University of Dayton

School of Engineering

Department of Electrical and Computer Engineering

ECE 477/595: Artificial Neural Networks

Project 1: Single Layer Perceptron

Instructor: Prof. K. Asari Vijayan

Student Name: Serge Alhalbi **Student ID:** 101682971

Date: 9/10/2022

I. Task 1

Methodology

Loading Steps

- Using the “loadMNISTImages” and “loadMNISTLabels” functions, we load the both the training and testing images with their labels from the uploaded files respectively.
  Initially, both the training and testing datasets are made up of images with pixel values ranging from 0 to 255 (8-bit pixel). These pixel values are then normalized (divided by 255), allowing each pixel to have a value between 0 and 1.

- A subset is created from the original datasets using the “balance_MNIST_selection” function. The function is simple and clear. I have written another function that can be used to select images at random from each dataset. The user may specify the number of training and testing images any of these functions.

- If simple thresholding is being used, the data is binarized bipolarly taking either 1 or -1 as a pixel value. Otherwise, the values of the pixels would still range from 0 to 1.
  Bipolar images are created through simple thresholding at 0.5 the original data (output = 0 if input < 0.5 and output = 1 if input > 0.5). Then the output is subtracted by 0.5 to make the data take either -0.5 or 0.5 as values. Lastly, the data is multiplied by 2 so that the final output takes either -1 or 1 as values.

Training Steps

The edited function "SingleLayerPerceptron" is used to implement the forward and backward passes of the training process.

1. Input the training dataset:
   $x^s;\ s = 1,2,\dots,P.$ P: number of patterns. (Size: $N \times P$)

2. Initialize the weights:
   $w_{ji};\ j = 1,2,\dots,M;\ i = 1,2,\dots,N.$ M: output neurons & N: input neurons. (Size: $N \times M$)

3. Loop over all the patterns and apply x^s for $s = 1,2,\dots,P$: (Size: $N \times M$)

   a. Compute the net for each neuron:
      $$Net_j = \sum_{i=1}^{N} w_{ji} \cdot x_i^s;\ j = 1,2,\dots,M$$
      (Size: $1 \times M$)

   b. Compute the actual outputs:
      $$y_j = \begin{cases} 1\ if\ Net_j > 0 \\ 0\ if\ Net_j < 0 \end{cases};\ j = 1,2,\dots,M$$ (Size: $1 \times M$)
      "0" is a simple threshold value.

   c. Compute the errors:
      $e_j = d_j - y_j;\ j = 1,2,\dots,M$ where d_j is the given desired output. (Size: $1 \times M$)
      e_{ji} is to be used when implementing the change of weights. (Size: $N \times M$)

   d. Find the change of weights according to the verified and proven delta rule:
      $\Delta w_{ji} = \eta(e_j)x_i;\ j = 1,2,\dots,M;\ i = 1,2,\dots,N$ (Size: $N \times M$)
      η is a learning rate that determine how fast the change is ($0 < \eta < 1$). (hyperparameter)

   e. Update the weights:
      $w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t);\ \forall i,j$

   f. Compute the squared error of pattern s: (Add this squared error to the previous pattern squared error to accumulate all the pattern squared errors)
      $$(e_j)^2 = \sum_{j=1}^{M} (d_j - y_j)^2$$

4. Compute the mean squared error: (for one complete data iteration)
   $$E = \frac{1}{M \cdot P} \sum_{j=1}^{M} \sum_{s=1}^{P} (d_j - y_j)^2$$

5. Repeat steps 3 to 4 until the mean squared error is less than the used threshold. (tolerance)
   The user may define a maximum number of iterations.

6. Use all the mean squared errors to plot the E vs the number of iterations:
   The user should expect the squared errors to decrease throughout the iterations.

Testing Steps

Using the testing dataset:

1. Loop over all the patterns and apply x^s for $s = 1,2, \dots, P_{test}$:

    a. Compute the net for each neuron:
    $$Net_j = \sum_{i=1}^{N} w_{ji} \cdot x_i;\ j = 1,2, \dots, M$$
    (Size: $1 \times M$)

    b. Compute the actual outputs:
    $$y_j = \begin{cases} 1\ if\ Net_j > 0 \\ 0\ if\ Net_j < 0 \end{cases};\ j = 1,2, \dots, M \quad \text{(Size: } 1 \times M\text{)}$$
    "0" is a simple threshold value.

    c. Find: the true positives, false negatives, false positives, and true negatives: (TP, FN, FP, and TN)

    d. Find: the true positive rate and the false positive rate: using the following formulas: (TPR and FPR)
    $$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
    $$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

