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

**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 decent, with batch size 64, learning rate 0.001, and momentum 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 $\epsilon$, $\alpha$, and *steps* are given and utilized in the same way. The hyper-parameter $\epsilon$ defines the bounds of a $l^\infty$ norm ball in which every adversarial example must be found. The hyper-parameter $\alpha$ 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 amount 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 from the original input data. In the non-targeted version, the perturbation added to the image at every step tries 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 a hyper-parameter $\epsilon$ which dictates the lower and upper bound of the distribution $l^\infty$ norm ball ($-\epsilon$ is the lower bound and $\epsilon$ is the upper bound). The perturbation is then added to an input data, creating the base for the adversarial example. Using a loss function (cross entropy loss in this case) and the gradient of the 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 modify the perturb the data in a direction where the loss will be maximized. The original label is used to calculate the loss. The sign of this result is taken and multiplied by the $\alpha$ 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, x is the input data, and y is the real label. The result is the perturbed image z which is then projected towards the $l^\infty$ 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, x is the original input data. Finding a new z and $\hat{a}^{new}$ is done for a certain amount of steps which is also a hyper-parameter. The reasoning behind this attack is that we are trying to take the original data and perturb it so as to move it's classification away from the right classification, in any direction. The sudo code would therefore look as bellow:

```
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, we perform very similar steps to the previous attack. The difference is that instead of using the normal labels and maximizing the loss with respect to the input, we use target labels, and minimize the loss with respect to the input and the target label. In this case, 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 we are now trying to minimize the loss between the perturbed data and targeted data, the formula for finding the perturbed data z:

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

Where $\hat{y}$ is the targeted label. The step is taken away from the current perturbed data because we are trying to move closer to the the targeted classification. The sudo code  would therefore loos as bellow:

```
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, however the targeted attack can allow the attacker to chose which the label of misclassified data. The reasoning behind this attack is that we are trying to take the original data and perturb it so as to move it's classification away from the right classification, and towards a classification that we have chosen.

**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. Bellow 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.

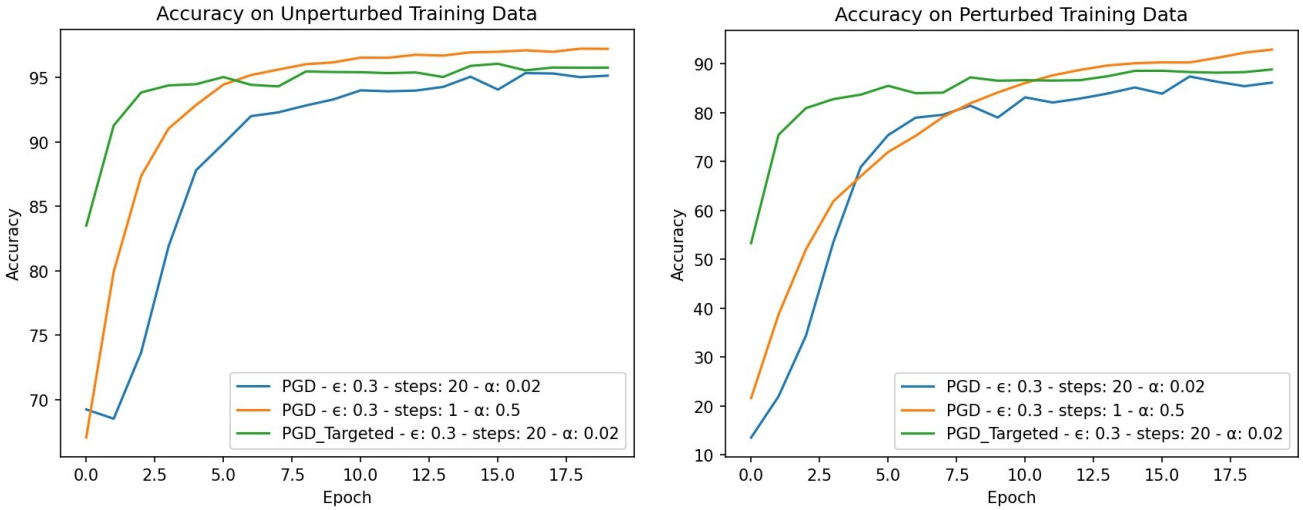


Figure 1. (a) Accuracy of unperturbed training data, (b) Accuracy of unperturbed training data

