University *of* Dayton

School of Engineering

Department of Electrical and Computer Engineering

ECE 477/595: Artificial Neural Networks

Project 2: Multilayer Layer Perceptron

Instructor: Prof. K. Asari Vijayan

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

Date: 9/26/2022

I. Tasks 1, 2, and 3

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.

Training Steps

The edited function "MultiLayerPerceptron" 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. (X: Size: $N \times P$)
   (x^s: Size: $N \times 1$) and (Size: $N \times L$) is to be used when implementing the change of weights.

2. Initialize some parameters

   a. Initialize the weights:
      $v_{ki};\ k = 1,2,\dots,L;\ i = 1,2,\dots,N$. L: hidden neurons & N: input neurons. (Size: $N \times L$)
      $w_{jk};\ j = 1,2,\dots,M;\ k = 1,2,\dots,L$. M: output neurons & L: hidden neurons. (Size: $L \times M$)
      $-0.5 \le v_{ki}, w_{jk} \le 0.5$

   b. Initialize the learning rate η. ($0 < \eta \ll 1$)

   c. Initialize the momentum term α. ($0 < \alpha \ll 1$)

   d. Initialize the first change of weights:
      $-0.5 \le \Delta v_{ki}(t_0) \le 0.5$ (Size: $N \times L$)
      $-0.5 \le \Delta w_{jk}(t_0) \le 0.5$ (Size: $L \times M$)

   e. The maximum iteration. ($= 1000$)

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 at the hidden layer:
      $$Net_k = \sum_{i=1}^{N} k_{ki} \cdot x_i^s;\ k = 1,2,\dots,L$$
      (Size: $1 \times L$)

   b. Compute the hidden outputs:
      $u_k = \frac{1}{1+e^{-Net_k}}$; $k = 1,2,\dots,L$ (Size: $1 \times L$)
      (Size: $L \times M$) and (Size: $N \times L$) are to be used when implementing the change of weights.

   c. Compute the net for each neuron at the output layer:
      $$Net_j = \sum_{k=1}^{L} w_{jk} \cdot u_k^s;\ j = 1,2,\dots,M$$
      (Size: $1 \times M$)

d. Compute the actual outputs:
$y_j = \frac{1}{1+e^{-Net_j}}$; $j = 1,2,\dots,M$ (Size: $1 \times M$)
(Size: $L \times M$) is to be used when implementing the change of weights.

e. Compute the errors at the output layer:
$e_j = d_j - y_j;\ j = 1,2,\dots,M$ where d_j is the given desired output. (Size: $1 \times M$)
(Size: $L \times M$) is to be used when implementing the change of weights.

f. Compute the errors at the hidden layer:
$e_k = \sum_{j=1}^{M} w_{jk} \cdot e_j \cdot y_j(1 - y_j);\ k = 1,2,\dots,L$ (Size: $1 \times L$)
(Size: $N \times L$) is to be used when implementing the change of weights.

g. Find the change of weights according to the verified and proven delta rule:
$\Delta w_{jk} = \eta \cdot e_j \cdot u_k \cdot y_j(1 - y_j);\ j = 1,2,\dots,M;\ k = 1,2,\dots,L$ (Size: $L \times M$)
$\Delta v_{ki} = \eta \cdot e_k \cdot x_i \cdot u_k(1 - u_k);\ k = 1,2,\dots,L;\ i = 1,2,\dots,N$ (Size: $N \times L$)

h. Update the weights using a momentum term to account for the previous weights' changes:
$w_{jk}(t+1) = w_{jk}(t) + (1-\alpha) \cdot \Delta w_{jk}(t) + \alpha \cdot \Delta w_{jk}(t-1);\ \forall j,k$
$v_{ki}(t+1) = v_{ki}(t) + (1-\alpha) \cdot \Delta v_{ki}(t) + \alpha \cdot \Delta v_{ki}(t-1);\ \forall k,i$

i. 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)
$E = \frac{1}{M \cdot P} \sum_{j=1}^{M} \sum_{s=1}^{P} (e_j^s)^2$

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. As well, the user may define a condition $E(t) = E(t - t')$ where the algorithm stops at iteration t'.

