Status Report: Robust Machine Learning and Adversarial Attacks

AM 221 Advanced Optimization

Vishnu Rudrasamudram

April 3, 2019

PROGRESS

The project goals remain almost the same.

According to the tentative plan given in the proposal, below are the tasks. The status is reported below for each task

- Task: Done with literature survey and finalize the algorithms to implement
  I have gone through the popular papers and blog posts regarding the adversarial noise generation. Regarding deep reinforcement learning, I am yet to document them in the report.
- Task: Gather data and set up the environment for experiments
  I used PyTorch's existing functions to get the models and MNIST data. Environment setup for implementing Deep Reinforcement Learning algorithm (DQN) is done and yet to decide the agent to be used in OpenAI gym.
- Task: Implement few of the finalized algorithms
  I have implemented FSGM, Basic Iterative Method, and Projected Gradient Method. Started but not finished with Carlini Wagner's Attack. I will take these implementations, mostly PGD and CW methods, and incorporate in DQN algorithm to see the performance, as the analysis available for FSGM.
- Task: Prepare status report
  Done

NEXT STEPS

I will start with reviewing the current literature and get familiarize with state-of-the-art algorithms for adversarial attacks.

Below is the tentative plan for milestone 2

- Implementation of DQN using pytorch
- Use adversarial examples generation techniques to alter the input to dqn in training and execution phase
- Define performance and evaluation metrics
- Get the data and report the findings
- prepare final report and code

Below is the report on algorithms and implementation.

1 INTRODUCTION

With significant progress in a wide spectrum applications, deep learning is being employed in various applications. Some of them, like autonomous vehicles where they require to perceive the environment using cameras and lidars [2], are safety critical. The same techniques are used to train agents to play atari games and achieve super-human level performance [1]. However, a recent study has found the deep neural networks vulnerable to well-designed input samples, called adversarial examples. Adversarial examples look quite normal to a human but can easily fool deep neural networks in various stages. This problem arises mainly because of the non-convex nature of neural network optimization which makes it very difficult to understand the failure cases of these systems.

As the systems are becoming increasingly safety-critical, studying such attacks to find ways of defending against them is an extremely important task.

2 PROBLEM STATEMENT

Given a trained deep learning model f and an original input data sample $\mathbf{x}$, generating an adversarial example $\mathbf{y}$ can be formulated as a box-constrained optimization problem [3]:

$$\begin{aligned} \underset{\mathbf{y}\in\mathbb{R}^d}{\operatorname{argmin}} \; & \|\mathbf{y}-\mathbf{x}\| \\ \text{s.t. } & f(\mathbf{y}) = l', \\ & f(\mathbf{x}) = l, \\ & l \neq l', \\ & \mathbf{y} \in [0,1], \end{aligned}$$

where l and l' denote the output labels of $\mathbf{x}$ and $\mathbf{y}$, $\|\cdot\|$ denotes the distance between two data samples. Let us define $\mathbf{r} = \mathbf{y} - \mathbf{x}$, the perturbation added on $\mathbf{x}$. This optimization problem minimizes the perturbation while mis-classifying the prediction with a constraint of input data. This optimization problem formulation captures various methods used to generate adversarial examples.

Adversarial examples effects the performance of the RL agents trained with Deep Q Network. Use these adversarial examples to evaluate the performance of DQN and how adversaries affect the learning.

3 ALGORITHMS

Depending upon the objectives of the adversaries, adversarial examples can, generally, be put under two categories: mis-classification attacks and targeted attacks. To generate such adversarial, a number of algorithms have been proposed, such as the Fast Gradient Sign Method (FGSM) by Goodfellow et. al. [4], and the Jacobian Saliency Map Algorithm (JSMA) approach by Papernot et. al., [5] . Most of the algorithms assume that the details of the neural network to be attacked are available. A black-box approach to generating adversarial examples is also proposed by Papernot et. al., [6].

3.1 MIS-CLASSIFICATION ATTACKS

Misclassification attacks go for creating models that are classified incorrectly by the objective system. In this section, few algorithms used to generate such adversarial examples are presented.

