Assignment 03: PGD 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. 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. The modification needs to be as small, and preferably imperceptible to the human eye. The two variations of this attack are non-targeted and targeted PGD.

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 ϵ which dictates the lower and upper bound of the distribution ($-\epsilon$ is the lower bound and ϵ is the upper bound). The perturbation is then added to an input data, which will create the basis for the adversarial example. Using a loss function, and in our case cross entropy loss, the gradient of the input given the loss function and perturbed image and original label is computed. The sign of this result is taken and multiplied by another hyper-parameter α which is the step size. The resulting value is added to the perturbed image. The following equation denotes the steps mentioned above:

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

The new image is then projected towards the l infinity ball T(x) 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 and z is the perturbed image before projection. 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 add noise to move it away from the right classification in any random direction. Therefor we are optimizing the loss of on the input data by trying to augment it. The sudo code would therefore look as bellow:

```
def PGD(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 error with respect to the input, we use target labels, and minimize the error between the input and the target label. In our 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.

```
def PGD(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
```

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. When choosing a PGD attack, it is possible to

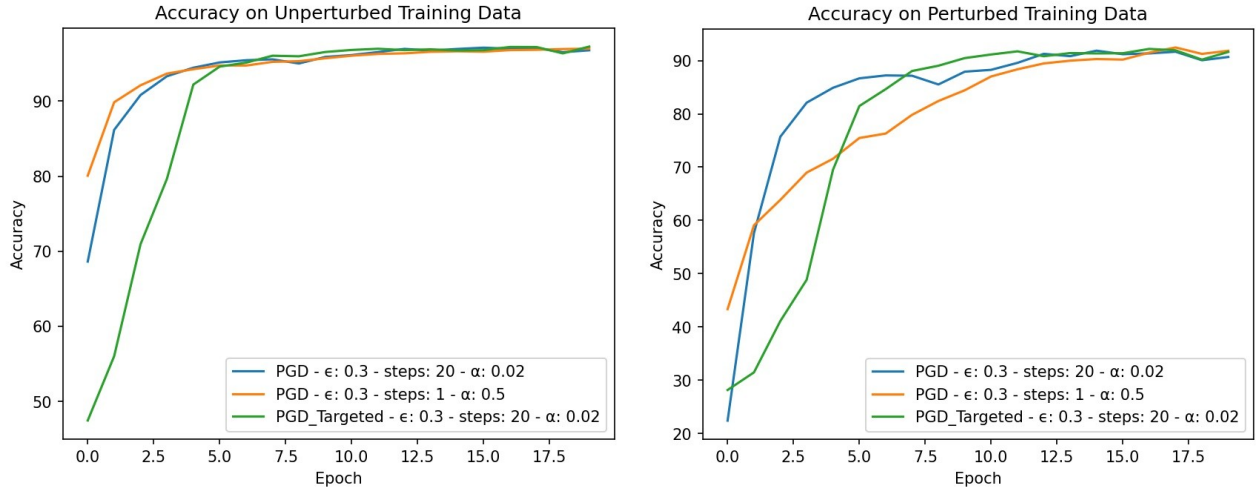


Figure 1.

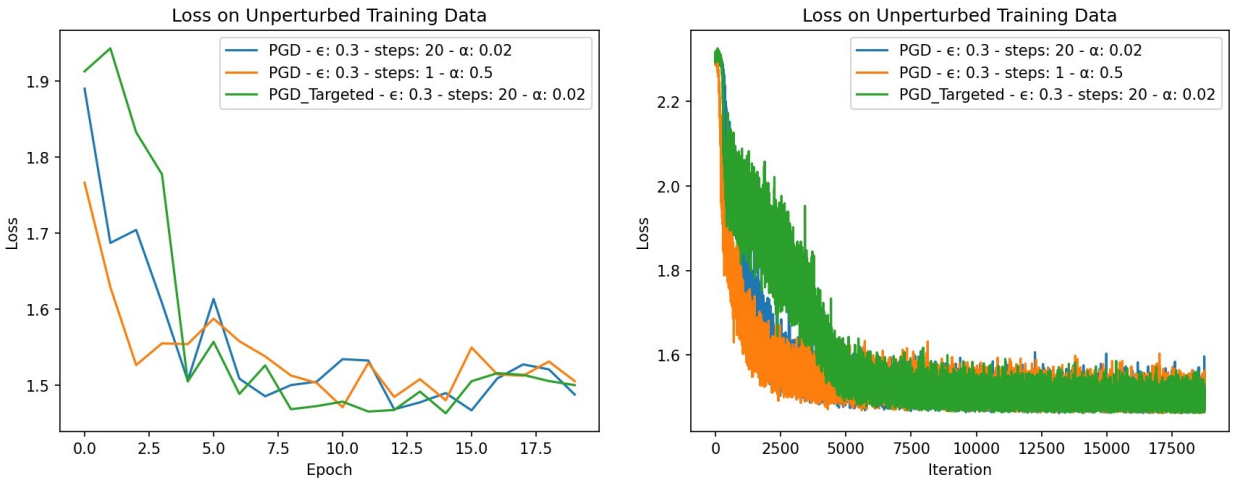


Figure 2.