6. Store the weights and 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^{test} for $test = 1,2, \dots, P_{test}$:

    a. Compute the net for each neuron at the hidden layer:
    $$Net_k = \sum_{i=1}^{N} k_{ki} \cdot x_i;\ k = 1,2, \dots, L$$
    (Size: $1 \times L$)

    b. Compute the hidden outputs:
    $u_k = \frac{1}{1+e^{-Net_k}}$; $k = 1,2, \dots, L$ (Size: $1 \times L$)

    c. Compute the net for each neuron at the output layer:
    $$Net_j = \sum_{k=1}^{L} w_{jk} \cdot u_k;\ j = 1,2, \dots, M$$
    (Size: $1 \times M$)

    d. Compute the actual outputs:
    $y_j = \frac{1}{1+e^{-Net_j}}$; $j = 1,2, \dots, M$ (Size: $1 \times M$)

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

    f. 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

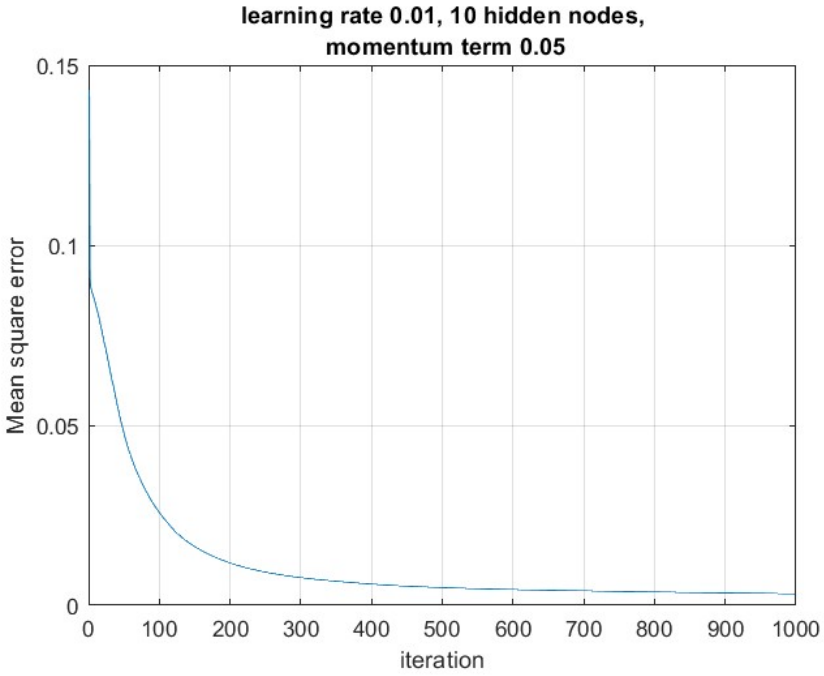

Figure 1: Error - 1000 training data, 300 testing data, learning rate = 0.01, hidden nodes = 10, momentum = 0.05

