# Intelligent Systems

# Homework 4

***Problem 1***

**System Description:**

Learning Rate – The learning rate ranges between 0 and 1. For this code, the learning rate taken is 0.02 as this is an ideal value. Too large a learning rate could cause the weights to switch back and forth from negative to positive and not give optimum results. Too low a learning rate would make the neural network learn too slow.

Momentum – The term alpha in the code is applied to in the equation of delta weights, which is 0.2, and updates the weights. It took a little longer for the code to run with these values of learning rate and momentum.

Dropout Rate: The probability at which the weights are chosen from input to hidden weights is taken as 0.5. This is a safe probability as a subset of the 785 x 200 (input neurons with bias weights x hidden neurons) weights are taken at random.

Output threshold – There are 10 output thresholds or output neurons as we have a range of 0 to 9 of numbers, i.e., range of 10 possible output values.

Rule for choosing initial weights – The initial weights are chosen at random between values of -0.2 and +0.2. These values will be good to begin the learning from.

Criterion for deciding when to stop training – Train dataset is 4000 out of 5000 data points. The training is only done for 3500 randomized data points out of 4000 as it is mentioned in the question that a subset of the training dataset should be used. Now, the weights from input to hidden layer are updated at a random probability of 0.5, and all the weights from hidden to output layer are updated at each iteration.

Number of hidden layers – The number of hidden layers chosen in the neural network is 1. This number is chosen so that the process of updating weights between the layers remain transparent and any discrepancy or error can be identified easily, i.e., less debugging time.