3.1.1 FAST GRADIENT SIGN METHOD

Let θ be the parameters of the model, x is the input image, y is the true label for the input x, and $J(\theta, x, y)$ is the loss function used in the network. To obtain the adversarial noise, we linearize J at the current value of θ:

$$\boldsymbol{\eta} = \boldsymbol{\epsilon}\operatorname{sign}\left(\nabla_{\boldsymbol{x}} \boldsymbol{J}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})\right) \tag{3.1}$$

This method is presented in [4]. To implement this algorithm, we assume that parameters of the model will not change and we always compute the gradient with respect to the input. This will result in a matrix of the same size as that of the input. This simple method is able to get huge classification error as ϵ increases. We can see this behaviour in implementation using MNIST Dataset.

It is also observed that the adversarial examples are transferable. It is really interesting to see that the adversarial examples generated for an architecture are more often misclassified on other architectures or by models trained on a disjoint training set. When various models classify an adversarial example, they tend classify it as a same misclassified class.This can be explained using the linear explanation of adversarial examples presented in [4]. For linear models, we calculate the inner product between the weights of the network $\boldsymbol{w}$ and the adversarial example $\tilde{\boldsymbol{x}}$, and $\boldsymbol{x}$ is the unperturbed image:

$$\boldsymbol{w}^T\tilde{\boldsymbol{x}} = \boldsymbol{w}^T\boldsymbol{x} + \boldsymbol{w}^T\boldsymbol{\eta} \tag{3.2}$$

The above equations explains that the activation increases by $\boldsymbol{w}^T\boldsymbol{\eta}$. Here, the goal is to increase the activation as much as possible while capping the maximum change per pixel. This can be achieved by $\boldsymbol{\eta} = \text{sign}(\boldsymbol{w})$. This shows that a simple linear model can have adversarial examples if its input has sufficient dimensionality.

The property the we discussed in the previous paragraph, **transferability** implies that we need not have the parameters for a model to generate the adversarial examples. Such type of attacks are known as black box attacks. These type of attacks worsen the security even further of the deep neural networks.

3.1.2 Basic Iterative Method

The Basic Iterative Method [11] is a sort of extension to the Fast Gradient Sign Method where, instead of taking one single step in the direction of gradient, we take multiple small steps of magnitude α.

$$X_0^{adv} = X \tag{3.3}$$

$$X_{N+1}^{adv} = Clip_{X,\epsilon}\left\{X_N^{adv} + \alpha\,\text{sign}\left(\nabla_X J\left(X_N^{adv}, y_{true}\right)\right)\right\} \tag{3.4}$$

After every iteration, the image pixel values are clipped to ensure they are in the range used in the original image. The number of iterations are taken to be $min(\epsilon + 4, 1.25\epsilon)$. This number is chosen heuristically; it is sufficient for the adversarial example to reach the edge of the max-norm ball but restricted enough to keep the computational cost tractable for experimenting.

3.2 Targeted Attacks

Targeted attacks aim for creating adversarial examples that are classified as a specific class as designed by the attacker. The algorithms that we are going to discuss facilitates the targeted classification of the data.

3.2.1 Iterative Least Likely Class Method

This method does not exactly classify the data point to any target. This attack is best suitable for the datasets with large number of classes. This method tries to classify the original image to the class with the least probability according to prediction of original image. This class is found by:

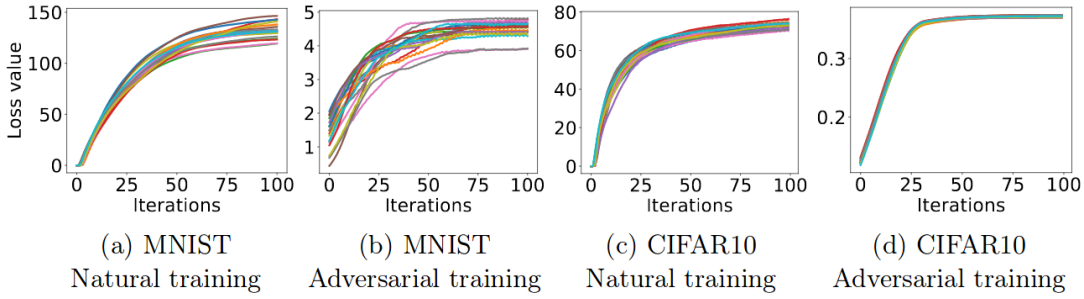
$$y_{LL} = \underset{y}{\text{argmin}}\{p(y|\boldsymbol{X})\} \tag{3.5}$$