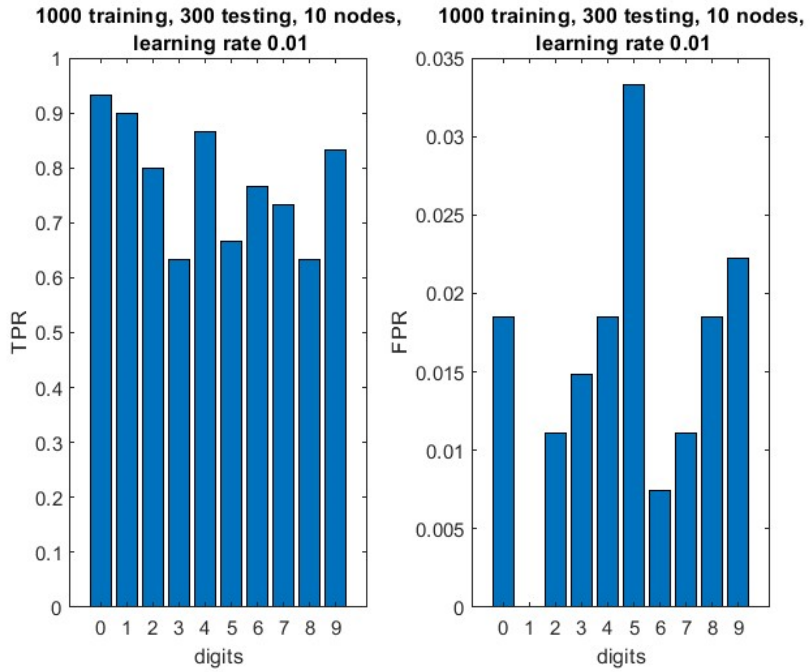

Figure 2: Plots - 1000 training data, 300 testing data, learning rate = 0.01, hidden nodes = 10, momentum = 0.05

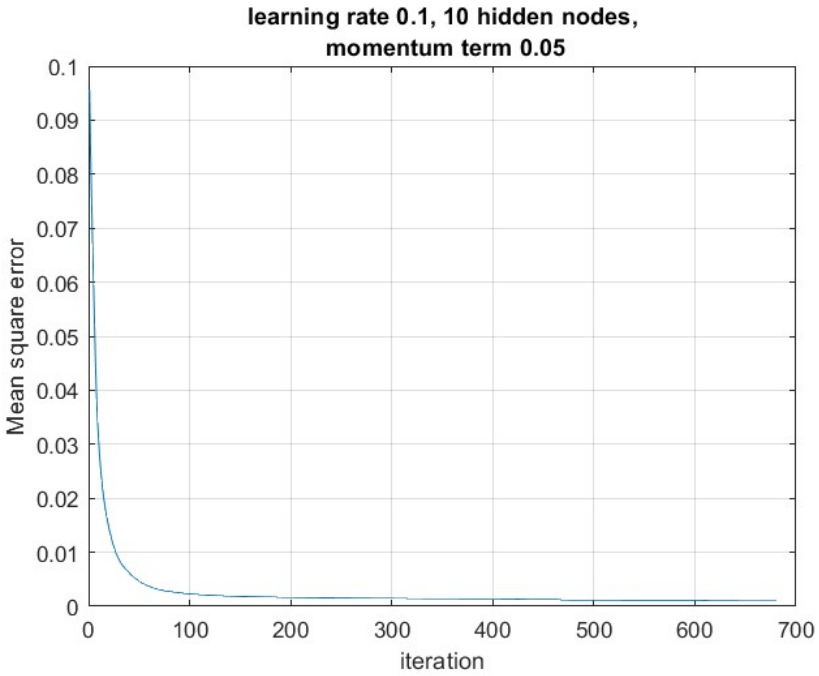

Figure 3: Error - 1000 training data, 300 testing data, learning rate = 0.1, hidden nodes = 10, momentum = 0.05

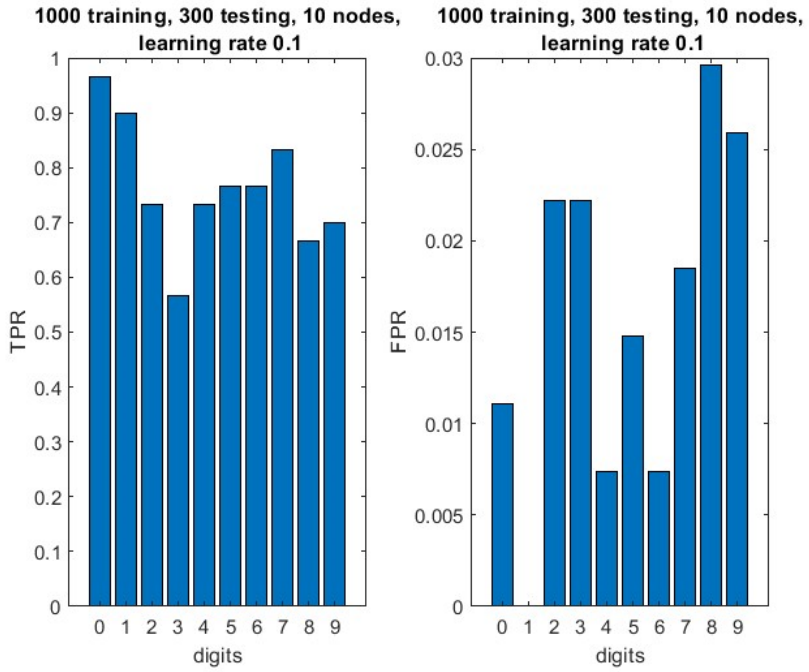

Figure 4: Plots - 1000 training data, 300 testing data, learning rate = 0.1, hidden nodes = 10, momentum = 0.05