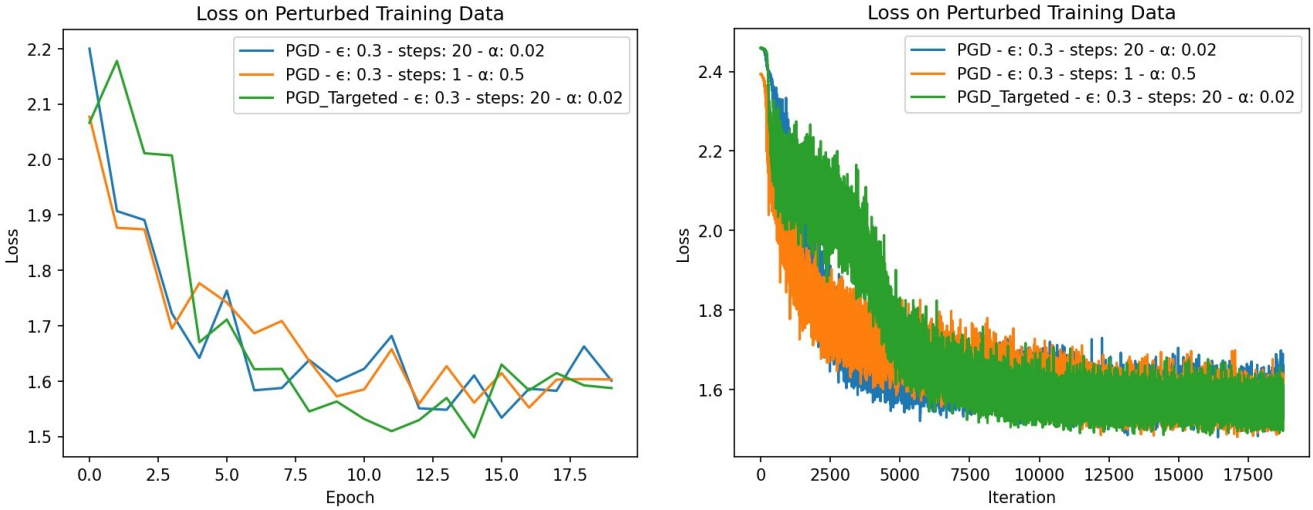


Figure 3.

Testing

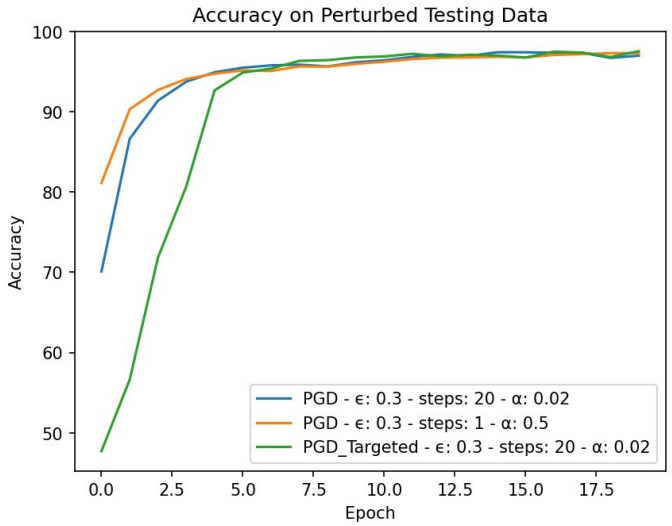


Figure 4. Model accuracy