In order to get this misclassification, we try to maximize $\log p(y_{LL}|\boldsymbol{X})$ by making iterative steps in the direction of $\text{sign}\left\{\nabla_X \log p\left(y_{LL}|\boldsymbol{X}\right)\right\}$.

$$X_0^{adv} = X \tag{3.6}$$

$$X_{N+1}^{adv} = Clip_{X,\epsilon}\left\{X_N^{adv} - \alpha\,\text{sign}\left(\nabla_X J\left(X_N^{adv}, y_{LL}\right)\right)\right\} \tag{3.7}$$

In this method, we use the same number iterations and value of alpha as we did in Basic Iterative Method.



(a) MNIST Natural training (b) MNIST Adversarial training (c) CIFAR10 Natural training (d) CIFAR10 Adversarial training

Figure 3.1: Cross-entropy loss values while creating an adversarial example from the MNIST and CIFAR10 evaluation datasets [12]

3.2.2 Projected Gradient Descent Attack

In [12], Madry et al. provided a way to understand adversarial robustness of classifiers through the eyes of optimization. A natural saddle point formulation is used to study the security against adversarial attacks. Since the existing defence mechanisms guard against only a specific type of attacks, this formulation is important. This formulation helps to bind the attacks and defenses into a single framework. Here, the different attacks correspond to solving the constrained optimization problem.

As mentioned in the previous paragraph, this can be viewed through optimization perspective. Let us consider the training data distribution $\boldsymbol{D}$ with training samples as (x, y), where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{y} \in [k]$ are the corresponding labels. Let $L(\theta, \mathbf{x}, \mathbf{y})$ be the loss function and θ are the weights of the network. As with all classifiers, we need to find θ which minimizes the risk $\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim \boldsymbol{D}}[\boldsymbol{L}(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y})]$.

To accomplish this we would want to perform the standard Empirical Risk Minimization(ERM). But, ERM does not deliver adversarially robust models, so we will attempt to change our problem to solve. To start with, since we need to catch the general thought of attacks instead of attempt to safeguard against individual attacks. Along these lines, we characterize an "Attack Model" ,i.e an exact notion of the attacks our models ought to be kept safe against. For each datapoint $\mathbf{x}$, we select a permitted noise $S \subset \mathbb{R}^d$ to allow a limit of power of the adversary. This is computed on the basis of l_∞ ball around $\mathbf{x}$. Now we define the risk using the notion of our adversary. We include the noise directly into the loss function which leads to a saddle point problem:

$$\min_{\theta} \rho(\theta), \text{ where } \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim D}\left[\max_{\delta \in S} L(\theta, x+\delta, y)\right] \tag{3.8}$$

In the above equation, the maximization corresponds to an adversary trying to achieve high loss, and the minimization signifies that the adversarial noise should be minimum.

If we focus on maximizing the data point's loss, the problem is highly non-concave and does not look tractable. To solve this problem, one way would be to approximate the objective function, for example with Taylor Series and use FGSM. It is a first order expansion of this inner maximization problem and it gives fairly good adversarial examples. Also, we get the advantage of quick computation. But, this is definitely not the best available method to do so.

A stronger variant of this attack would be to use Projected Gradient Descent (PGD) on the negative loss function [11]

$$x^{t+1} = \Pi_{x+S}\left(x^t + \alpha \operatorname{sgn}\left(\nabla_x L(\theta, x, y)\right)\right) \tag{3.9}$$

In [12] it is stated that the maximization problem was tractable (at least from the point of view of first order methods). They showed this by running PGD from various initial oints within the l_∞ norm ball of the data points. And they found that the loss for the point increased with the number of iterations and plateaued quickly. Almost, the same values are observed when starting from the remaning initial points. This can be found in 3.1. *While there are many local maxima spread apart from one another within* $\mathbf{x}+S$*, they have very well-concentrated loss values.*

The experiments [12] found that local maxima have similar loss values, so if the network is robust against PGD adversaries, it must be robust against other adversaries as well, for example first order adversaries. With this reasoning, they concluded that robustness against PGD adversaries would more or less ensure robustness against other first order adversaries.

3.2.3 Carlini Wagner's L2 Attack

Carlini and Wagner [13] proposed a set of attacks based on L_0, L_2, L_∞ metric which broke a defensive distilled model [14], which is known for its robustness against previous attacks.

