SIL-765 Network System security

Assignment-5 Report

ML Classification and Adversarial Attacks

April 28, 2024

Submitted By

Anjali Singh

M.Tech Cyber Security

2023JCS2565

Submitted To

Mr. Vireshwar Kumar

Indian Institute Of Technology, Delhi

1 Building and Evaluating an ANN Model

1.1 Data Preparation

I split the MNIST dataset into a 60:10 ratio for training and testing, respectively. The training set, comprising 60% of the data, was saved in a directory labeled ”train data,” while the remaining 10% constituted the test set, stored in a directory named ”test data.” This approach ensures a balanced distribution of data for model training and evaluation. By organizing the data in separate folders, it becomes easier to manage and access the datasets for training and testing purposes, facilitating efficient model development and evaluation processes.

1.2 Model Implementation

The model architecture consists of three fully connected layers (also known as dense layers). The input layer has 784 neurons, corresponding to the 28x28 size of the input MNIST images. This layer is connected to a hidden layer with 128 neurons, followed by another hidden layer with 64 neurons, and finally an output layer with 10 neurons, representing the 10 classes in the MNIST dataset (digits 0 through 9).

The **hyperparameters** used in training the model are as follows:

1. **Batch Size:** It determines the number of samples that will be propagated through the network at each iteration. In this case, a batch size of 1 is used, meaning that each training sample is processed individually.

2. **Learning Rate:** It controls the step size during the optimization process. A learning rate of 0.01 is used here.

3. **Number of Epochs:** An epoch refers to one complete pass through the entire training dataset. The model is trained for 20 epochs in this scenario.

1.3 Model Training

During training, the model utilizes the **Stochastic Gradient Descent (SGD)** optimizer with the specified learning rate. The **Cross Entropy Loss function** is employed as the criterion to measure the difference between the predicted and actual class labels. Additionally, the model employs ReLU activation functions after the first two fully connected layers to introduce non-linearity.

After training, the model's performance is evaluated on both the training and validation datasets. The training accuracy of the Model is **99.55%** and validation accuracy of the model is **97.47%**. Finally, the trained model is saved for future use.

1.4 Model Evaluation

Now we evaluated the performance of the trained model based on its accuracy in classifying the test data and recorded the test accuracy. The Test Accuracy is **97.47%**.

1.5 Analysis and Improvement

To improve the robustness of the model, several techniques and strategies can be employed:

1. **Data Augmentation:** Augmenting the training data with transformations such as rotations, translations, scaling, and flipping can help the model generalize better to unseen variations in the test data.

2. **Dropout:** Dropout regularization randomly drops neurons during training, which prevents the model from relying too much on specific features and encourages it to learn more robust representations.

3. **Weight Regularization:** L2 regularization (weight decay) penalizes large weights in the model, preventing overfitting and improving generalization.

4. **Learning Rate Scheduling:** Adjusting the learning rate during training can help the model converge faster and avoid getting stuck in local minima. Techniques like learning rate decay or using adaptive learning rate algorithms like Adam can be beneficial.

5. **Early Stopping:** Monitor the model's performance on a validation set and stop training when the performance starts to degrade, thus preventing overfitting.