of test set while training

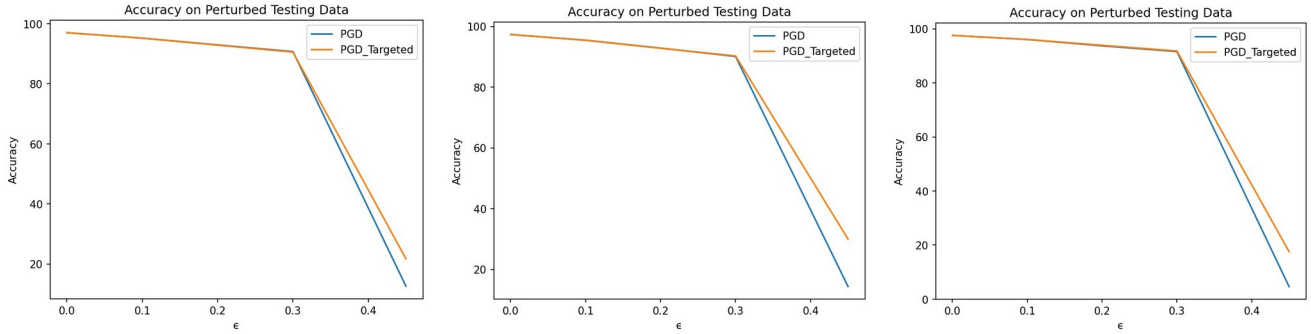
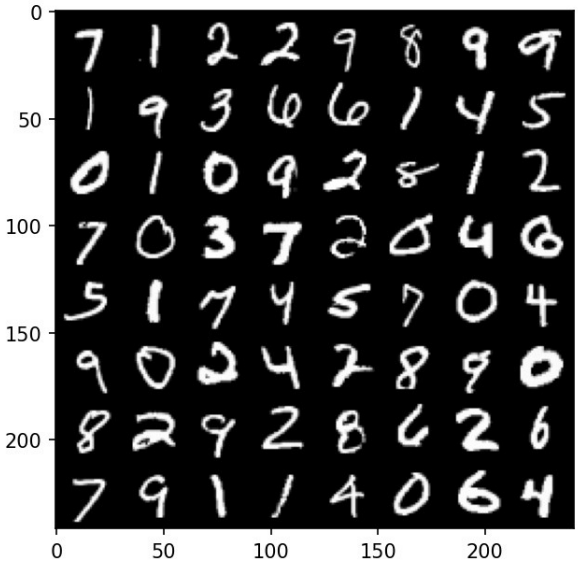