In figure 1 (a), we see that the training accuracy on the unperturbed data augments rapidly and plateaus at around 95%, regardless of the fact that PGD is being used during training. That said, the model using only one step PGD is performing better than the others early on. 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. Because of this, the model is able to perform fairly well at the very beginning. When comparing the targeted and non-targeted PGD attack training algorithms, we note that the non-targeted attack is performing better at the

beginning but worse at the end. This leads to believe that the targeted PGD attack produces better adversarial examples which are making the model perform worst at the beginning. Intuitively this makes sense because, in our case, we are using the second most probable classification from a models output given the non-perturbed data as the target label. This makes the attack misclassify faster, given it is in the direction of the closest wrong label. After around three epochs, the models seem to perform very similarly and augment at the same pace.

In figure 1 (b), we see that both the non-targeted PGD models perform much worst at the beginning than they did on the unperturbed data. This shows how the perturbed data from the targeted is performing better than that of the non-targeted methods. The same can be said regarding the one step vs 20-steps methods.

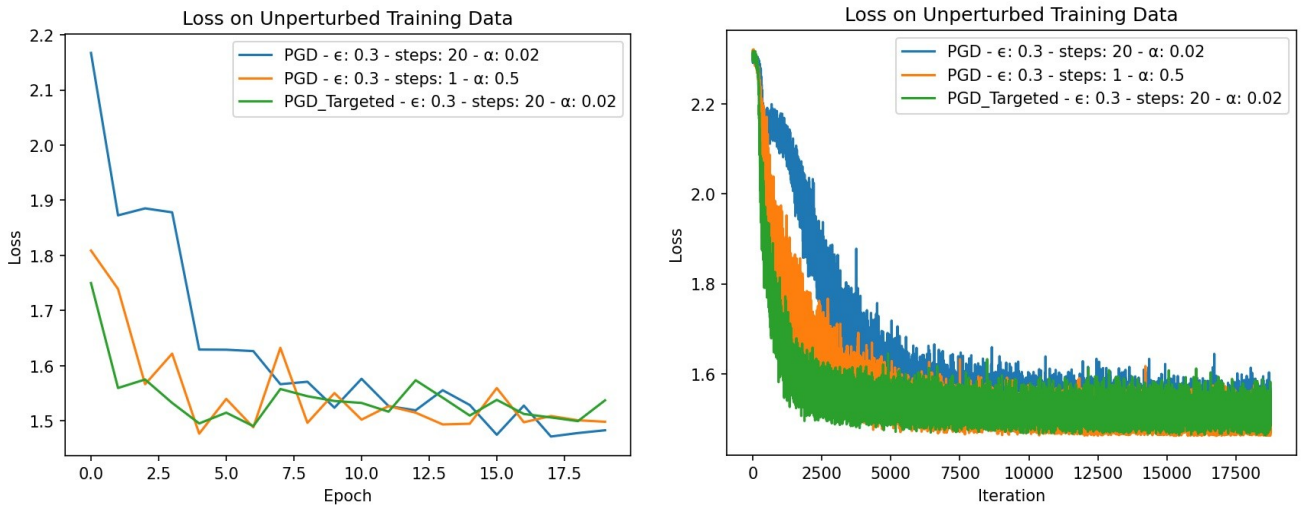


Figure 2. Loss on unperturbed training data

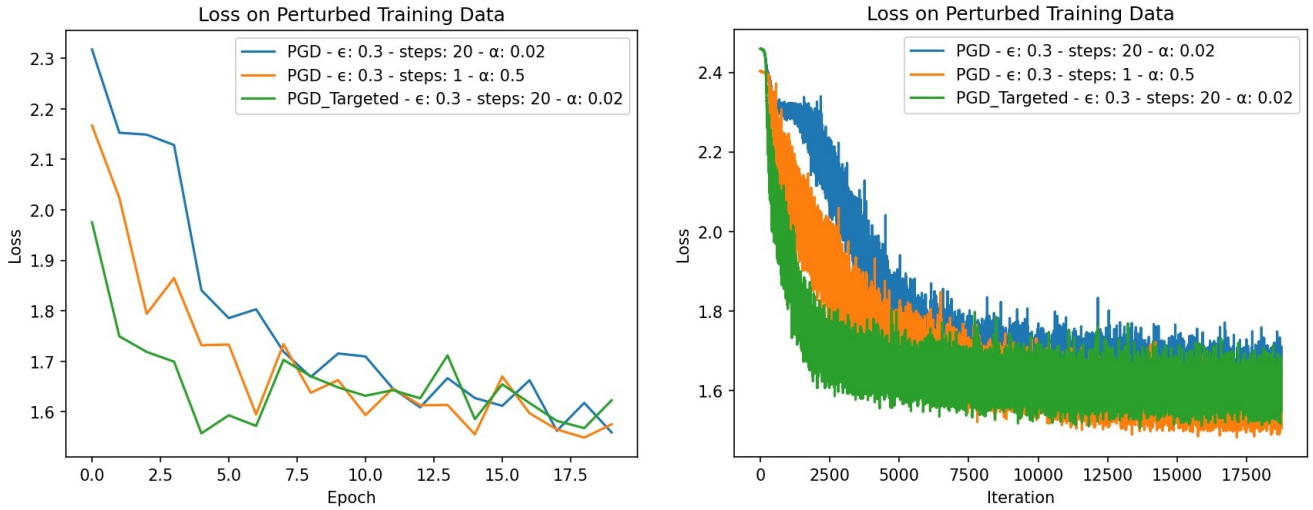


Figure 3. Loss on perturbed training data

In figure 2,

**Testing**

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

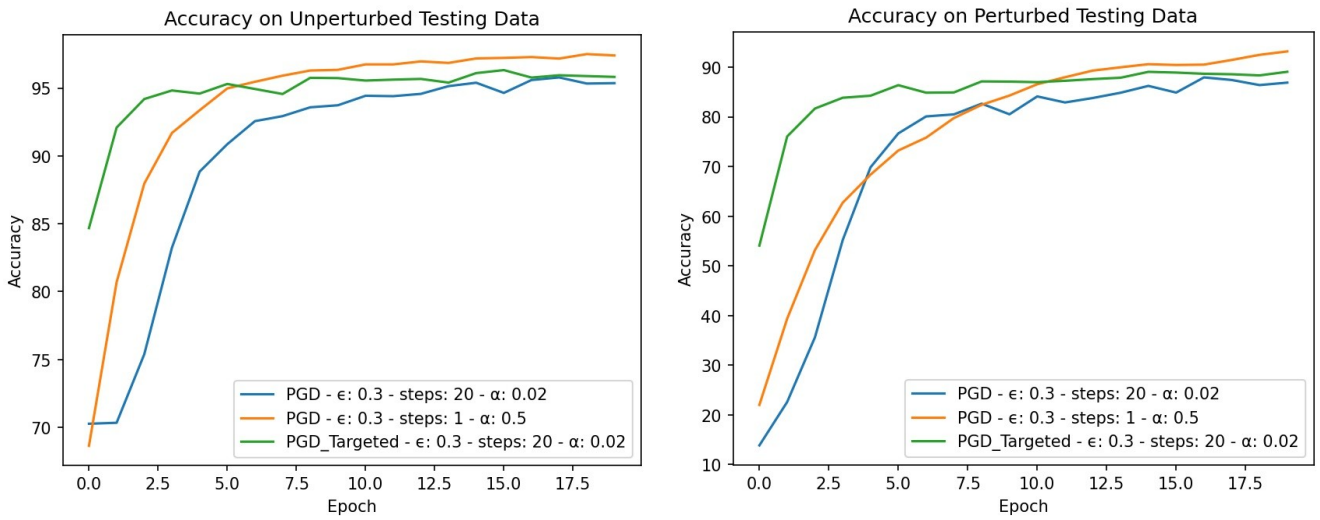


Figure 4. (a) Model accuracy of unperturbed test set while training,
(b) Model accuracy of perturbed test set while training

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 fit to the perturbed data. That said, the model is not able to achieve the same accuracy as when trained on regular data. The model trained on regular data was consistently performing with 98% accuracy or over, meanwhile the model trained using adversarial training never attains results over 97% on the non-perturbed testing set.