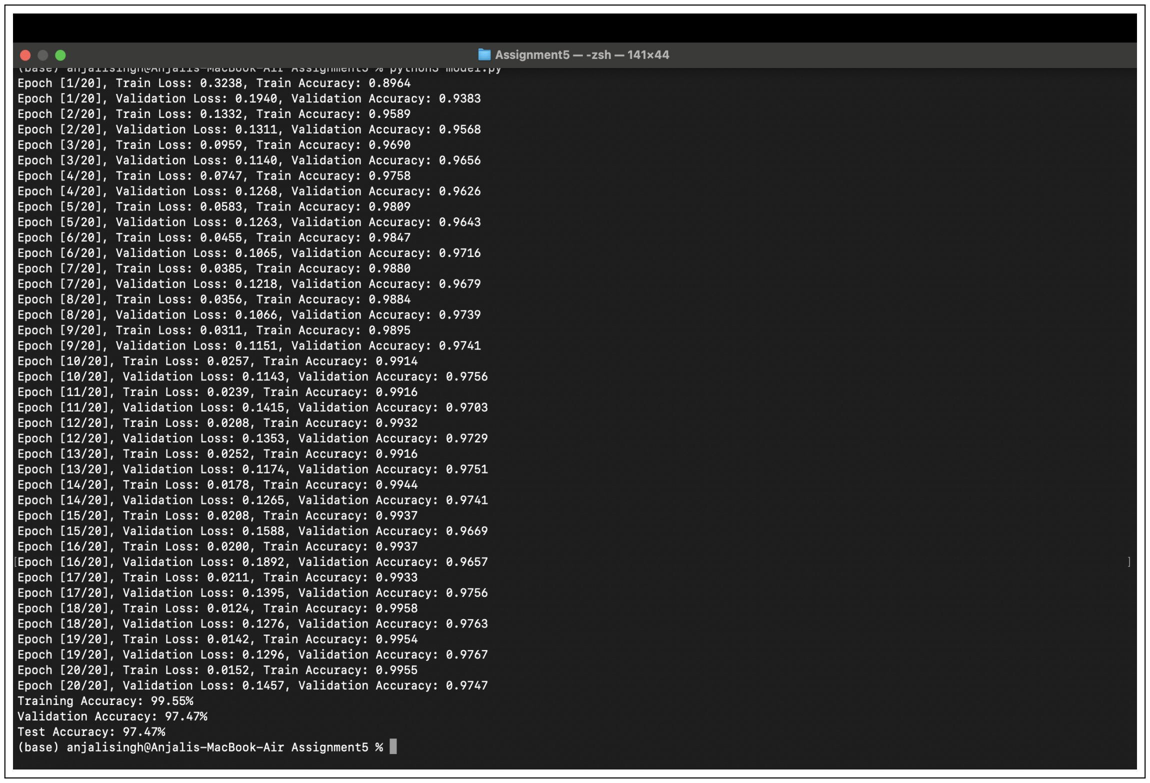
6. **Model Ensembling:** Combining predictions from multiple independently trained models can often result in better performance than using a single model, as it reduces the risk of overfitting and captures a broader range of patterns in the data.

7. **Transfer Learning:** Utilizing pre-trained models on similar tasks or datasets and fine-tuning them on the specific task at hand can help leverage knowledge learned from large datasets and improve generalization.

8. **Batch Normalization:** Normalizing the activations of each layer can help stabilize the training process and accelerate convergence, leading to improved generalization.

By incorporating these techniques into the training pipeline, the model can become more robust, perform better on unseen data, and generalize well across different variations of the input.