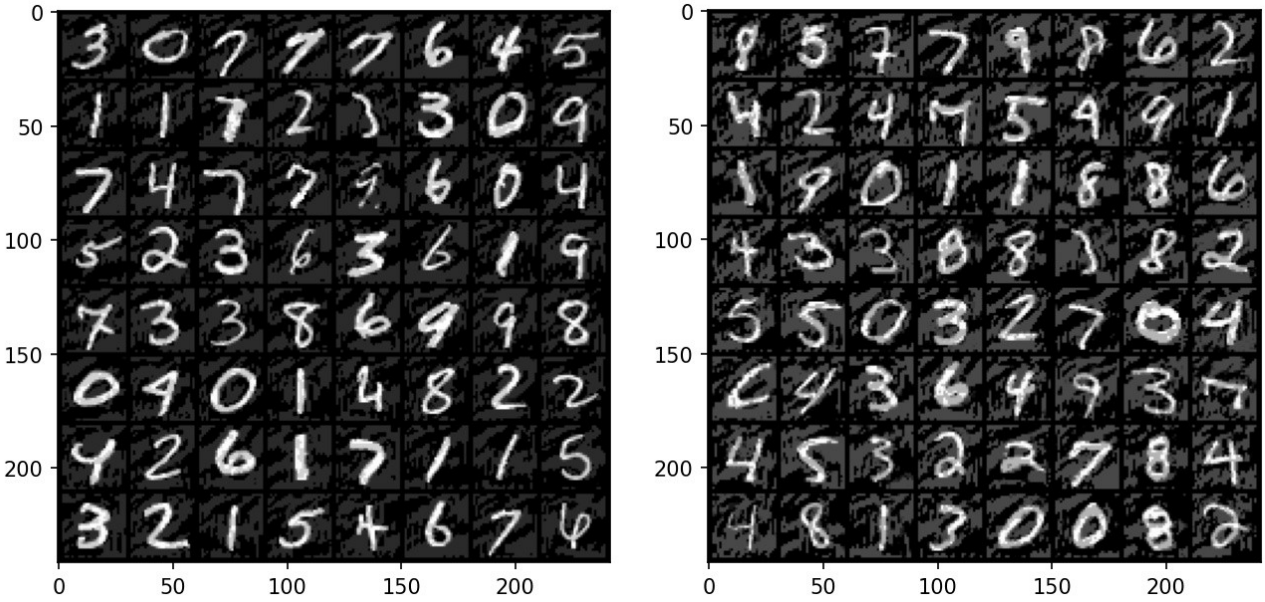


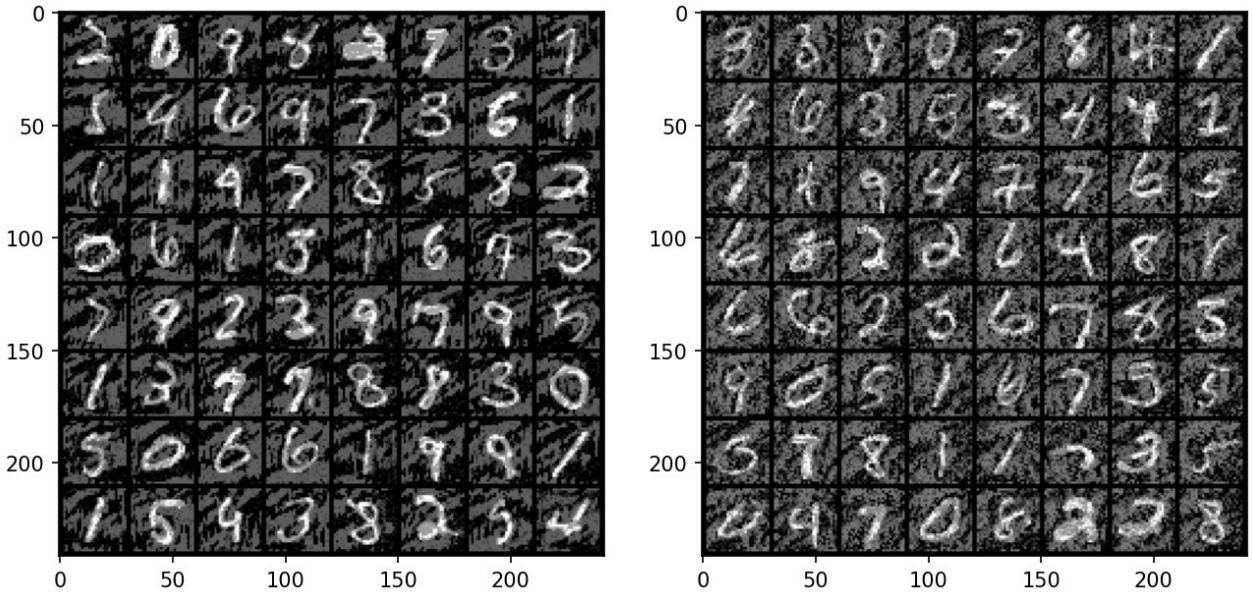
Figure 5.



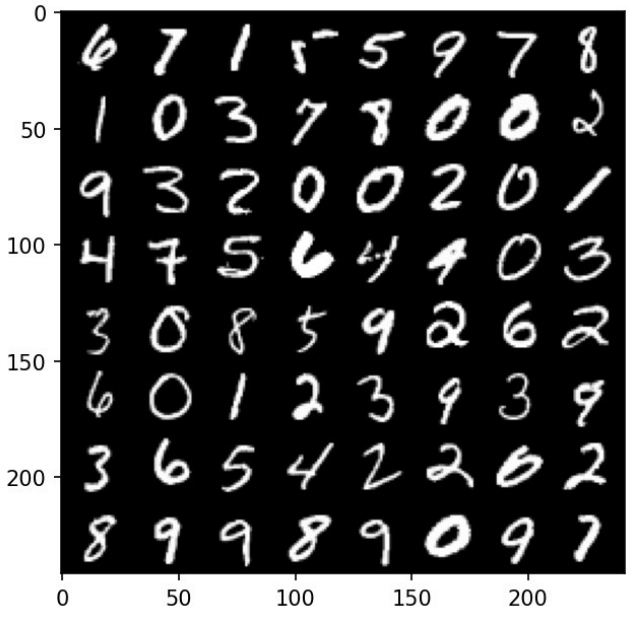
Figure