The problem formulation is same as the formulation given in section 2, except that the objective can be any of L_0, L_2, L_∞ metrics.

Need to include more details

3.3 Reinforcement Learning and Neural Networks [1]

Reinforcement Learning problems are generally formulated as Markov Decision Process and there are Monte Carlo and Temporal-Difference methods we can choose from, to solve the problem. Q-Learning is one of the TD methods which is to estimate the optimal value of each action, defined as the expected sum of future rewards when taking that action and following the optimal policy thereafter. The value of an action a in a state s is given by the action-value function Q defined as:

$$Q(s,a) = R(s,a) + \gamma \max_{a'} \left(Q\left(s', a'\right) \right) \tag{3.10}$$

The above function is referred to as a Q-table. This is used to approximate the action-value function q_π for the policy π.

One drawback of Q-Learning is that the use of the Q-table assumes we are using only discrete states and actions. This is assumption is significant since most of the physical environments are continuous. Moreover, the states can also be continuous.

To address this issue, the use of function approximation created a major breakthrough, since this is exactly the purpose of neural networks to approximate such functions. However, Deep Q Learning is found to be highly unstable when neural networks are used to represent the action values. This is addresses in [1] using

- Experienced Replay
- Fixed Q-Targets

Experience Replay

When the agent interacts with the environment, the sequence of the experiences tuples can be highly correlated. In order to satisfy the assumption that the data is independent and indentically distributed, replay buffer is used. By using this, we can keep track of states that we visited and using experience replay to sample from the buffer at random, preventing action values from oscillating or diverging catastrophically.

Fixed Q-Targets

In Q-Learning, when we want to compute the TD error, we compute the difference between the TD target and the current predicted Q-value (estimate of Q). But we do not have any idea of the read TD target and this is the reason why we estimate it. However, the problem is that we are using the same parameters (weights) for estimating the target and the Q-value. As a consequence, there is a big correlation between TD target and the parameters we are changing.

Therefore, it means that at every step of training, out Q-values shift but so do the target values. So we are getting closer to out target but the target is also moving. This results in divergence or oscillations in the tarining phase.

To remove these correlations with the target, we use a separate target network that has the same architecture as the DQN. Thae change lies in the fact that we are updating the target Q-Network's weights less often than the primary Q-Network.

4 Data and Experiment Setup

I have used pytorch to implement FGSM, Basic Iterative Method, and Projected Gradient Descent on MNIST [7]. I have setup the OpenAI gym environment and run some tutorials. Now I need to figure out how to bridge adversarial generation with DQN.

I have tested how the performance of the network (accuracy) changed by changing parameter values to generate adversarial examples. This can be seen in 4.1. After generating the adversaries, I have tested it with few of the data points taken from the dataset: FGSM Testing 4.2, BIM Testing 4.3, PGD Testing 4.4

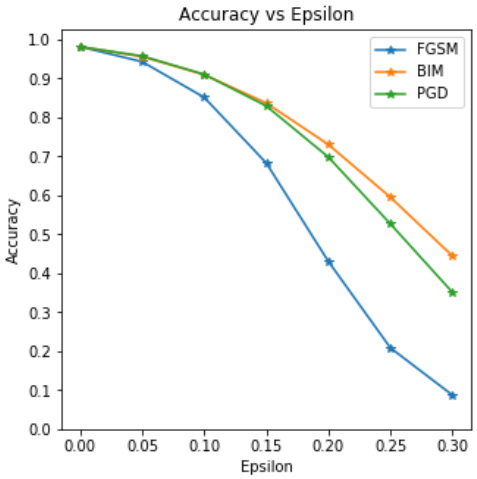


Figure 4.1: Accuracy vs ϵ

Figure 4.2: Testing with FGSM Attack

REFERENCES

[1] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves et al. "Human-level control through deep reinforcement learning." Nature 518, no. 7540 (2015): 529.

[2] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).

[3] Yuan, Xiaoyong, et al. "Adversarial examples: Attacks and defenses for deep learning." IEEE transactions on neural networks and learning systems (201

[4] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572 (2014).

[5] Papernot, Nicolas, et al. "The limitations of deep learning in adversarial settings." 2016 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2016.

[6] Papernot, Nicolas, et al. "Practical black-box attacks against machine learning." Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ACM, 2017.