One question that I had when comparing the accuracy on the testing data and training data was whether I had made a mistake and used the same data loader. They seemed really similar, almost too similar. Turns out they are different and the accuracy on the testing data is normally around 1% less accurate than that of the training data set accuracy;

> 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]

Examples from 1-step non-targeted PGD adversarial training with ϵ as 0.3 and α as 0.5.

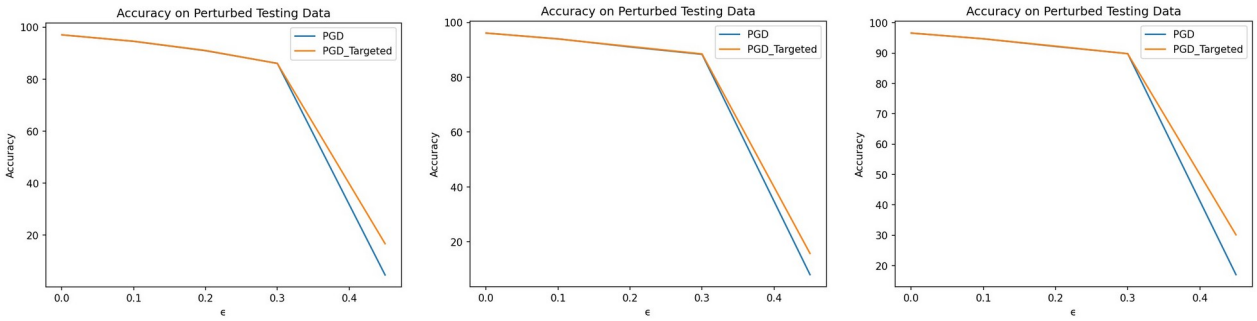


Figure 5. (a) Model: PGD - ϵ: 0.3 - steps: 20 - α: 0.02,
(b) Model: PGD - ϵ: 0.3 - steps: 1 - α: 0.5,
(c) Model: PGD_Targeted - ϵ: 0.3 - steps: 20 - α: 0.02

In figure 5, we see that every classifier performs well, regardless of the type of attack, when the attacks are being made in the $l^\infty$ 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 $l^\infty$ 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.

Bellow we show some examples of adversarial examples that were used to 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 using a non-targeted and targeted approach. That being said, testing the model on the targeted labels with the perturbed images does yield 100% accuracy, meaning that the model is classifying 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%.

