Introduction to Reinforcement Learning & Deep Learning

CSI 5340

Robust Models: *Using Projected Gradient Descent Attack for CNN Model Training on MNIST*

Assignment 3

Authors
Alexandra Sklokin (300010511)

Professor
Runzhi Tian

November 8, 2022

Contents

1 Introduction **2**

2 MNIST Dataset **2**

3 PGD Attack **2**

3.1 Untargeted 2

3.2 Targeted 3

3.3 Training using PGD Attack 4

4 Methodology **4**

4.1 CNN Model 4

4.2 Classifier Parameterization 5

5 Experimental Results **6**

5.1 Adversarial Examples 6

5.2 Training and Testing Performance 8

6 Discussion **9**

6.1 Adversarial Examples 9

6.2 Training Performance 9

6.3 Testing Performance 10

7 Conclusion **11**

1 Introduction

In this report I will discuss the implementation of Projected Gradient Descent (PGD) for the purpose of training a robust Convolutional Neural Network (CNN). Both targeted and untargeted PGD attack is used to create adversarial examples, by perturbing the data such that it is incorrectly classified by our CNN model. Yet these perturbations can be imperceptible for the human eye.

2 MNIST Dataset

The Modified National Institute of Standards and Technology (MNIST) dataset used in this assignment contains handwritten digits, and can be used to train neural networks for the purpose of image recognition. This dataset contained 60,000 training samples, and 10,000 testing samples.

3 PGD Attack

Projected Gradient Descent is a possible solution to the constrained optimization problem which arises when creating adversarial examples. Adversarial examples are those which can fool our model into a wrong classification. Adversarial training uses these perturbed samples during training, to increase model robustness against attack. This is a white-box method, since it requires model gradients to make perturbations.

There are three hyper-parameters to consider when creating adversarial examples using PGD:

- α - the learning rate or step size. If α is small, then the PGD attack will hardly perturb the original image. We would like to choose *alpha* sufficiently large to converge to an adversarial example, but not too large as to skip the optimal solution.
- ϵ - the perturbation constraint which identifies the l^∞ ball's size. This ball represents the perturbation set generated around one original image sample. Larger ϵ allows for more variance in perturbation.
- *steps* - number of repetitions of the PGD attack algorithm. As *steps* increases, the image perturbation will become more apparent to the human eye.

3.1 Untargeted

Untargeted PGD attack allows for an adversarial example to be wrongfully classified by our model, without specifying the target 'wrong' prediction.

Algorithm 1 Untargeted PGD Attack ($model$, $original\_data$, ϵ, α, $steps$)

$images \leftarrow original\_data.images$
$true\_labels \leftarrow original\_data.labels$
for $j = 1$ to $steps$ **do**
  # CNN Model Predictions
  $predicted\_labels \leftarrow model.predict(images)$
  # Compute Gradients
  $gradients \leftarrow 0$
  $loss \leftarrow Cross\_Entropy\_Loss(predicted\_labels, true\_labels)$
  $loss.backward\_propagation()$
  # Update Images
  $adversarial\_images \leftarrow images + \alpha * images.gradients.sign()$
  $\eta \leftarrow clamp(adversarial\_images - original\_images, min = -\epsilon, max = \epsilon)$
  $images \leftarrow clamp(original\_images + \eta, min = 0, max = 1)$
end for
return $(images, true\_labels)$

3.2 Targeted

Targeted PGD attack specifies the target prediction we would like to achieve when the model is fed an adversarial example.

Algorithm 2 Targeted PGD Attack ($model$, $original\_data$, ϵ, α, $steps$)

$images \leftarrow original\_data.images$
$true\_labels \leftarrow original\_data.labels$
for $j = 1$ to $steps$ **do**
  # CNN Model Predictions
  $predicted\_labels \leftarrow model.predict(images)$
  # Compute Gradients
  $gradients \leftarrow 0$
  $loss \leftarrow Cross\_Entropy\_Loss(predicted\_labels, true\_labels)$
  $gradients \leftarrow loss.backward\_propagation()$
  # Update Images
  $adversarial\_images \leftarrow images + \alpha * images.gradients.sign()$
  $\eta \leftarrow clamp(adversarial\_images - original\_images, min = -\epsilon, max = \epsilon)$
  $images \leftarrow clamp(original\_images + \eta, min = 0, max = 1)$
end for
# Targeted Label
$target\_labels \leftarrow model.predict(images)$
$target\_labels \leftarrow target\_labels.exclude(true\_labels)$
return $(images, target\_labels)$

3.3 Training using PGD Attack

The PGD adversarial examples obtained from Algorithms 1 and 2 are used to train robust models. Rather than training the classifier on the original data, we use the perturbed images and labels.