2. Plot the TPR and FPR vs the handwritten digits classes. (0 to 9)

Results

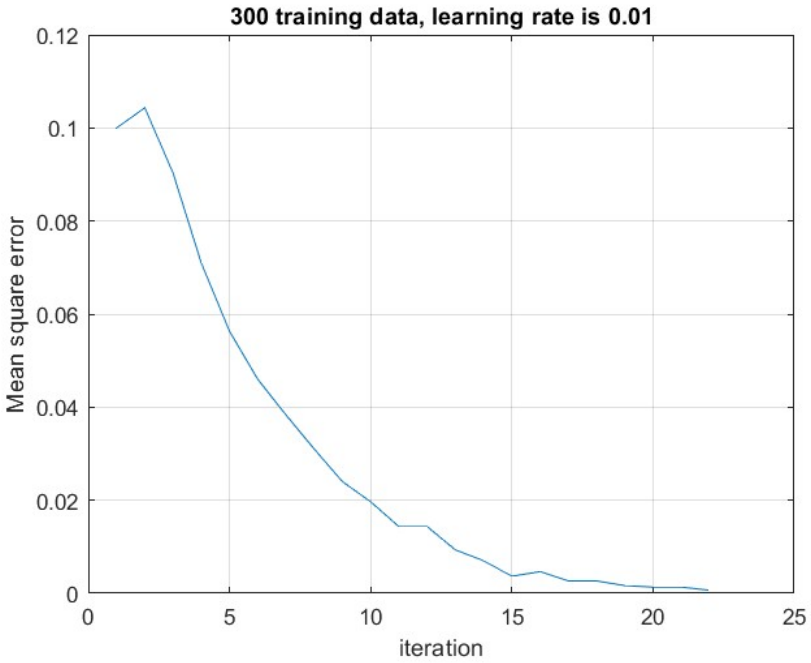

Figure 1: Error - 300 training data, 100 testing data, learning rate = 0.01, simple threshold, 22 iterations

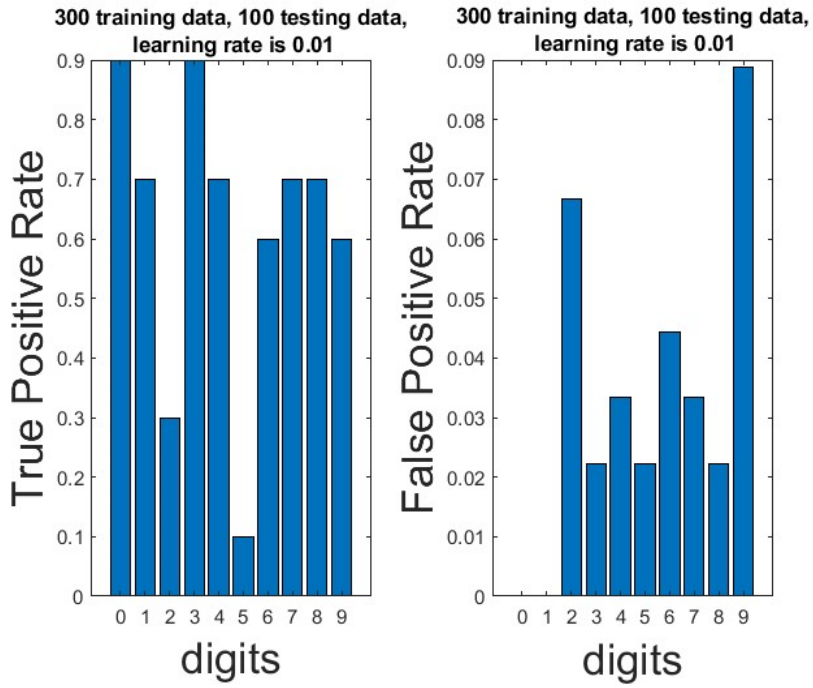

Figure 2: Plots - 300 training data, 100 testing data, learning rate = 0.01, simple threshold