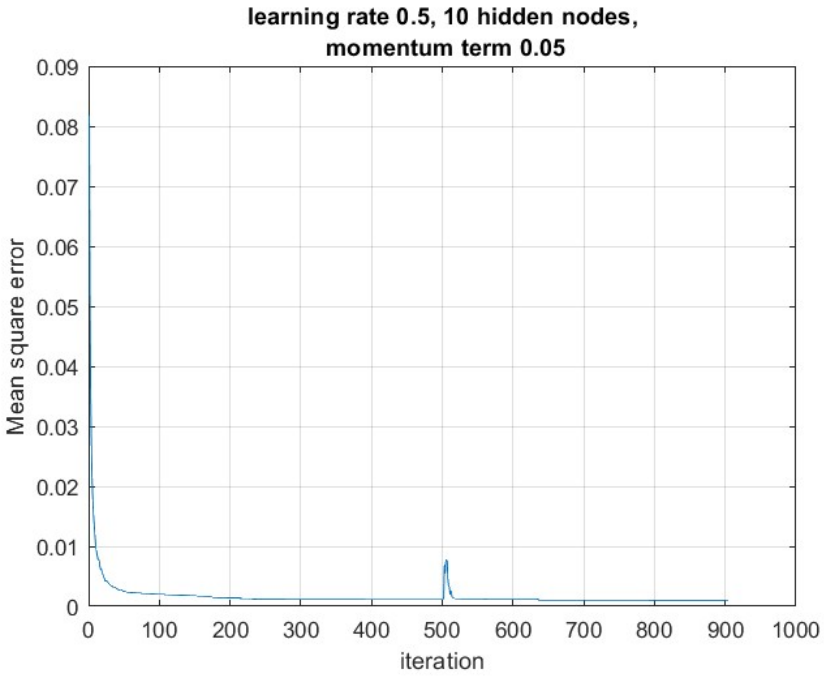

Figure 5: 1000 training data, 300 testing data, learning rate = 0.5, hidden nodes = 10, momentum = 0.05

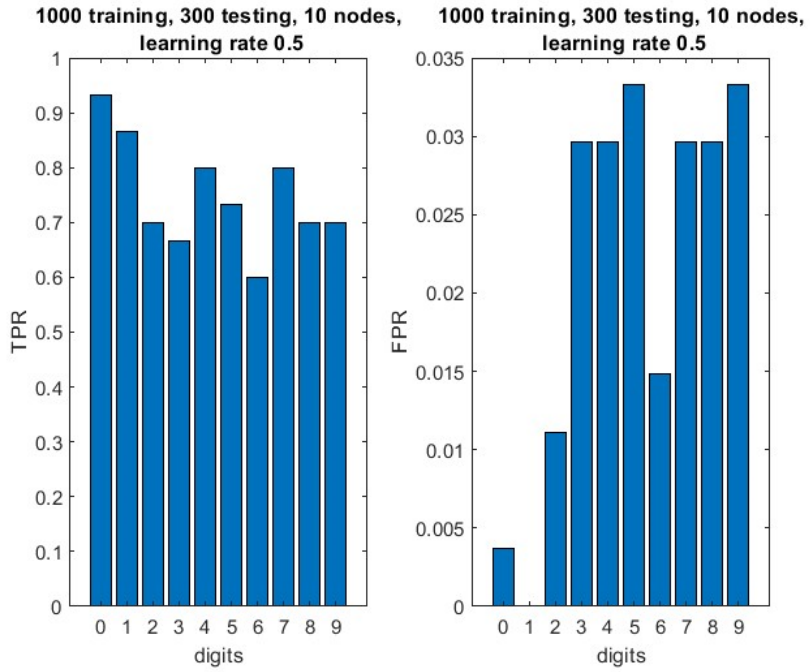

Figure 6: 1000 training data, 300 testing data, learning rate = 0.5, hidden nodes = 10, momentum = 0.05

II. Task 4

Methodology

The methodology is still the same. Only the number of nodes in the hidden layer has changed.