Algorithm 3 PGD Training (*model*, *training_data*, *pgdAttack*, *epochs*, ϵ, α, *steps*)

for $j = 1$ to *epochs* **do**
  for *batch* in *training_data* **do**
    # Use Algorithm 1 or 2
    $(pgd\_images, pgd\_labels) = pgdAttack(model, batch, \epsilon, \alpha, steps)$
    # Train Model
    $predicted\_labels \leftarrow model.predict(pgd\_images)$
    $loss \leftarrow Cross\_Entropy\_Loss(predicted\_labels, pgd\_labels)$
    $gradients \leftarrow 0$
    $gradients \leftarrow loss.backward\_propagation()$
    $model.step(gradients)$
  end for
end for

4 Methodology

For this assignment I used *jupyter notebooks* and Excel spreadsheets. Please find all of the code for this assignment in my Git repository[1].

CNN_PGD.py contains all of the CNN and PGD implementation code, including model training, testing, targeted PGD attack, and untargeted PGD attack. *Experiment.ipynb* is the notebook for viewing experimentation results.

4.1 CNN Model

I designed a CNN model using the *pytorch.nn* library, which allows for backward propagation, gradient calculations, model evaluation, etc. This library also allows for the customization of training and testing functions, as needed by Algorithm 3. The model architecture is described in Figure 1.