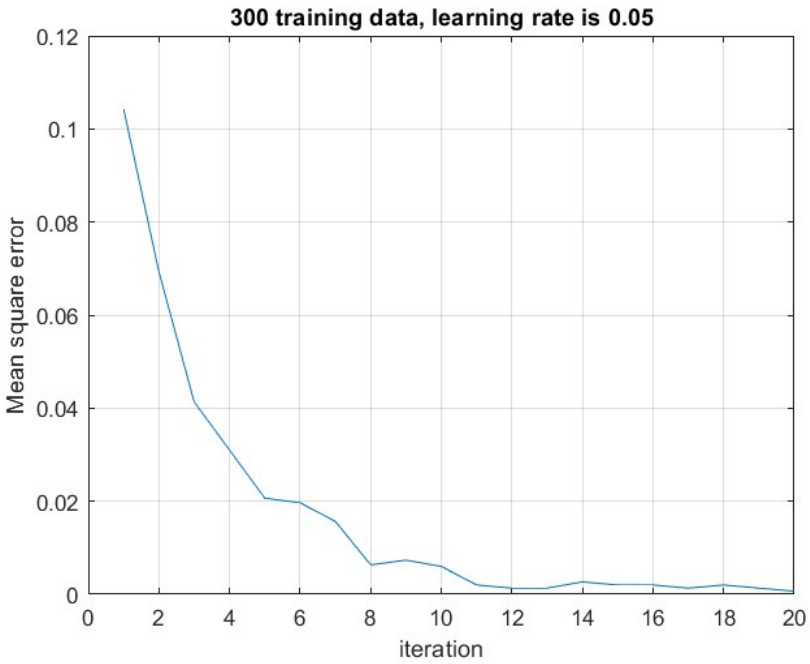

Figure 3: Error - 300 training data, 100 testing data, learning rate = 0.05, simple threshold, 20 iterations

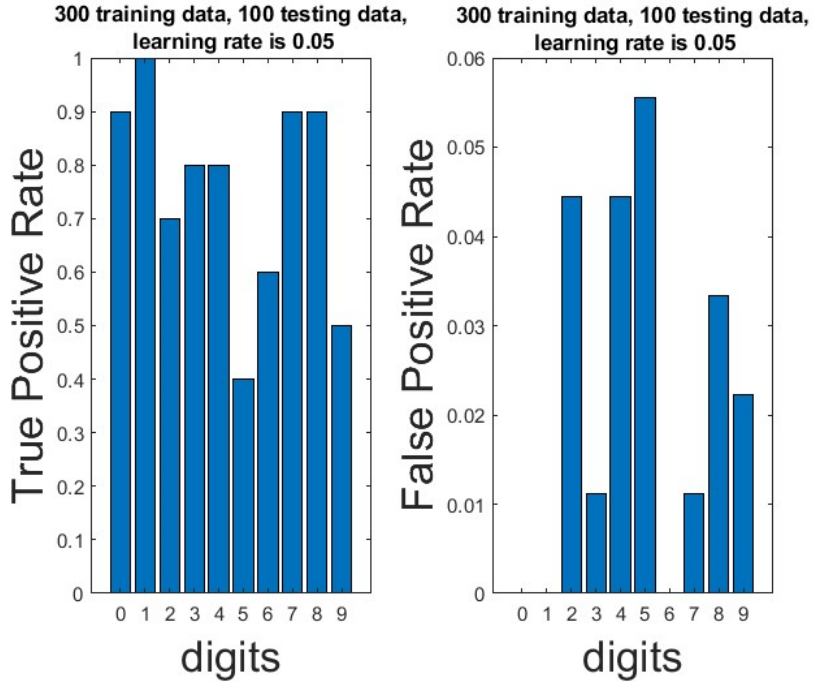

Figure 4: Plots - 300 training data, 100 testing data, learning rate = 0.05, simple threshold

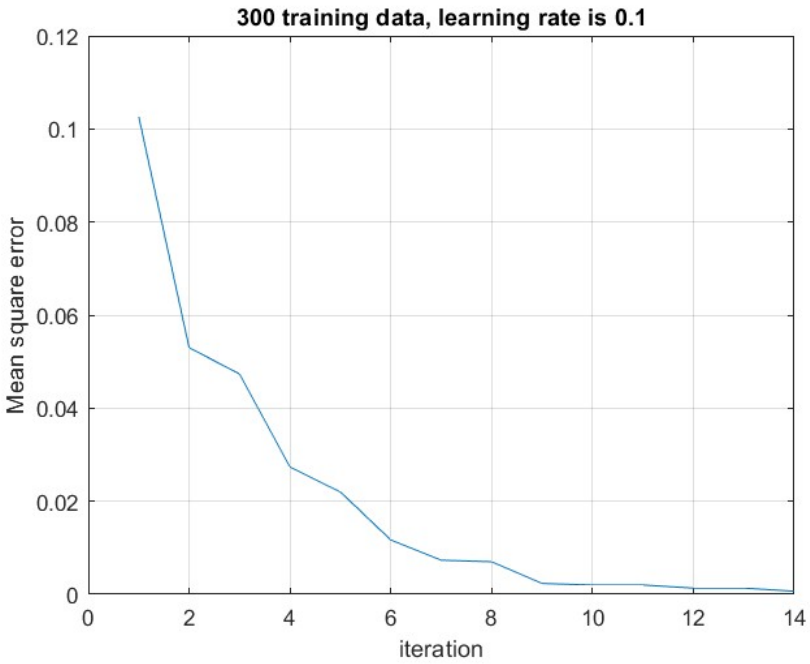

Figure 5: Error - 300 training data, 100 testing data, learning rate = 0.1, simple threshold, 14 iterations