Results

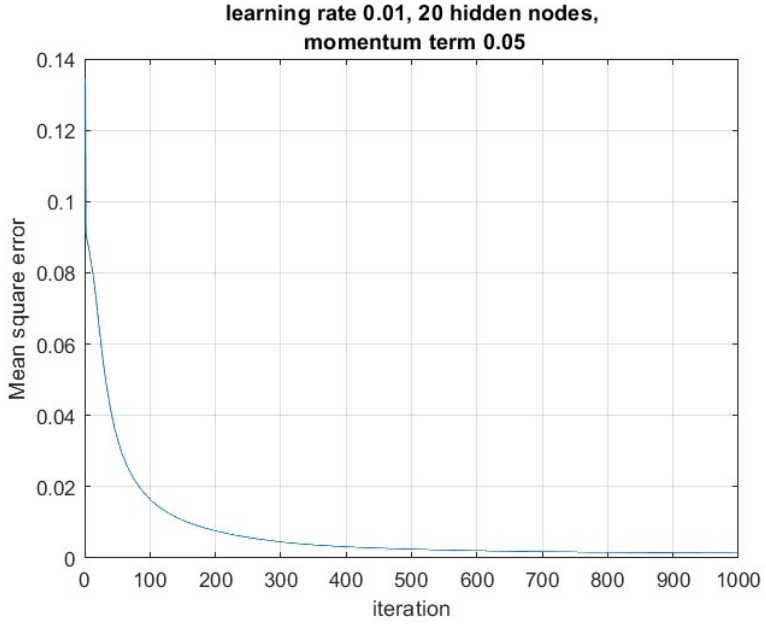

Figure 7: Error - 1000 training data, 300 testing data, learning rate = 0.01, hidden nodes = 20, momentum = 0.05

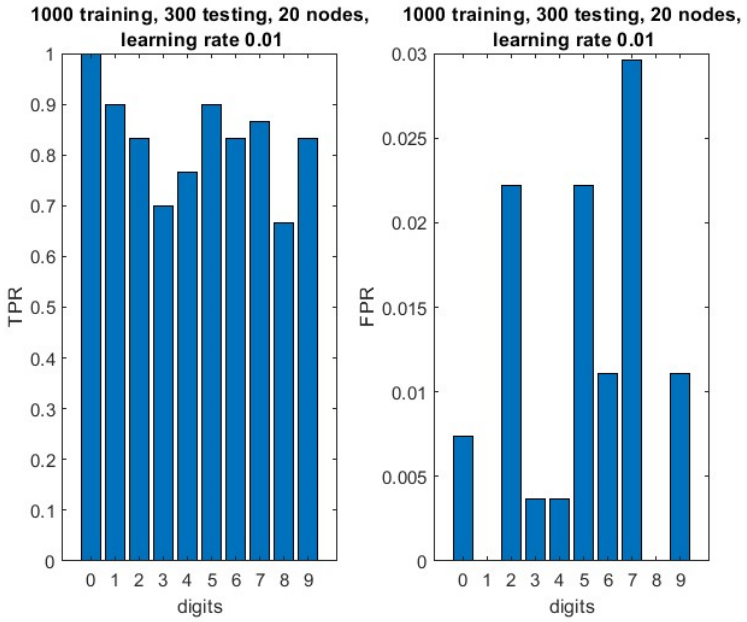

Figure 8: Plots - 1000 training data, 300 testing data, learning rate = 0.01, hidden nodes = 20, momentum = 0.05