1.6 Performance Metrics



```
(base) anjalisingh@Anjalis-MacBook-Air Assignment5 % python3 model.py
Epoch [1/20], Train Loss: 0.3238, Train Accuracy: 0.8964
Epoch [1/20], Validation Loss: 0.1940, Validation Accuracy: 0.9383
Epoch [2/20], Train Loss: 0.1332, Train Accuracy: 0.9589
Epoch [2/20], Validation Loss: 0.1311, Validation Accuracy: 0.9568
Epoch [3/20], Train Loss: 0.0959, Train Accuracy: 0.9690
Epoch [3/20], Validation Loss: 0.1140, Validation Accuracy: 0.9656
Epoch [4/20], Train Loss: 0.0747, Train Accuracy: 0.9758
Epoch [4/20], Validation Loss: 0.1268, Validation Accuracy: 0.9626
Epoch [5/20], Train Loss: 0.0583, Train Accuracy: 0.9809
Epoch [5/20], Validation Loss: 0.1263, Validation Accuracy: 0.9643
Epoch [6/20], Train Loss: 0.0455, Train Accuracy: 0.9847
Epoch [6/20], Validation Loss: 0.1065, Validation Accuracy: 0.9716
Epoch [7/20], Train Loss: 0.0385, Train Accuracy: 0.9880
Epoch [7/20], Validation Loss: 0.1218, Validation Accuracy: 0.9679
Epoch [8/20], Train Loss: 0.0356, Train Accuracy: 0.9884
Epoch [8/20], Validation Loss: 0.1066, Validation Accuracy: 0.9739
Epoch [9/20], Train Loss: 0.0311, Train Accuracy: 0.9895
Epoch [9/20], Validation Loss: 0.1151, Validation Accuracy: 0.9741
Epoch [10/20], Train Loss: 0.0257, Train Accuracy: 0.9914
Epoch [10/20], Validation Loss: 0.1143, Validation Accuracy: 0.9756
Epoch [11/20], Train Loss: 0.0239, Train Accuracy: 0.9916
Epoch [11/20], Validation Loss: 0.1415, Validation Accuracy: 0.9703
Epoch [12/20], Train Loss: 0.0208, Train Accuracy: 0.9932
Epoch [12/20], Validation Loss: 0.1353, Validation Accuracy: 0.9729
Epoch [13/20], Train Loss: 0.0252, Train Accuracy: 0.9916
Epoch [13/20], Validation Loss: 0.1174, Validation Accuracy: 0.9751
Epoch [14/20], Train Loss: 0.0178, Train Accuracy: 0.9944
Epoch [14/20], Validation Loss: 0.1265, Validation Accuracy: 0.9741
Epoch [15/20], Train Loss: 0.0208, Train Accuracy: 0.9937
Epoch [15/20], Validation Loss: 0.1588, Validation Accuracy: 0.9669
Epoch [16/20], Train Loss: 0.0200, Train Accuracy: 0.9937
Epoch [16/20], Validation Loss: 0.1892, Validation Accuracy: 0.9657
Epoch [17/20], Train Loss: 0.0211, Train Accuracy: 0.9933
Epoch [17/20], Validation Loss: 0.1395, Validation Accuracy: 0.9756
Epoch [18/20], Train Loss: 0.0124, Train Accuracy: 0.9958
Epoch [18/20], Validation Loss: 0.1276, Validation Accuracy: 0.9763
Epoch [19/20], Train Loss: 0.0142, Train Accuracy: 0.9954
Epoch [19/20], Validation Loss: 0.1296, Validation Accuracy: 0.9767
Epoch [20/20], Train Loss: 0.0152, Train Accuracy: 0.9955
Epoch [20/20], Validation Loss: 0.1457, Validation Accuracy: 0.9747
Training Accuracy: 99.55%
Validation Accuracy: 97.47%
Test Accuracy: 97.47%
(base) anjalisingh@Anjalis-MacBook-Air Assignment5 %
```

Figure 1: Results

Axes:

- **X-axis (Horizontal):** Likely represents the number of epochs (iterations) the model has been trained on. Each epoch involves going through the entire training dataset once.
- **Y-axis (Vertical):** Represents the loss value. Lower loss generally indicates better model performance.

Curves:

- **Training Loss:** This curve shows how the model's loss on the training data changes as the training progresses.

- **Validation Loss:** This curve shows how the model's loss on a separate validation dataset changes during training. The validation set is used to assess how well the model generalizes to unseen data and avoid overfitting. Ideally, the validation loss curve should also decrease or at least not significantly increase.

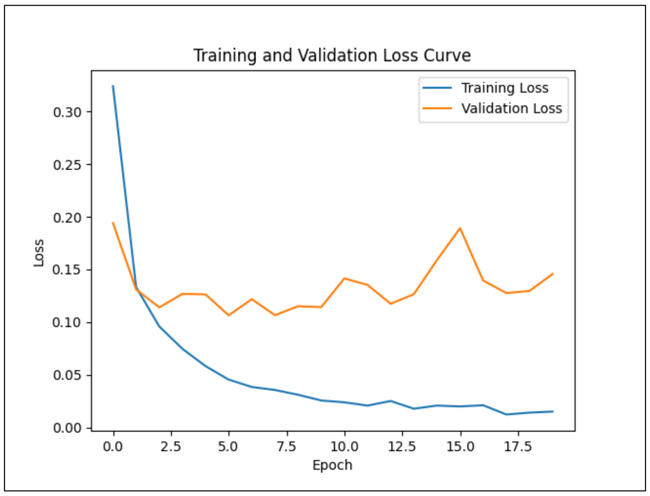
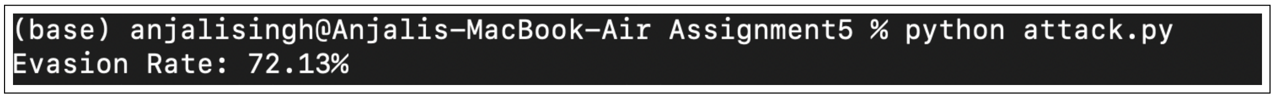