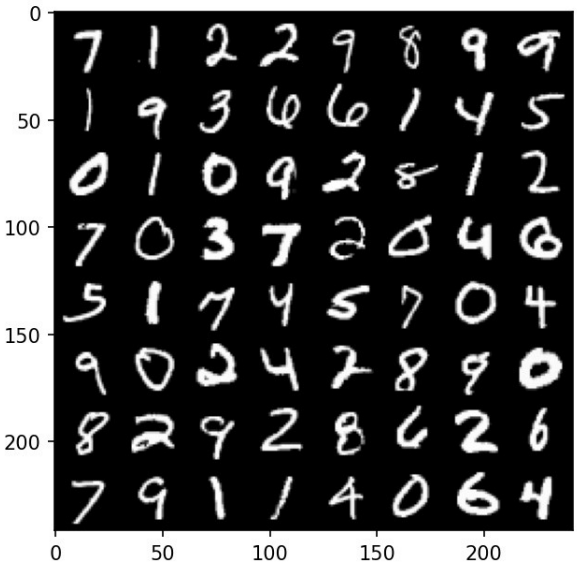

Figure 6. Examples of non-targeted perturbed data using ϵ = 0

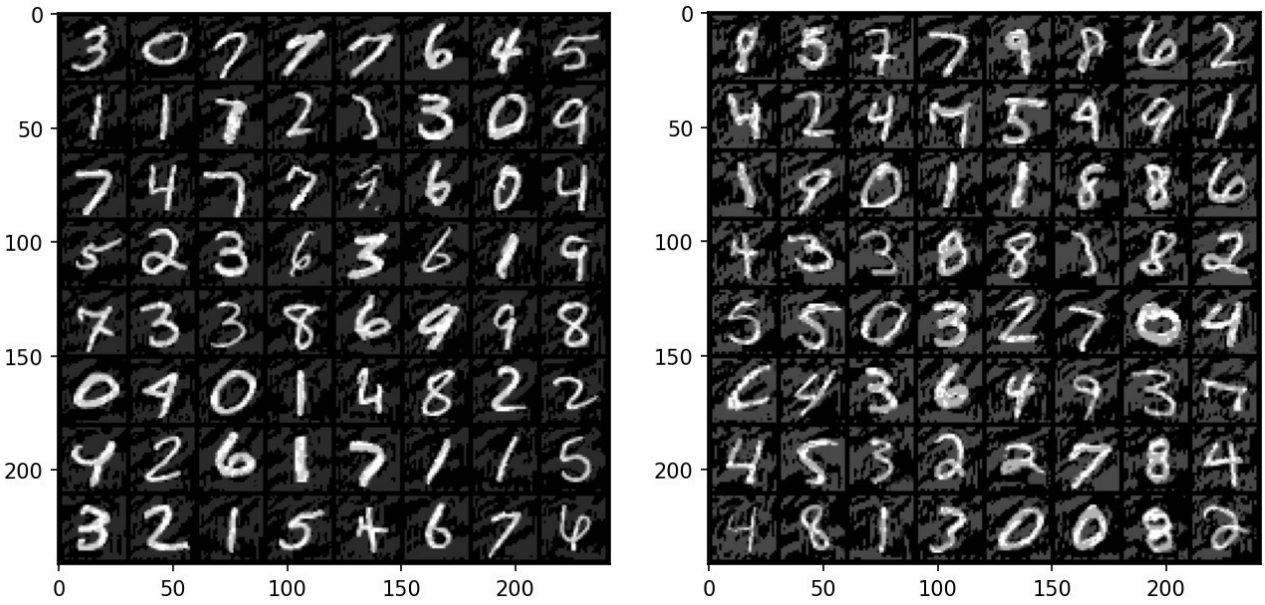


Figure 7. (a) Examples of non-targeted perturbed data using $\epsilon = 0.1$,
(b) Examples of non-targeted perturbed data using $\epsilon = 0.2$