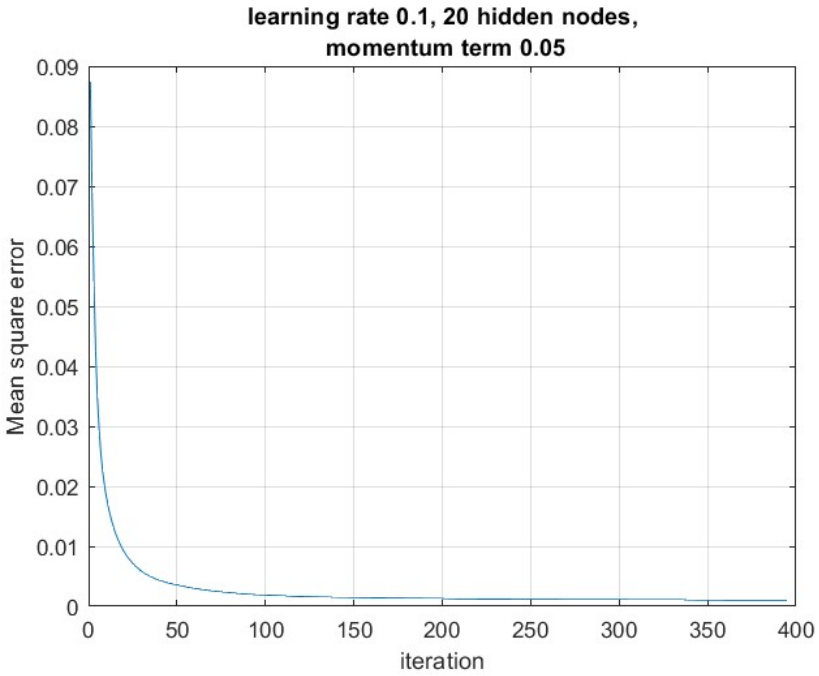

Figure 9: Error - 1000 training data, 300 testing data, learning rate = 0.1, hidden nodes = 20, momentum = 0.05

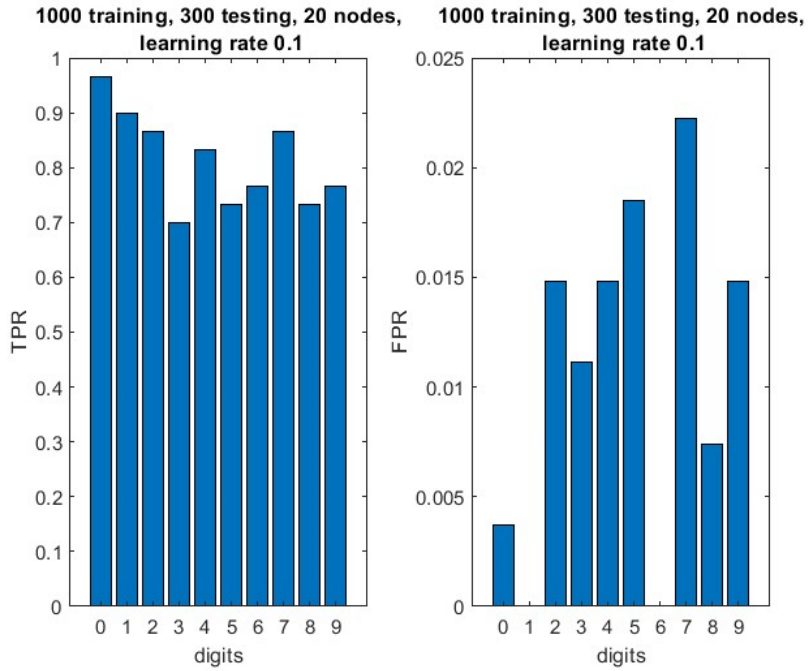

Figure 10: Plots - 1000 training data, 300 testing data, learning rate = 0.1, hidden nodes = 20, momentum = 0.05