[1] https://github.com/alexandrasklokin/CSI5340/tree/main/CSI5340_A3

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 8, 28, 28]             208
              ReLU-2           [-1, 8, 28, 28]               0
         MaxPool2d-3           [-1, 8, 14, 14]               0
            Conv2d-4          [-1, 16, 14, 14]           3,216
              ReLU-5          [-1, 16, 14, 14]               0
         MaxPool2d-6           [-1, 16, 7, 7]               0
            Linear-7                [-1, 150]         117,750
              ReLU-8                [-1, 150]               0
            Linear-9                [-1, 150]          22,650
             ReLU-10                [-1, 150]               0
           Linear-11                 [-1, 10]           1,510
================================================================
Total params: 145,334
Trainable params: 145,334
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.17
Params size (MB): 0.55
Estimated Total Size (MB): 0.72
----------------------------------------------------------------
```

Figure 1: CNN Model Architecture Summary

4.2 Classifier Parameterization

For the purposes of this assignment, I trained 4 CNN classifiers.

The *Baseline* model should be used as a benchmark for the best possible performance of the CNN model, on the unperturbed MNIST dataset.

The next three classifiers are parameterized in the assignment requirements. These models will likely have lower performance, as the images will be highly perturbed due, depending on the *step* and α values.

1. **Baseline**
   - No PGD attack.
   - ** Used as a baseline for the performance of CNN model on MNIST digit classification problem. Expected around 99% accuracy.*
2. **Classifier_One**
   - α - 0.02; ϵ - 0.3; *steps* - 20
   - Untargeted PGD Algorithm 1
3. **Classifier_Two**
   - α - 0.5; ϵ - 0.3; *steps* - 1

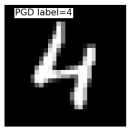
- Untargeted PGD Algorithm 1

4. **Classifier_Three**
   - α - 0.02; ϵ - 0.3; *steps* - 20
   - Targeted PGD Algorithm 2

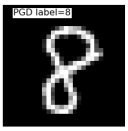
5 Experimental Results

5.1 Adversarial Examples

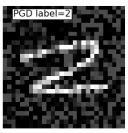
(a) Original 2



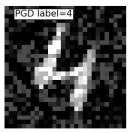
(b) Original 4



(c) Original 8



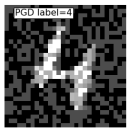
(d) Classifier One 2



(e) Classifier One 4

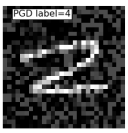
(f) Classifier One 8

(g) Classifier Two 2

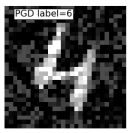


(h) Classifier Two 4

(i) Classifier Two 8



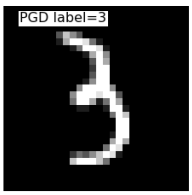
(j) Classifier Three 2



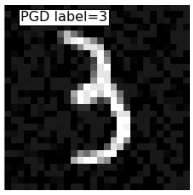
(k) Classifier Three 4

(l) Classifier Three 8

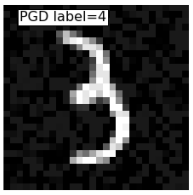
Figure 2: Adversarial Images Produced when Training the CNN Models.



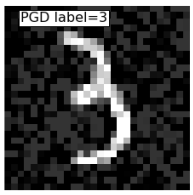
(a) $\epsilon = 0$



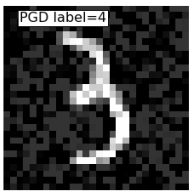
(b) UntargetedPGD $\epsilon = 0.1$



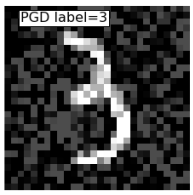
(c) TargetedPGD $\epsilon = 0.1$



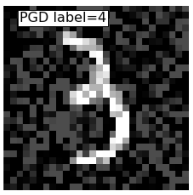
(d) UntargetedPGD $\epsilon = 0.2$



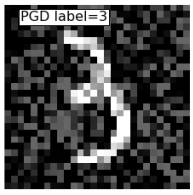
(e) TargetedPGD $\epsilon = 0.2$



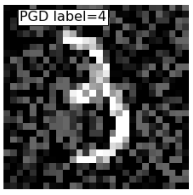
(f) UntargetedPGD $\epsilon = 0.3$



(g) TargetedPGD $\epsilon = 0.3$



(h) UntargetedPGD $\epsilon = 0.45$



(i) TargetedPGD $\epsilon = 0.45$

Figure 3: Adversarial Images during Model Testing, for Different Values of PGD Attack ϵ.

5.2 Training and Testing Performance

		BASELINE		CLASSIFIER ONE				CLASSIFIER TWO				CLASSIFIER THREE			
PGD Train Parameters				α - 0.02	ε - 0.3	steps - 20	targetedPGD	α - 0.5	ε - 0.3	steps - 1	tagetedPGD	α - 0.02	ε - 0.3	steps - 20	untargetedPGD
	Epoch	TrainLoss	TrainAccuracy	TrainLoss	TrainAcc	PerturbedLoss	PerturbedAcc	TrainLoss	TrainAcc	PerturbedLoss	PerturbedAcc	TrainLoss	TrainAcc	PerturbedLoss	PerturbedAcc
TRAINING	0	2.3095	0.10	2.3026	0.09	2.3026	0.09	2.1286	0.04	2.1578	0.04	2.3105	0.03	2.3105	0.08
	1	0.0648	0.97	0.7997	0.79	1.2389	0.57	0.5765	0.96	0.9371	0.51	0.7608	0.74	1.7630	0.45
	2	0.0692	0.98	0.3011	0.93	0.6683	0.78	0.4522	0.8	0.8265	0.56	0.7526	0.75	1.1842	0.56