**Results:**

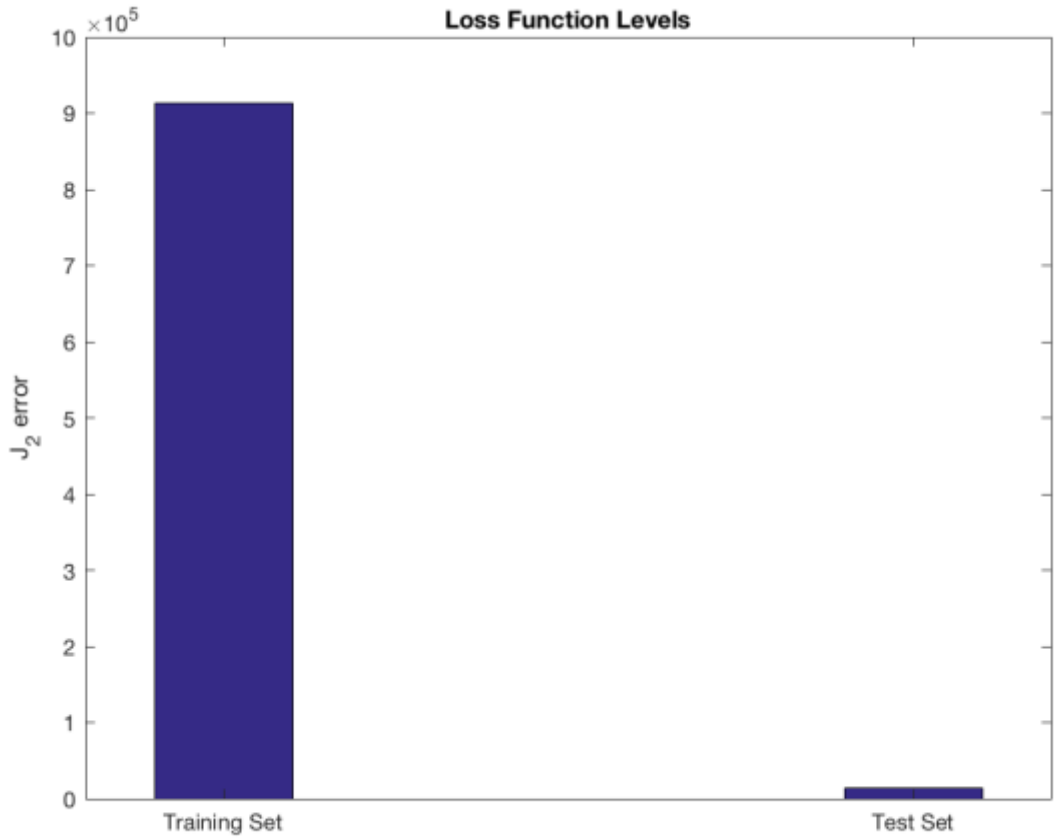


*Figure 1: Loss Function Levels for Training and Testing Sets*

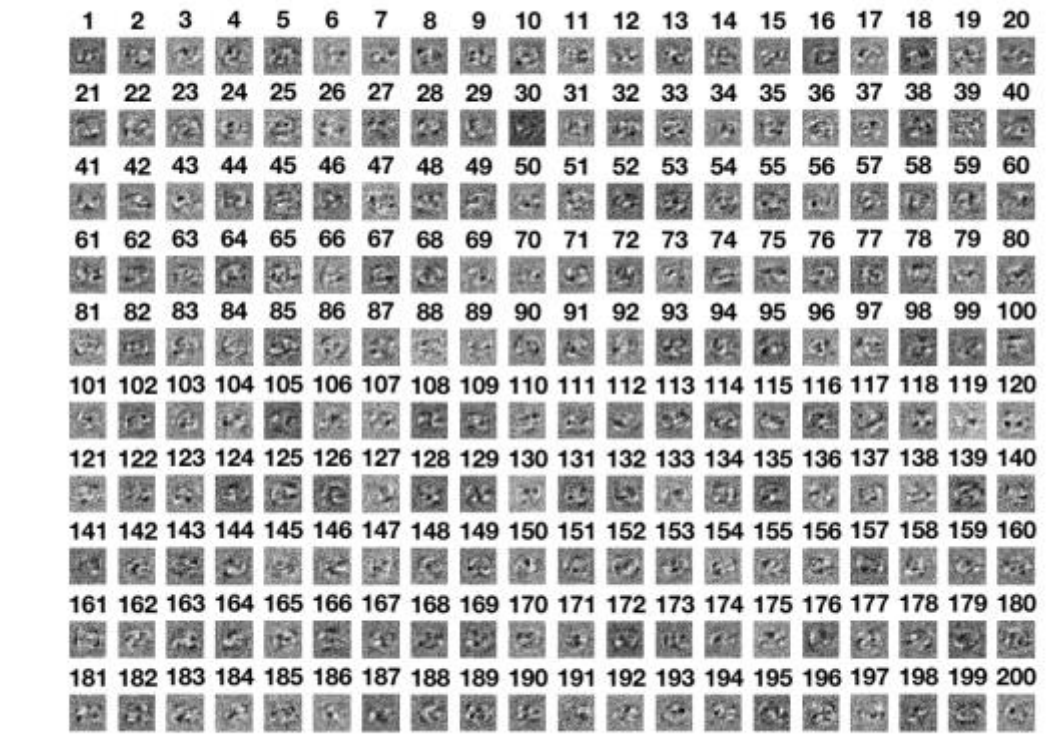


*Figure 2: Weights between input and hidden layer neurons*

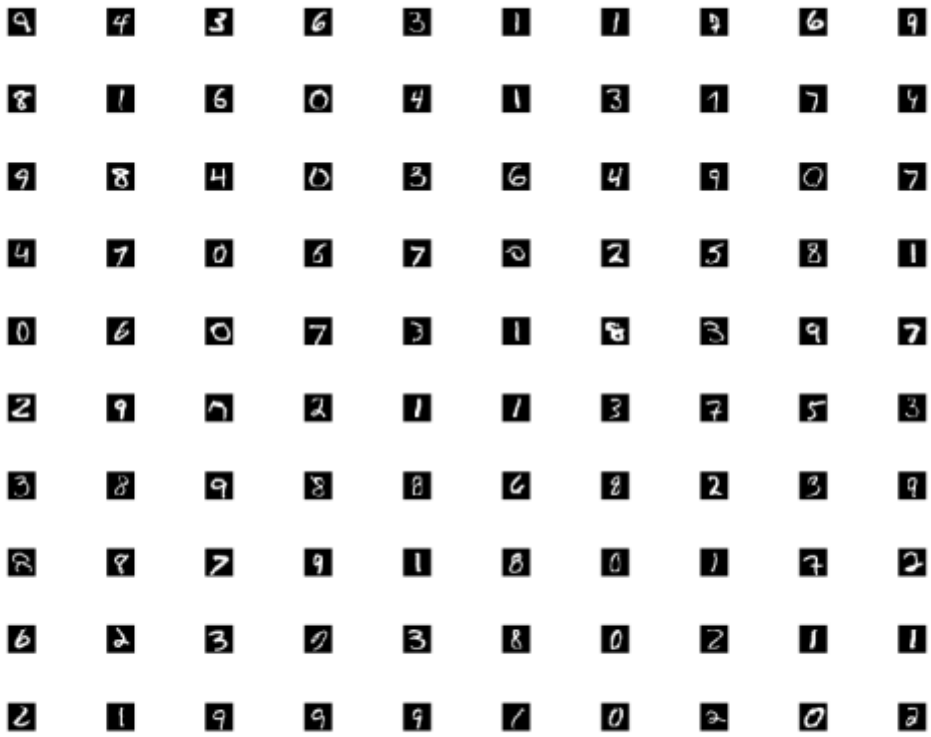

*Figure 3: Images of Input (Original Data)*

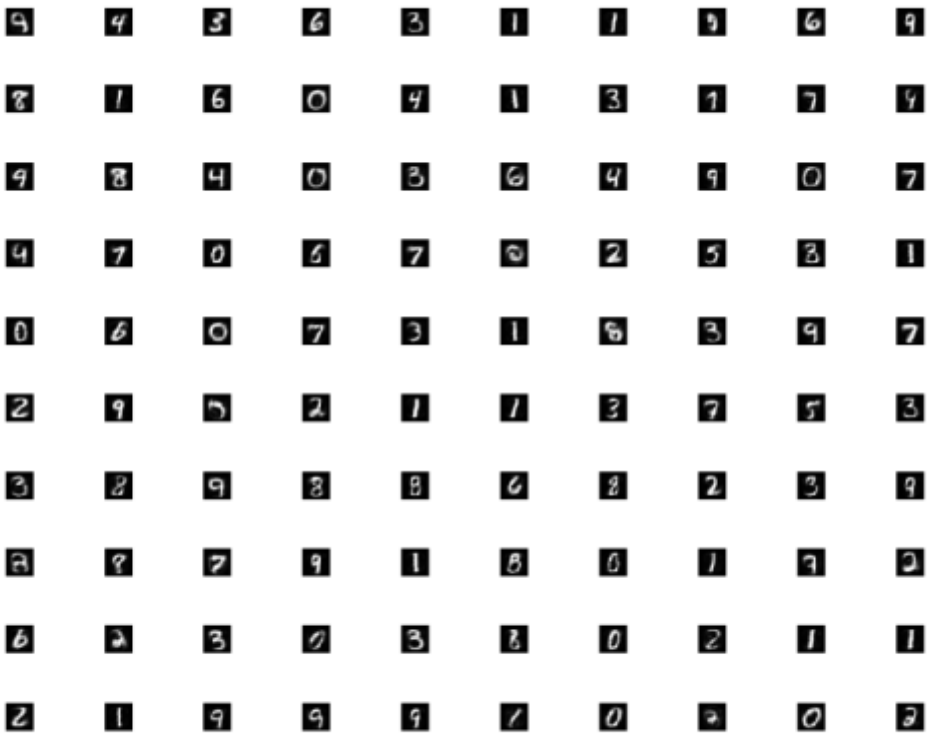

*Figure 4: Images of Calculated Images*

**Analysis of Results:**

Figure 1: Loss Function Levels

Figure 2: Weights between input layer and hidden layer neurons

Figure 3: Plotting 1st 100 original input images

Figure 4: Plotting 1st 100 calculated images

***Description:***

For the auto-encoder problem, entire 4000 points of the training data. The number of hidden neurons in the hidden layer are taken as 200. This code runs over the entire training data for a series of epochs. Training is done until J2 function goes below 25000 (observed low value from HW3 Problem 1, sufficiently low). For the previous problem, we had, k (inputs) = 784, j (hidden layer) = 200 and i (output) = 10. For this problem, we have k (inputs) = 784, j (hidden layer) = 200 and i (output) = 784. Ran the code for one epoch on the test dataset. Figure 1 shows the J2 loss function calculated for the training dataset for over 200 epochs, and J2 loss function calculated on the test set for one run. The loss function observed is high for the training data but the loss is barely anything for the test data. The network is trained using back-propagation using momentum. At the hidden layer, 100 hidden neurons can be used instead of 200 too and this will give a very similar result. If 1000 neurons are used in the hidden layer, instead of 200 neurons, the neurons will appear much clearer than it would for 200 as more weights are used to train the same neural network, hence giving a clear intermediate hidden layer image. Next, Figure 2 is a plot of the weights between the input and hidden layer. Here, the transformation can be observed very clearly. Figure 3 is the output of the original dataset that is given to us. This is just a plot of how the desired output should look like. Figure 4 is the