Figure 2: Training Validation Loss Curve

2 Generating Adversarial Examples

2.1 Evasion Rate

Evasion Rate is a metric used to measure the evasiveness of adversarial examples against a trained ML model. The evasion rate is calculated by taking the ratio of the total number of adversarial examples misclassified and the total number of examples. For my model is **Evasion Rate: 72.13%**.



```
(base) anjalisingh@Anjalis-MacBook-Air Assignment5 % python attack.py
Evasion Rate: 72.13%
```

Figure 3:

2.2 L2 norm

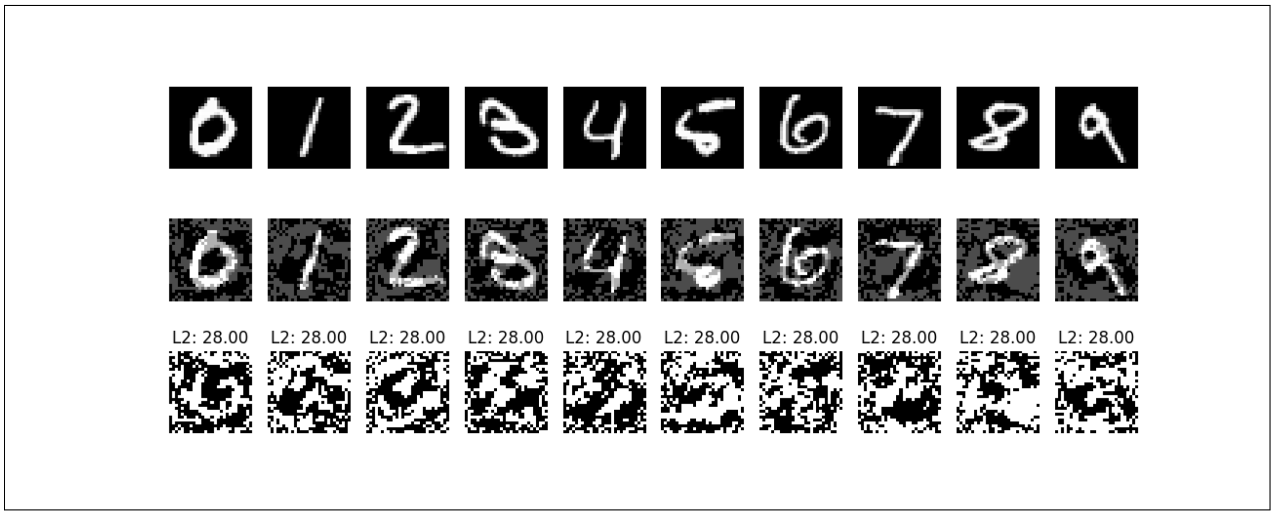
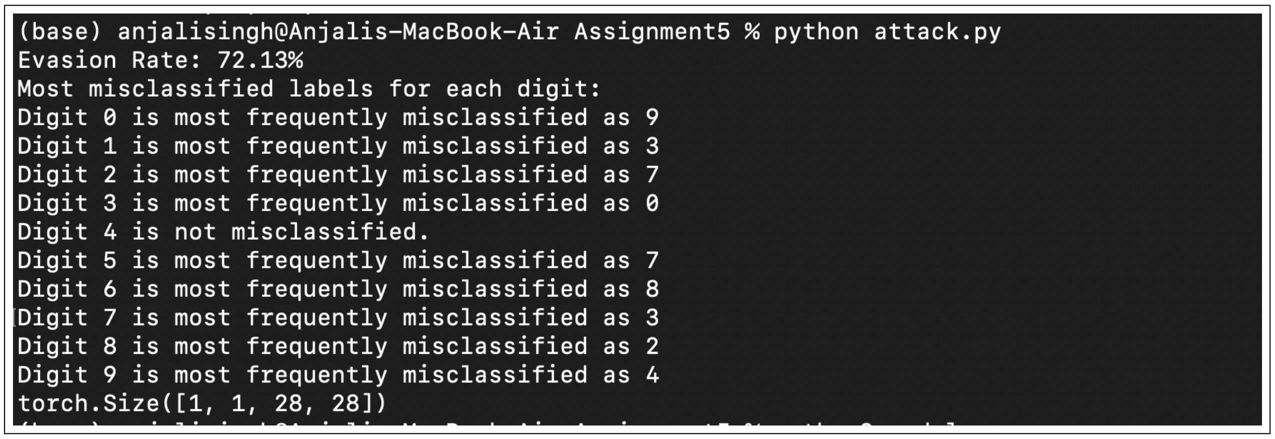