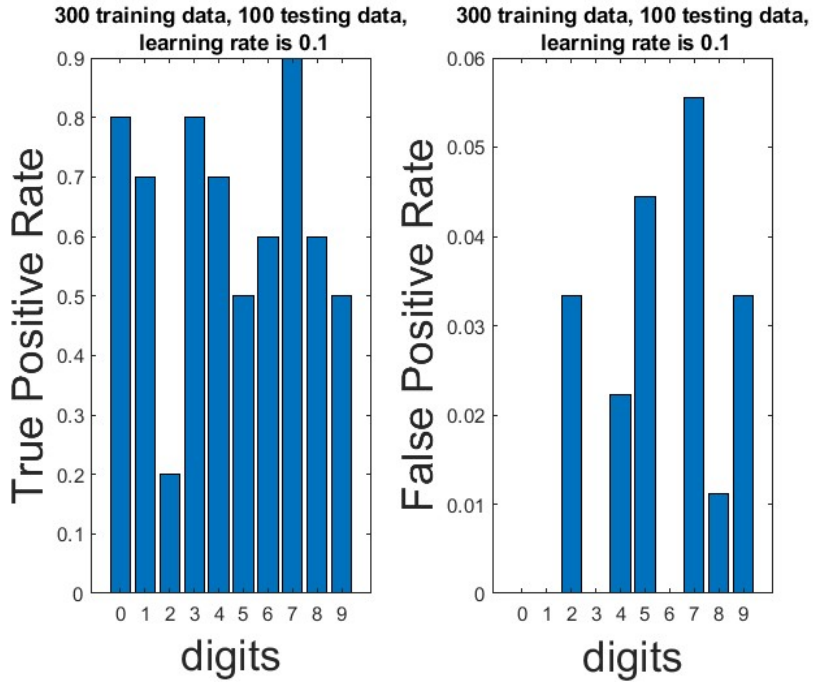

Figure 6: Plots - 300 training data, 100 testing data, learning rate = 0.1, simple threshold

II. Task 2

Methodology

A bigger subset of the original data is used in this task. The methodology is still the same.

Results

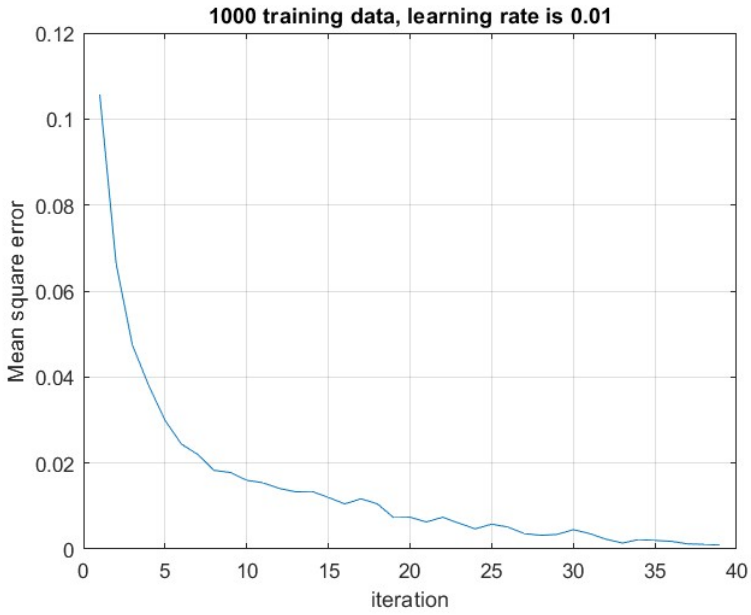

Figure 7: Error - 1000 training data, 300 testing data, learning rate = 0.01, simple threshold, 39 iterations

