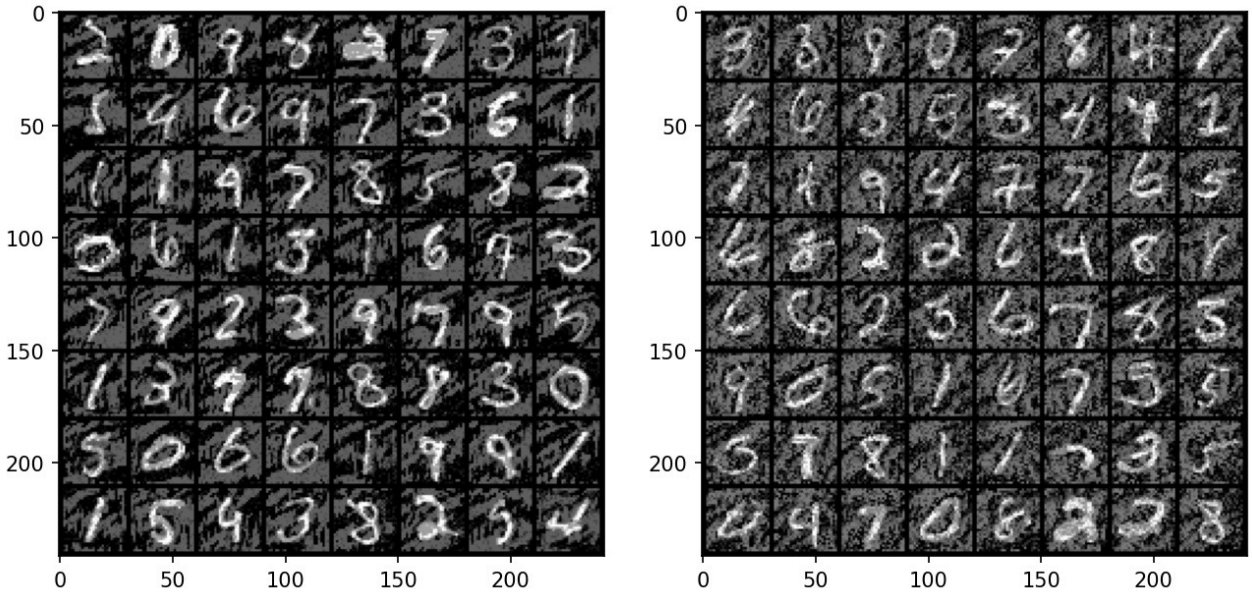

Figure 8. (a) Examples of non-targeted perturbed data using $\epsilon = 0.3$,
(b) Examples of non-targeted perturbed data using $\epsilon = 0.45$