Figure 4:

In figure 4 the first row shows original images of ten different digits, the second row shows their corresponding adversarial examples, and the final row shows the L2 norm of their difference (L2 norm of the adversarial noise).

2.3 Missclassification



```
(base) anjalisingh@Anjalis-MacBook-Air Assignment5 % python attack.py
Evasion Rate: 72.13%
Most misclassified labels for each digit:
Digit 0 is most frequently misclassified as 9
Digit 1 is most frequently misclassified as 3
Digit 2 is most frequently misclassified as 7
Digit 3 is most frequently misclassified as 0
Digit 4 is not misclassified.
Digit 5 is most frequently misclassified as 7
Digit 6 is most frequently misclassified as 8
Digit 7 is most frequently misclassified as 3
Digit 8 is most frequently misclassified as 2
Digit 9 is most frequently misclassified as 4
torch.Size([1, 1, 28, 28])
```

Figure 5: Misclassified labels for each digit

- **Digit 0 (misclassified as 9):** Both digits 0 and 9 have circular shapes. However, digit 9 often has a longer tail or descender, which might be the cause of confusion.

- **Digit 1 (misclassified as 3):** Both digits 1 and 3 have similar vertical strokes. The main difference lies in the presence of a horizontal stroke in digit 3, which might lead to misclassification.

- **Digit 2 (misclassified as 7):** Both digits 2 and 7 have similar curved shapes, particularly in their upper halves. The main distinguishing feature is the absence of a horizontal line in digit 2.

- **Digit 3 (misclassified as 0):** Both digits 0 and 3 have circular shapes, which might lead to confusion. Additionally, if the top part of digit 3 is poorly formed or not clearly defined, it can resemble the circular shape of digit 0.

- **Digit 5 (misclassified as 7):** Both digits 5 and 7 have curved shapes, particularly in their upper halves. The presence of a horizontal stroke in digit 5 might lead to misclassification when it is poorly formed or not clearly defined.

- **Digit 6 (misclassified as 8):** Both digits 6 and 8 have similar curved shapes, particularly in their lower halves. The main distinguishing feature is the absence of a closed loop at the top in digit 6.

- **Digit 7 (misclassified as 3):** As mentioned earlier, both digits 3 and 7 have similar vertical strokes. The main difference lies in the presence of a horizontal stroke in digit 3, which might lead to misclassification.

- **Digit 8 (misclassified as 2):** Both digits 2 and 8 have curved shapes. However, digit 2 typically has a more pronounced curve at the top, whereas digit 8 has a more symmetrical shape. Misclassification might occur if the top part of digit 8 is poorly formed or not clearly defined.

- **Digit 9 (misclassified as 4):** Both digits 4 and 9 have similar circular shapes. However, digit 9 often has a longer tail or descender, which might lead to misclassification.

In summary, the structural similarities between the digits and their most misclassified counterparts primarily involve similar shapes or strokes, with subtle differences in specific features such as loops, tails, or additional strokes. These similarities can contribute to misclassification, especially when the distinguishing features are not clearly defined or are poorly formed in the input images.