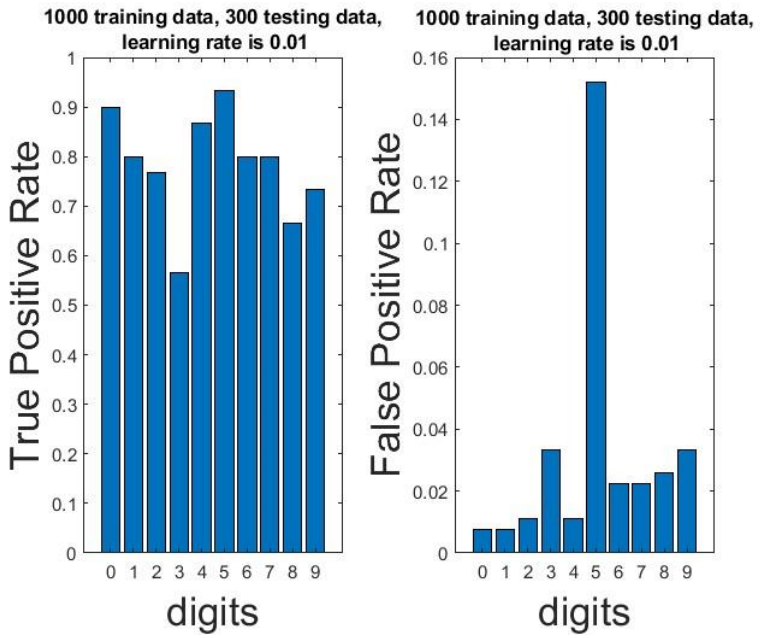

Figure 8: Plots - 1000 training data, 300 testing data, learning rate = 0.01, simple threshold

Figure 9: Error - 1000 training data, 300 testing data, learning rate = 0.05, simple threshold, 38 iterations

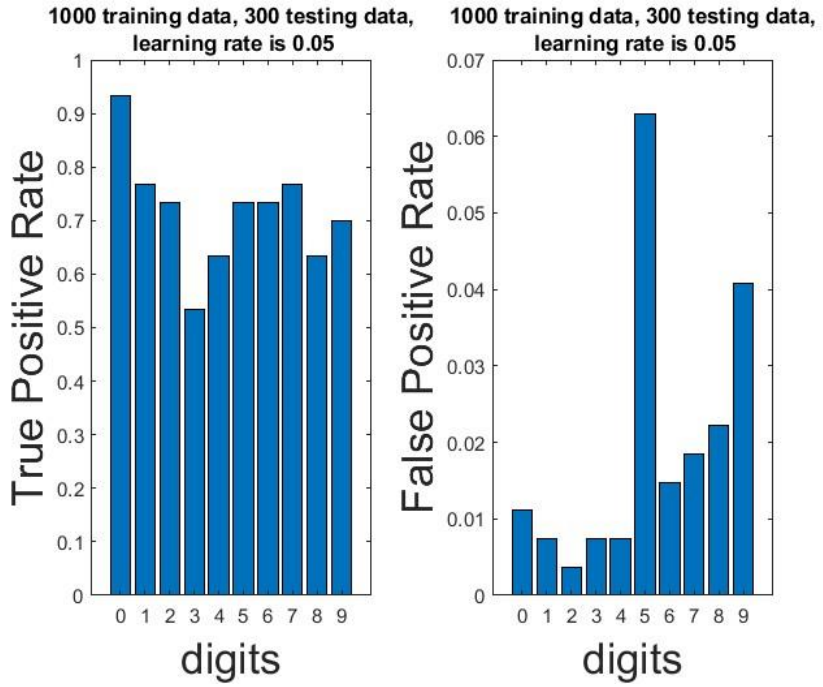

Figure 10: Plots - 1000 training data, 300 testing data, learning rate = 0.05, simple threshold

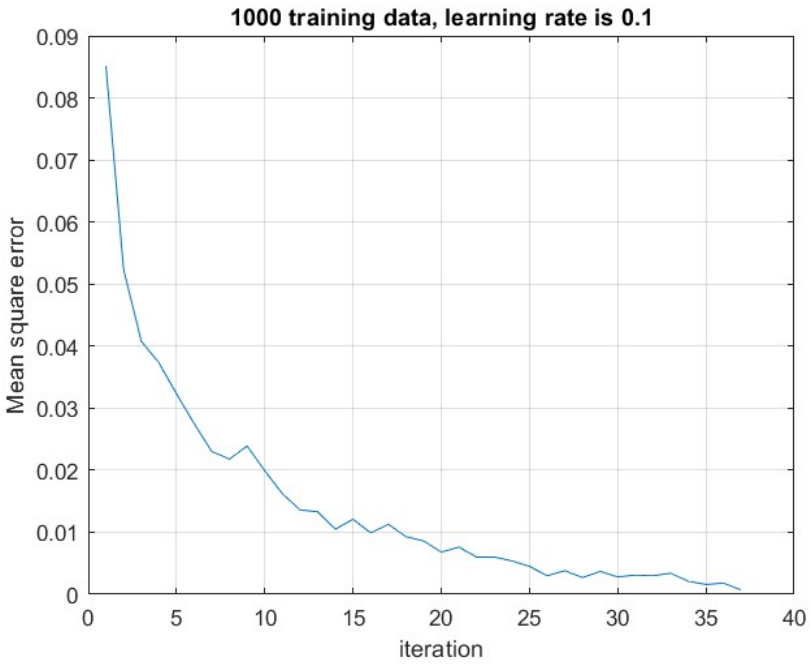

Figure 11: Error - 1000 training data, 300 testing data, learning rate = 0.1, simple threshold, 27 iterations