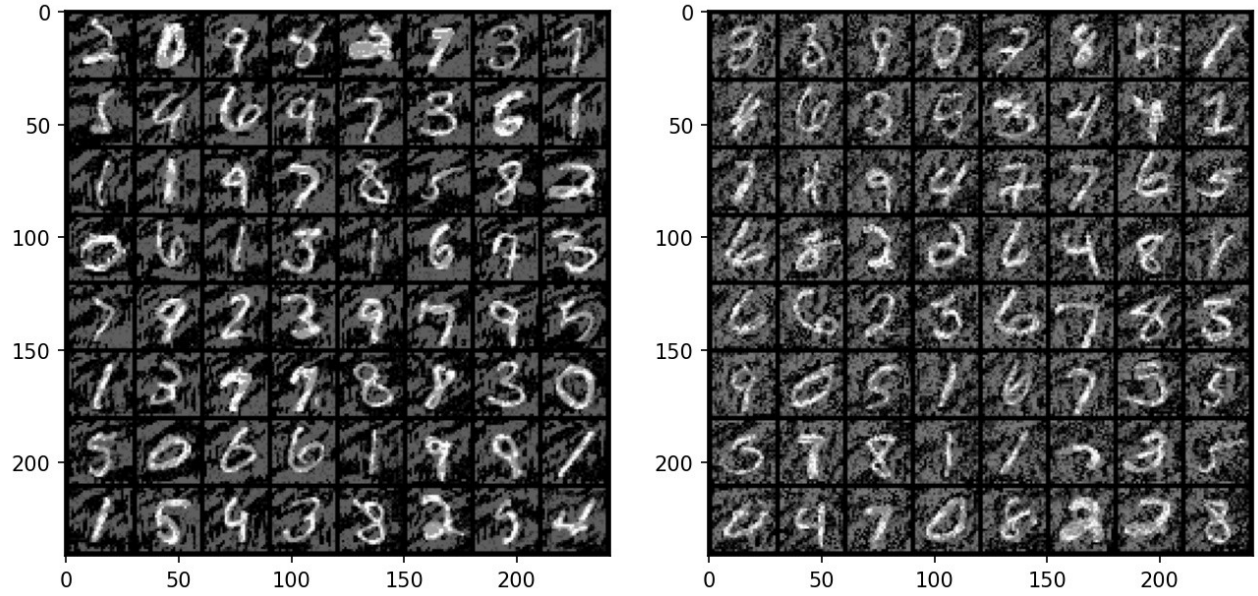


Figure 8. (a) Examples of non-targeted perturbed data using $\epsilon = 0.3$,
(b) Examples of non-targeted perturbed data using $\epsilon = 0.45$