	3	0.1016	0.96	0.2207	0.93	0.5286	0.84	0.4440	0.8	0.6543	0.57	0.7430	0.75	0.9207	0.72
	4	0.0989	0.97	0.1765	0.94	0.4057	0.86	0.4416	0.82	0.6110	0.65	0.6929	0.76	0.5140	0.83
	5	0.0146	1.00	0.1431	0.99	0.3866	0.89	0.4637	0.82	0.5776	0.67	0.6927	0.77	0.4298	0.86
	6	0.0528	0.99	0.1884	0.95	0.2729	0.92	0.5113	0.81	0.5749	0.69	0.6955	0.79	0.3550	0.87
	7	0.0125	1.00	0.1331	0.96	0.1946	0.94	0.5619	0.76	0.4540	0.83	0.7234	0.76	0.3099	0.86
	8	0.0179	0.99	0.0752	0.99	0.1683	0.94	0.5400	0.76	0.4405	0.85	0.7346	0.75	0.1907	0.91
	9	0.0324	0.98	0.1160	0.98	0.1300	0.98	0.5979	0.7	0.4033	0.86	0.7467	0.75	0.1861	0.92
	10	0.0123	1.00	0.1109	0.98	0.12	0.98	0.6123	0.66	0.4194	0.87	0.7583	0.74	0.1795	0.93
	Epsilon	Original_Test	Untargeted	Targeted	Original_Test	Untargeted	Targeted	Original_Test	Untargeted	Targeted		Original_Test	Untargeted	Targeted	
TESTING	0		0.97			0.95			0.85				0.9		
	0.1		0.34	0.52		0.94	0.89		0.86	0.84			0.91	0.92	
	0.2	0.97	0.00	0.00	0.95	0.90	0.83	0.85	0.84	0.75		0.9	0.86	0.89	
	0.3		0.00	0.00		0.84	0.76		0.80	0.67			0.80	0.88	
	0.45		0.00	0.00		0.27	0.06		0.21	0.10			0.15	0.25	

Table 1: Summary of Loss and Accuracy Obtained during Training and Testing of the CNN Models. Evaluated on Original and Perturbed Data.

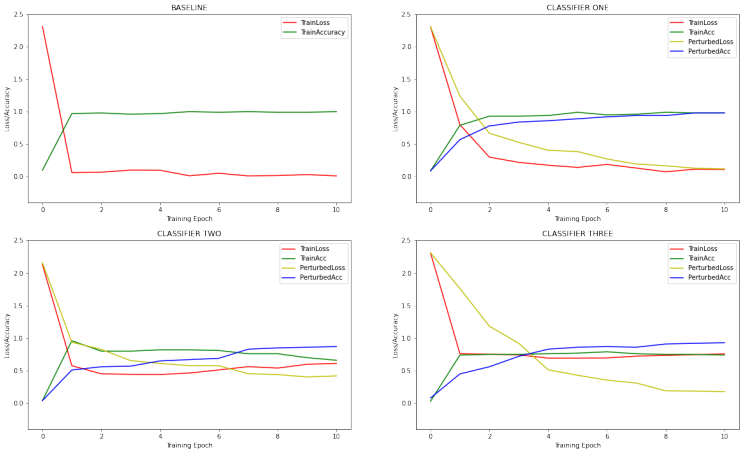


Figure 4: Training Loss and Accuracy Curves.

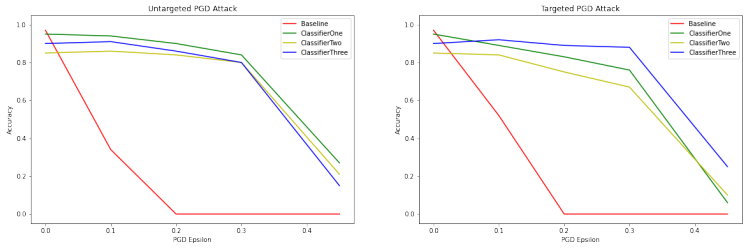


Figure 5: Testing Accuracy Curves for Different Values of PGD Attack ϵ.

6 Discussion

6.1 Adversarial Examples

Figures [2, 3] show different perturbed images produced from the untargeted and targeted PGD attack algorithms. Clearly, my PGD attack implementation is able to produce images which trick our CNN model into making incorrect predictions, by inducing small changes to the pixels' colours.

From Figure [2] we can see examples of perturbed images created during the model training phase of this experiment. *Classifier_One* and *Classifier_Three* have a large number of *steps* and small learning rate α, meaning the PGD algorithm is more likely to converge to adversarial examples over time. By contrast, *Classifier_Two* only has one *steps* and a large learning rate α, so this is not likely to reach a global maxima. Usually, we would prefer a large number of *steps* and small α, to guarantee convergence, and thus to train a more robust model.

From Figure [3] we can observe the effect of the ϵ parameter on the perturbation of the images. These images were produced during the testing phase of this experiment. Clearly, as ϵ increases, the amount of perturbation to the images increases, and the changes to the images become more human recognizable. I expect that the CNN models' accuracy will decrease as the images become more perturbed, particularly since our training ϵ was small.

6.2 Training Performance

I will call my hypothesis the Accuracy-Robustness Trade-Off, whereby I predict that increased model robustness must come at the cost of model performance/accuracy. I would like to choose a best model which remains accurate on both the original training (and testing) data, and the perturbed data.

Table [1] shows the training loss and accuracy over the *epochs* of our 4 classifiers. As expected, the *Baseline* model has the highest training accuracy, since it does not have to compromise for model robustness against adversarial attack.

However, this model is absolutely not robust, so we will not chose it. The best model from this experiment is $Classifier\_One$, followed by $Classifier\_Three$ and finally $Classifier\_Two$.

On the original training data, $Classifier\_One$ performs almost as well as the *Baseline* model, only 2% lower accuracy. We can say that this model has compromised the least accuracy for robustness on perturbed data. This model also has the highest accuracy on the perturbed training data. In Figure [4], this first classifier has a smoothly decreasing training loss for both the original and perturbed data. This model speaks to the success of my implementation of untargeted PGD attack.