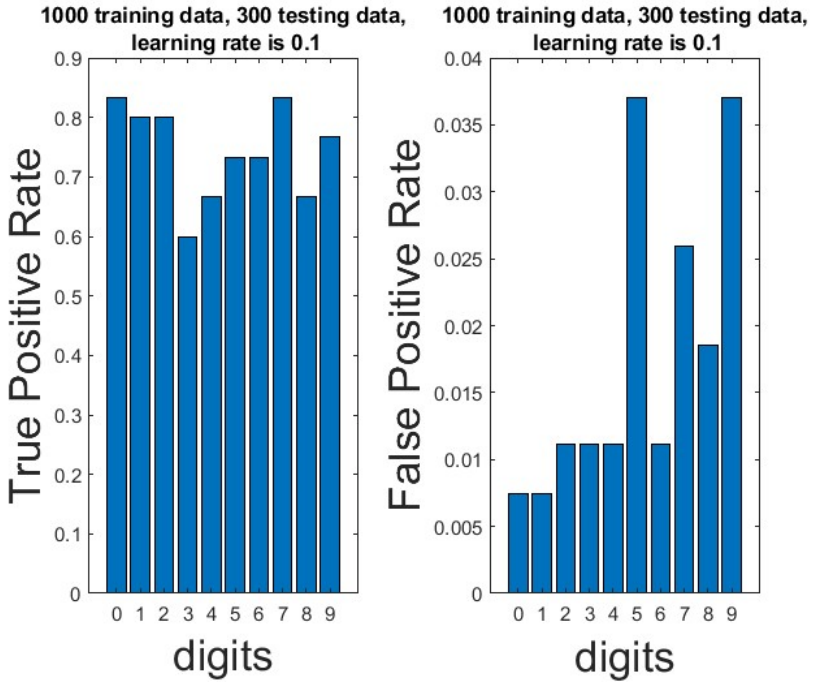

Figure 12: Plots - 1000 training data, 300 testing data, learning rate = 0.1, simple threshold

III. Task 3

Methodology

In this task, the original data is only normalized, and the sigmoid function is used to threshold the output. Everything else is still the same as in tasks 1 and 2.

Actual output:

$$y_j = \frac{1}{1 + e^{-Net_j}}$$

Results

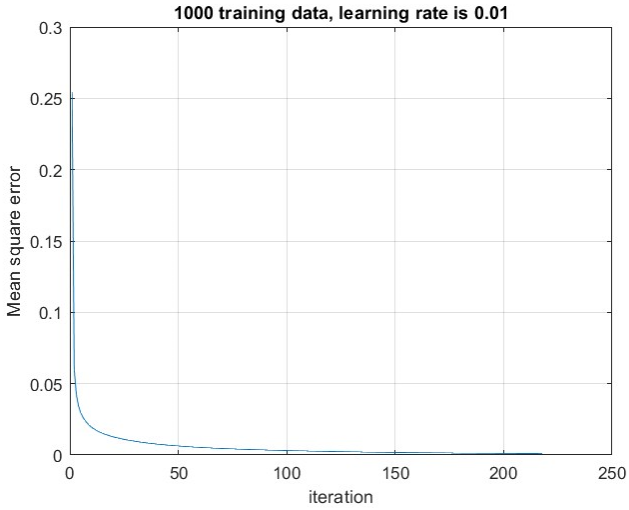

Figure 13: Error - 1000 training data, 300 testing data, learning rate = 0.01, sigmoid threshold, 218 iterations

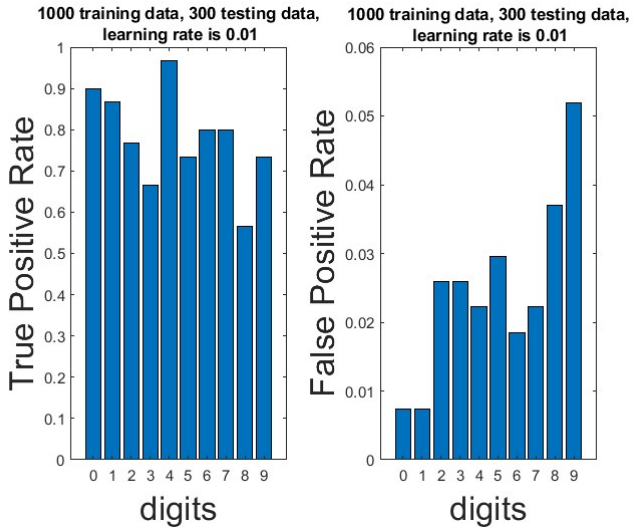

Figure 14: Plots - 1000 training data, 300 testing data, learning rate = 0.01, sigmoid threshold