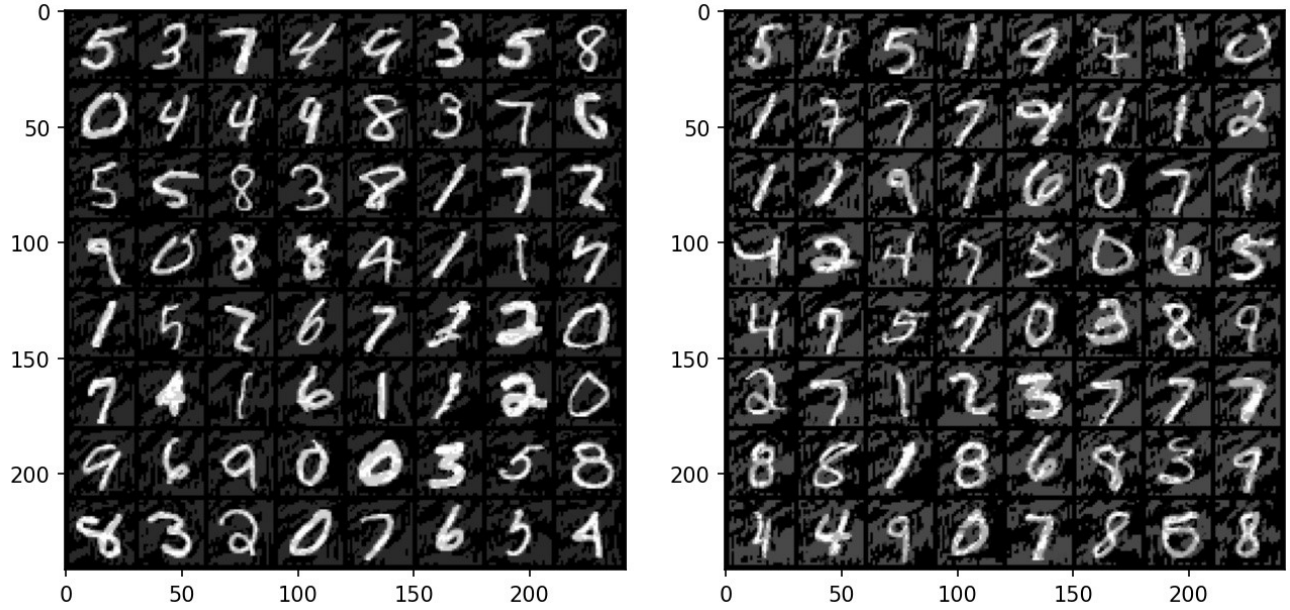
Figure



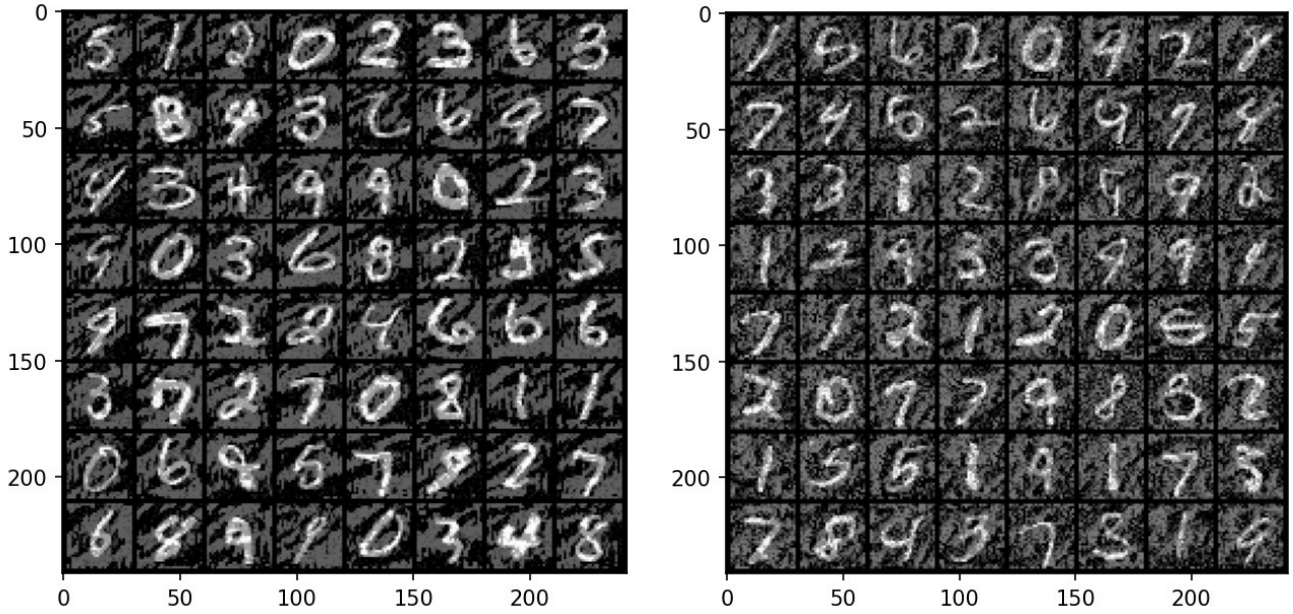
Figure



Figure



Figure



Figure