[7] MNIST Dataset `http://yann.lecun.com/exdb/mnist/`

[8] CIFAR-10 Dataset `https://www.cs.toronto.edu/~kriz/cifar.html`

[9] PyTorch `pytorch.org`

[10] OpenAI Gym `gym.openai.com`

[11] Kurakin, Alexey, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world." arXiv preprint arXiv:1607.02533 (2016).

[12] Madry, Aleksander, et al. "Towards deep learning models resistant to adversarial attacks." arXiv preprint arXiv:1706.06083 (2017).

[13] Carlini, Nicholas, and David Wagner. "Towards evaluating the robustness of neural networks." 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017.

[14] Papernot, Nicolas, et al. "Distillation as a defense to adversarial perturbations against deep neural networks." 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016.

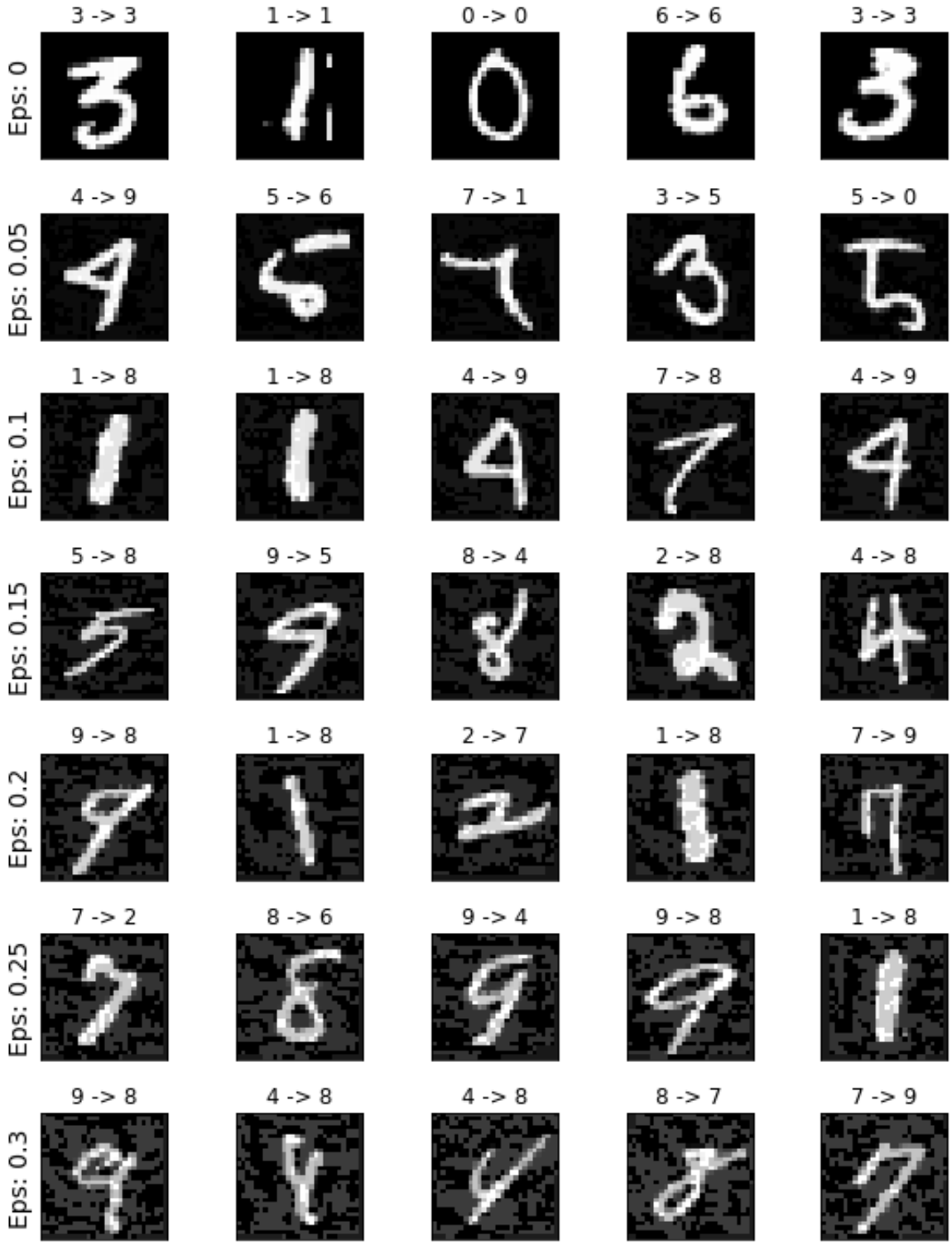


Figure 4.3: Testing with BIM Attack

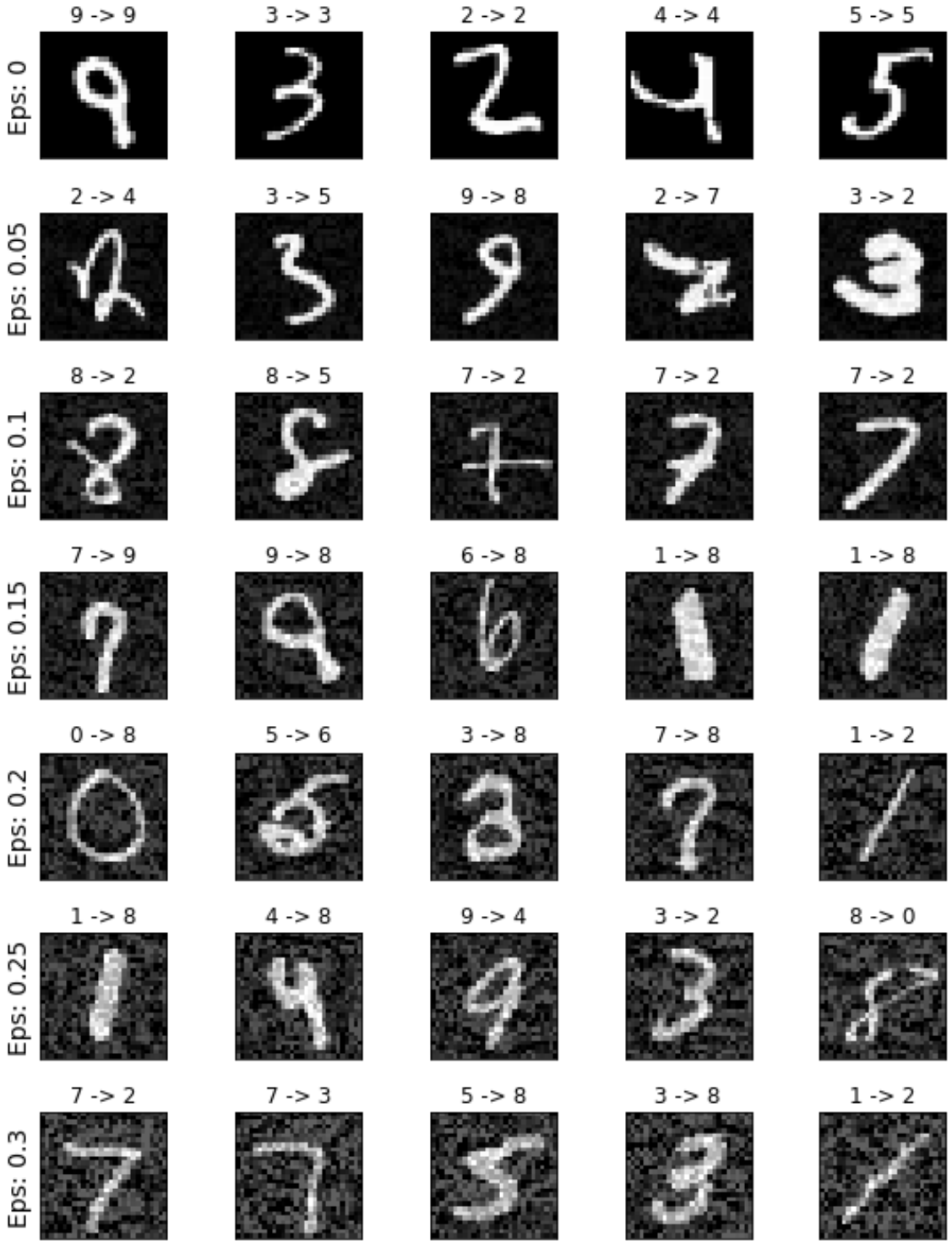


Figure 4.4: Testing with PGD Attack