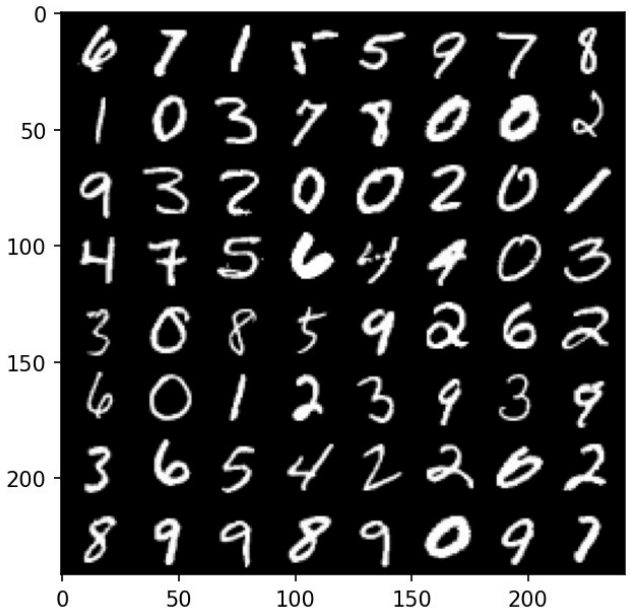


Figure 9. Examples of targeted perturbed data using $\epsilon = 0$

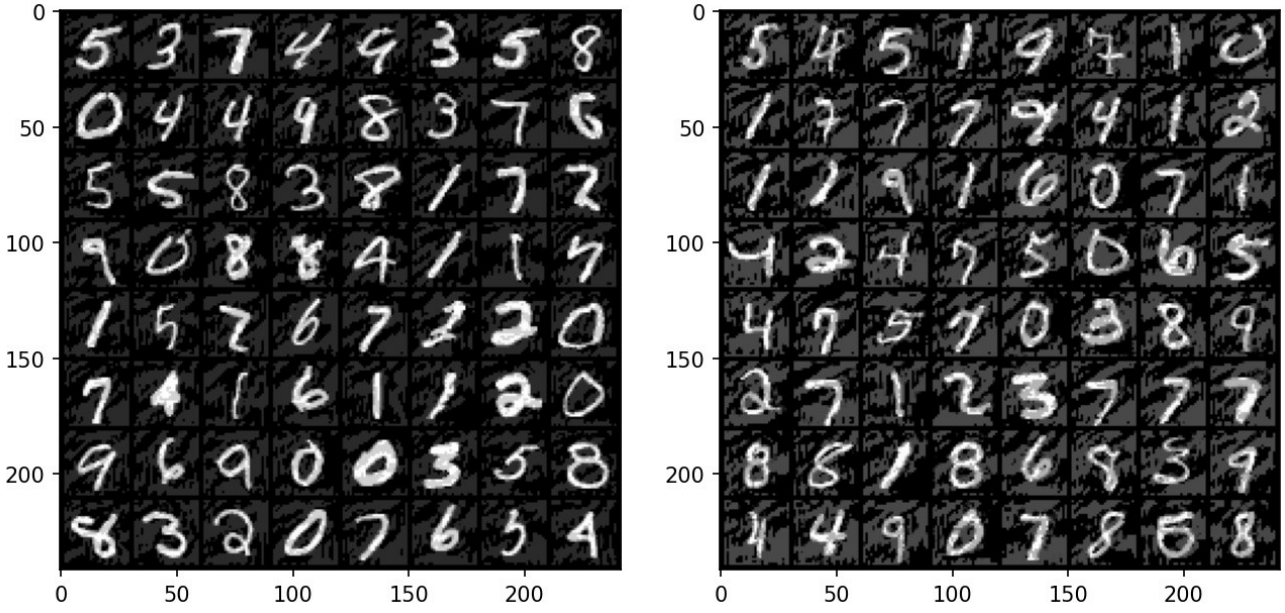


Figure 10. (a) Examples of targeted perturbed data using $\epsilon = 0.1$,
(b) Examples of targeted perturbed data using $\epsilon = 0.2$

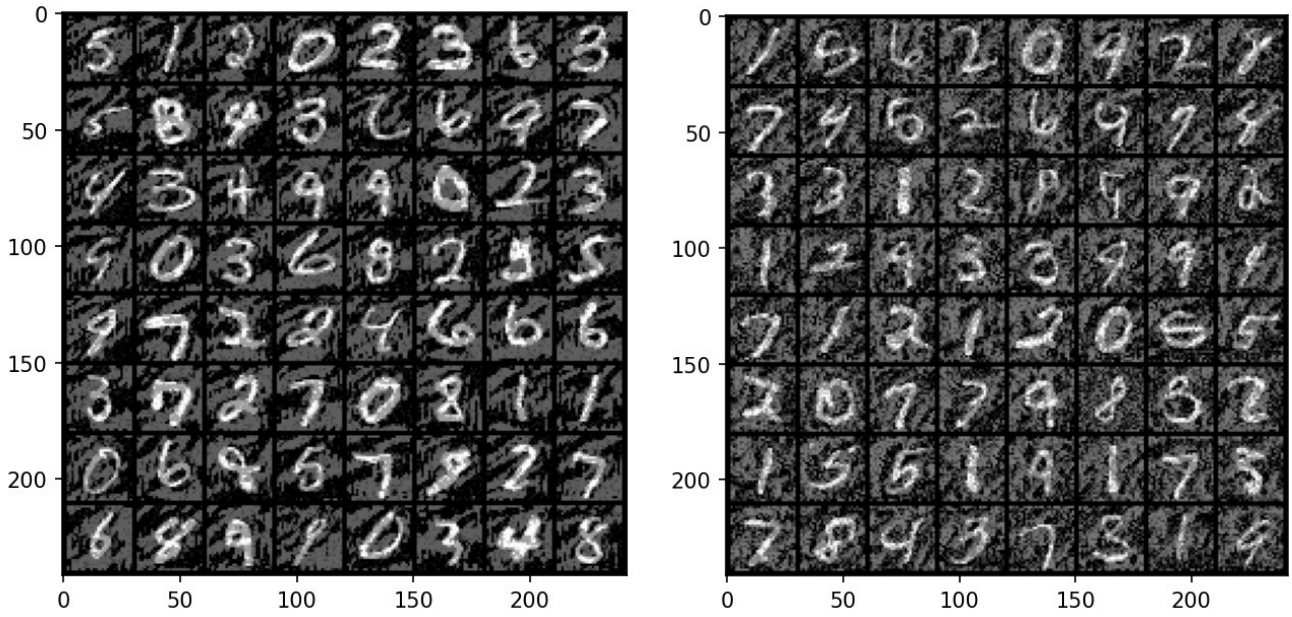


Figure 11. (a) Examples of targeted perturbed data using $\epsilon = 0.3$,
(b) Examples of targeted perturbed data using $\epsilon = 0.45$

End of report.