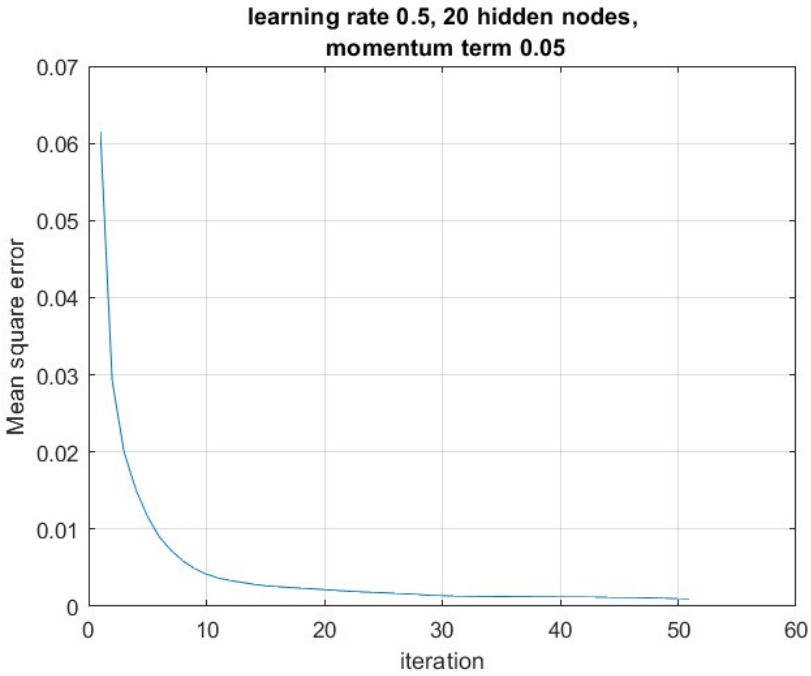

Figure 11: Error - 1000 training data, 300 testing data, learning rate = 0.5, hidden nodes = 20, momentum = 0.05

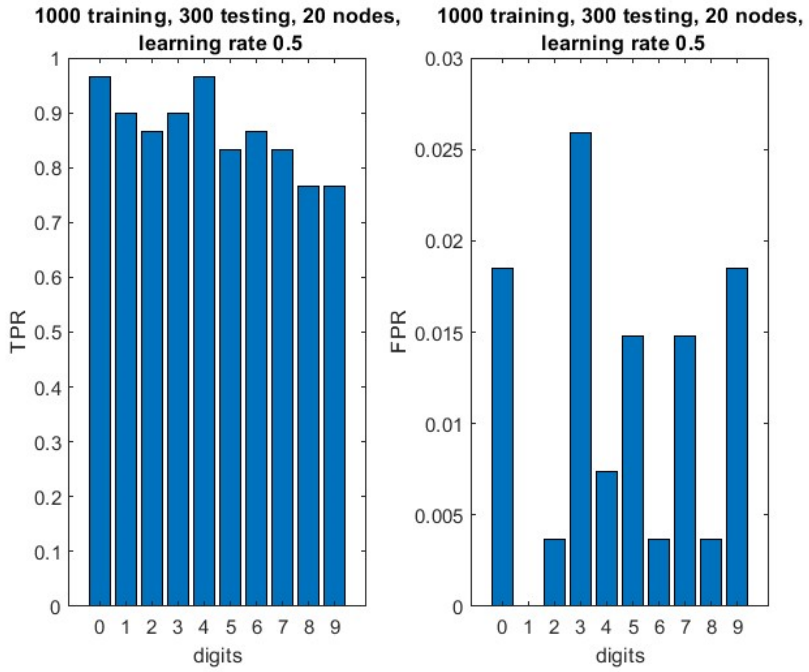

Figure 12: Plots - 1000 training data, 300 testing data, learning rate = 0.5, hidden nodes = 20, momentum = 0.05

III. Conclusion/Comments

The goal of this assignment was met, and a multi-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.

- Different sets of weights were computed for the same dataset with different parameters (learning rate, weights initialization, momentum term, number of nodes in the hidden layer). This is reasonable since one parameter greatly affects the learning process.

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

- The Error plot has some bumps in some cases. This demonstrates that the error will occasionally increase slightly before reaching convergence.

- 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.

- When comparing the 10-hidden nodes layer to the 20-hidden nodes layer, the error converged faster in the 20-hidden layer, and the false positive rate for each digit was lower overall. In addition, the true positive rate for each digit was higher overall. Thus, for some extent, a high enough number of nodes in the hidden layer might do a great job in the learning process.

- For more complex datasets, more hidden layers could be built, and both forward and backward passes would remain the same.