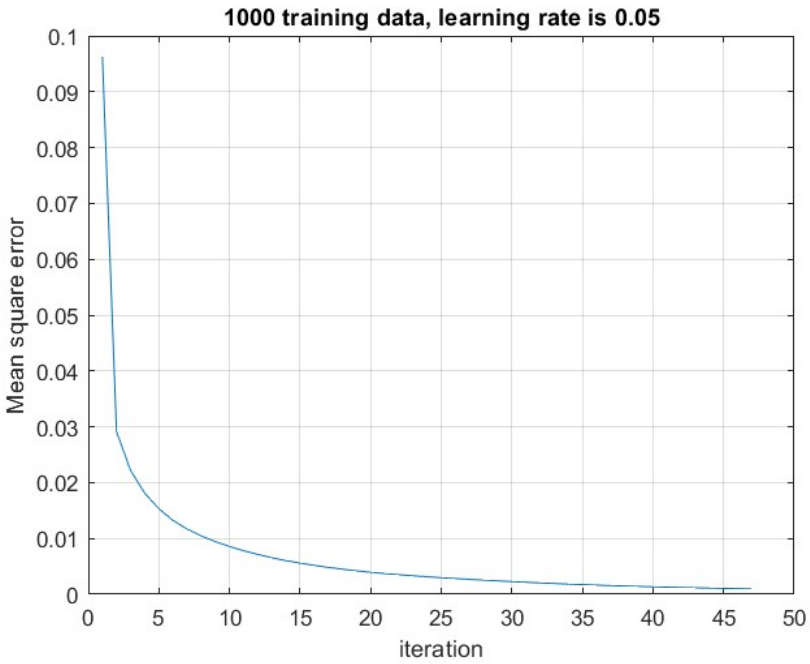

Figure 15: Error - 1000 training data, 300 testing data, learning rate = 0.05, sigmoid threshold, 47 iterations

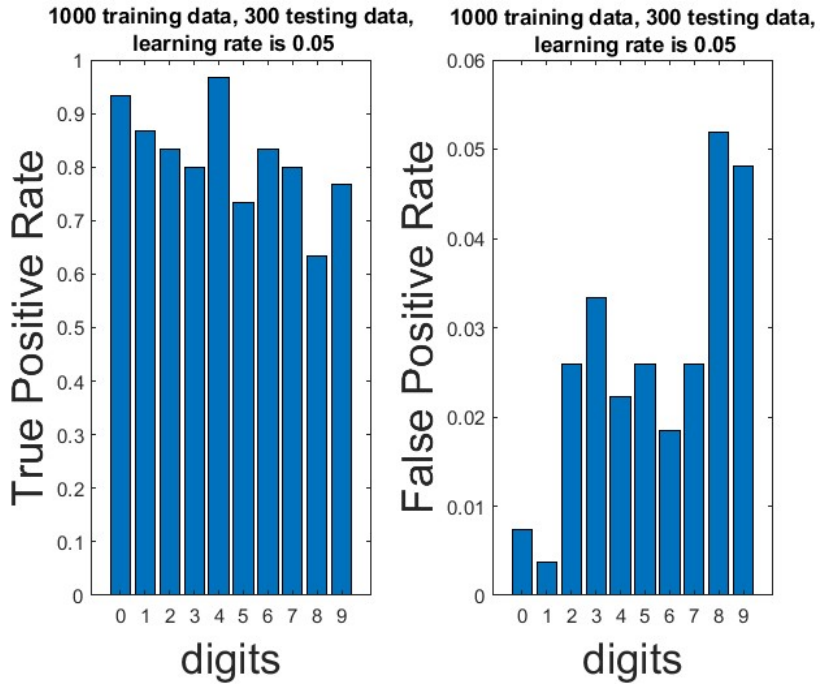

Figure 16: Plots - 1000 training data, 300 testing data, learning rate = 0.05, sigmoid threshold