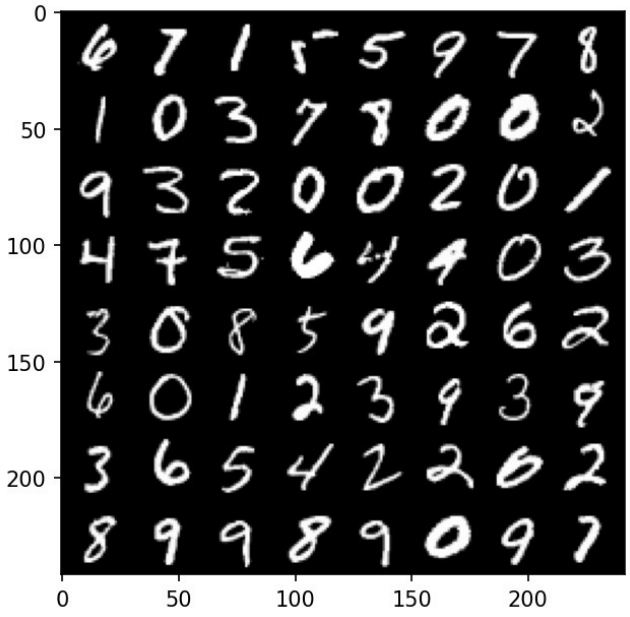

Figure 9. Examples of targeted perturbed data using $\epsilon = 0$

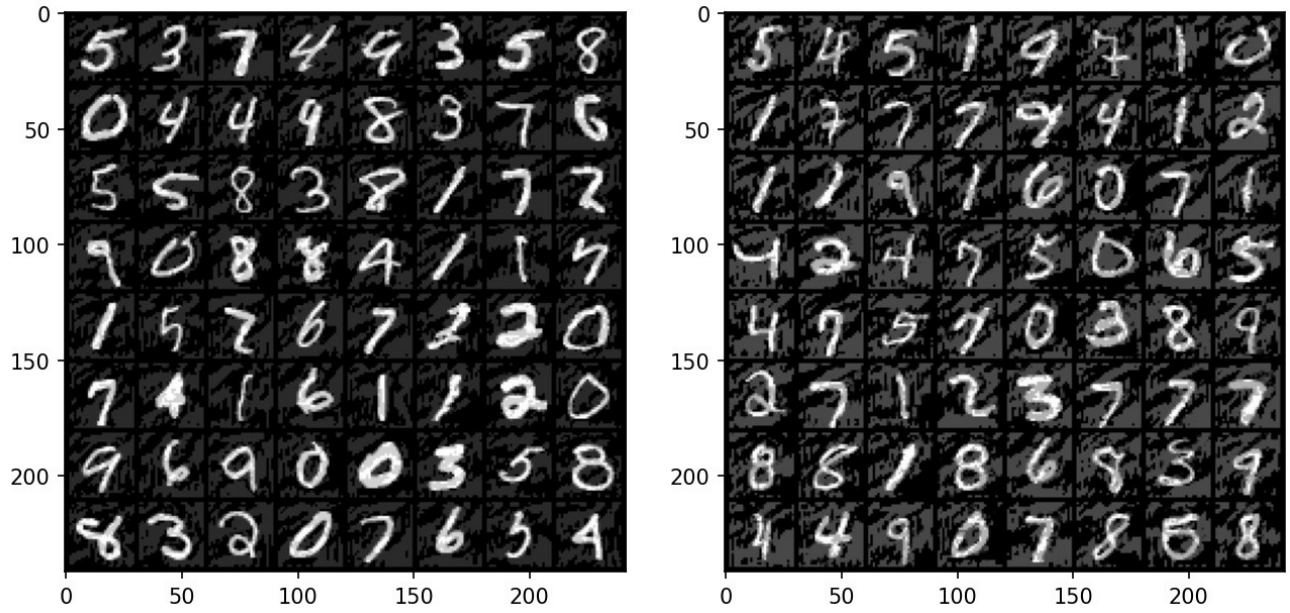

Figure 10. (a) Examples of targeted perturbed data using $\epsilon = 0.1$,
(b) Examples of targeted perturbed data using $\epsilon = 0.2$

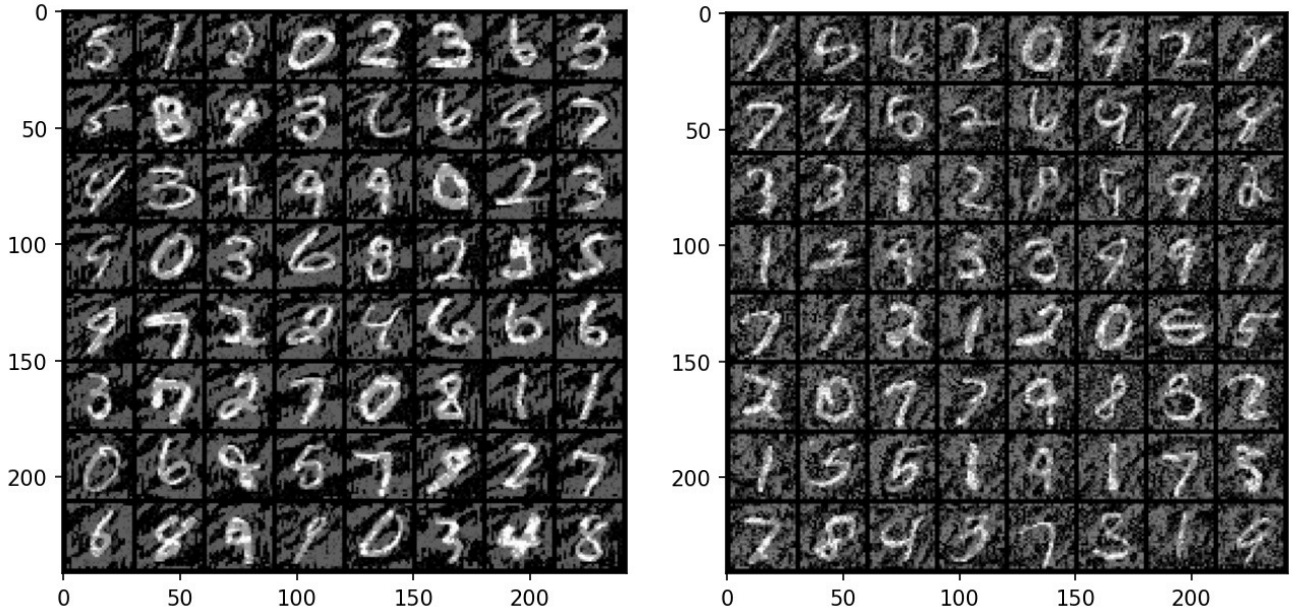

Figure 11. (a) Examples of targeted perturbed data using $\epsilon = 0.3$,
(b) Examples of targeted perturbed data using $\epsilon = 0.45$

End of report.