In comparison, $Classifier\_Three$ uses targeted PGD attack, with the same parameter values. This classifier performed well on the perturbed data, but not very well on the original data. This is likely because the targeted PGD attack optimization problem is a more constrained optimization problem.

Clearly, $Classifier\_Two$ does not perform as well as the previously mentioned models since the learning rate is too large. This is also apparent in Figure [4], where we can observe that the training loss and accuracy 'wobbles' up and down throughout model training. Although this model performs well on the perturbed training data, it does not produce accurate predictions for the original training data, likely because the perturbed data is too dissimilar from the original.

Therefore, I found the best model performance and robustness is achieved when using the untargeted PGD attack algorithm, although both untargeted and targeted PGD perform well. Rather, it is the PGD attack parameterization which is the more important factor to consider. The best conditions are when α is small, *steps* is large. Next, we will discuss the effect of ϵ on the accuracy of our model, which was not explored during the training phase. During training, ϵ was equal to 0.3 for all three classifiers. We will also compare models on their testing accuracy which, in general, is more important that the training accurayc.

6.3 Testing Performance

Table [1] shows the testing accuracy for different values of ϵ of our 4 classifiers. During testing, we use many *steps* and a small α to increase likeliness of convergence to an adversarial example, as explained previously. Unsurprisingly, the *Baseline* classifier performance plummets as ϵ increases. This baseline model may be the most accurate on the original data (ϵ=0), but it is not robust to attack.

In Figure [5] we can see that $Classifier\_One$ and $Classifier\_Three$ again outperform $Classifier\_Two$. Also, for testing data perturbed using untargeted PGD, $Classifier\_One$ (also utilizing the untargeted PGD algorithm) performs best. $Classifier\_Three$, utilizing targeted PGD, performs better on the targeted testing data. The reason is clear here- since each classifier is trained on

their respective untargeted or targeted data, these classifiers will perform better for that particular perturbation type.

Additionally, we note that as ϵ increases, data becomes more perturbed. Each of the 4 classifiers' performance decreases as ϵ increases. Once any level of perturbation is introduced, the *Baseline* model performance plummets. For the other three classifiers, while ϵ is smaller than or equal to 0.3 (used during training) the accuracy remains high. Once ϵ is equal to 0.45 (higher than during training), all three models struggle to make predictions, and perform almost as badly as the *Baseline*.

Therefore, when implementing PGD attack, one must be careful of their choice of ϵ. When ϵ is close to and larger than 0.5, the radius of the l^∞ ball is bigger, allowing for more perturbation variability.

For model robustness, during training ϵ should be chosen sufficiently large, but smaller than 0.5. By contract, during testing or a real attack, we will always prefer a smaller ϵ. If the attacker uses a large ϵ our robust models will still struggle to make correct predictions on the data.

7 Conclusion

I was able to successfully implement Targeted and Untargeted PGD attack to create perturbed images and for the purpose of training a robust CNN model on the MNIST dataset. Both algorithms were suitable for training accurate and robust models.

I investigated the effects of α, ϵ, and *steps* on the PGD attack algorithm training and testing performance. I found that during training, we will like α small, number of *steps* large, and ϵ slightly bellow 0.5. When testing the classifiers on unseen data, I found that as ϵ increased, the models were unable to make accurate predictions on the perturbed data.

In conclusion, PGD attack can be used to train and deploy accurate and robust models, which will be more immune to adversarial attacks that baseline models.