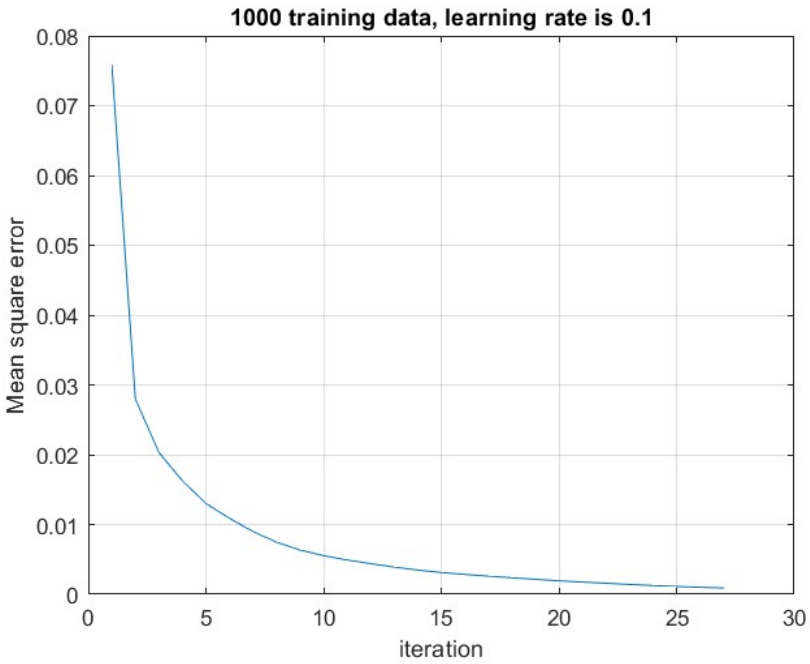

Figure 17: Error - 1000 training data, 300 testing data, learning rate = 0.1, sigmoid threshold, 27 iterations

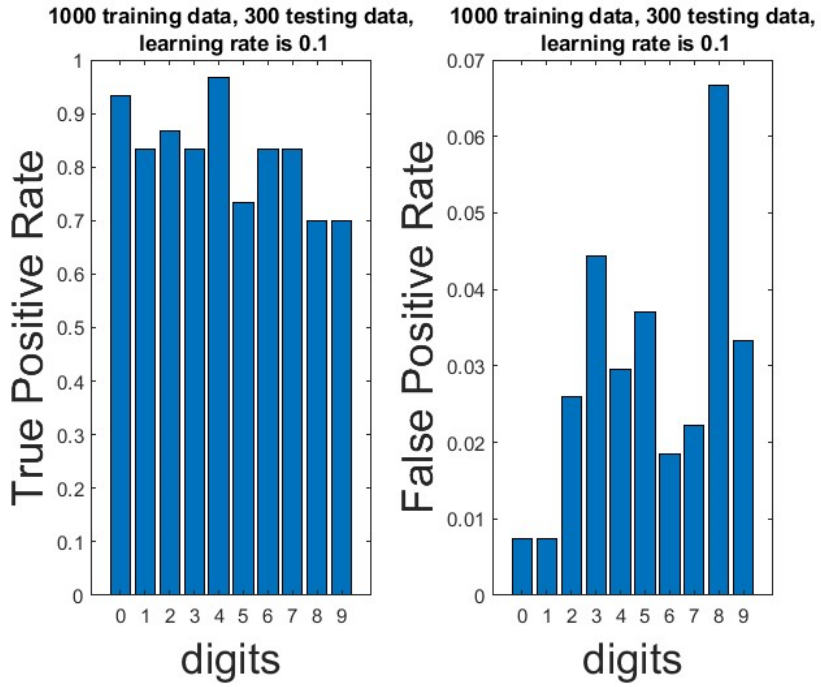

Figure 18: Plots - 1000 training data, 300 testing data, learning rate = 0.1, sigmoid threshold

IV. Conclusion/Comments

The goal of this assignment was met, and a single layer perceptron was developed to classify handwritten images. Some points may be discussed below:

- All tasks and subtasks have a decreasing mean squared error as the number of iterations increases. This demonstrates that the machine only improves when the algorithm is repeated. This is evidence that the algorithm is reliable.

- Comparing task 1 and task 2, using bigger training datasets may help find the best set of weights for the machine.

- When using simple thresholding, the mean squared error could go up a bit in some iterations.

- Different sets of weights were computed for different datasets. This is reasonable since one image greatly affects the learning process.

- Using the sigmoid function was advantageous because the mean squared error was decreasing very quickly and smoothly at each iteration. The sigmoid function might be better than a simple thresholding function according to the cases above.

- In the cases mentioned above, the number of iterations gets lower when the learning rate increases. This is logical since a bigger step towards the global minimum error is taken with a higher learning rate value.

- It should also be noted that when comparing task 1 to task 2, using larger training dataset increased somewhat the number of iterations.

- For task 2, it may be noticed that a smaller learning rate is more reliable than a higher one since the true positive rates are almost at their highest and the false positive rates are at their lowest compared to the other subtasks. This may be true because, while more steps are being taken towards the global minimum error, they are more accurate.

- It may be noted that for the handwritten digits “0” and “1”, the true positive rate is almost always higher than the rates for the other digits. As well, the false positive rate is always at its lowest compared to the other handwritten digits. This could be since "0" and "1" have simple features compared to the others, such as a straight line and an ellipse.