input; the hidden neuron is most responsive for. Here, the calculated images are not as distinct as they were for HW3 Problem 2 because only half the weights are getting updated from input to hidden layer as probability of randomly chosen weights is taken as 0.5. This directly affects the distinctiveness characteristic and the features as observed of the images. In this problem, the weights from input to hidden layer are chosen at a random probability of 0.5 as seen in the code.

***Problem 2***

**System Description:**

Learning Rate – The learning rate ranges between 0 and 1. For this code, the learning rate is taken higher than the other problems of Hw3 and HW4. It is 0.2, otherwise the execution time would take too long. Too large a learning rate could cause the weights to switch back and forth from negative to positive and not give optimum results. Too low a learning rate would make the neural network learn too slow.

Momentum – The term alpha in the code is applied to in the equation of delta weights, which is 0.5, and updates the weights. This value taken here is higher than done in the other problems to reduce the execution time of the code.

Output threshold – There are 10 output thresholds or output neurons as we have a range of 0 to 9 of numbers, i.e., range of 10 possible output values.

Rule for choosing initial weights – Here, the initial weights between the input and hidden layer are taken from the last iteration obtained from the final network in HW3 problem 2. The weights between the hidden and output layer are initialized randomly between -0.2 and +0.2.

Criterion for deciding when to stop training – Train dataset is 4000 out of 5000 data points. The training is only done for 3500 randomized data points out of 4000 as it is mentioned in the question that a subset of the training dataset should be used.

Number of hidden layers – The number of hidden layers chosen in the neural network is 1. This number is chosen so that the process of updating weights between the layers remain transparent and any discrepancy or error can be identified easily, i.e., less debugging time.

**Results:**

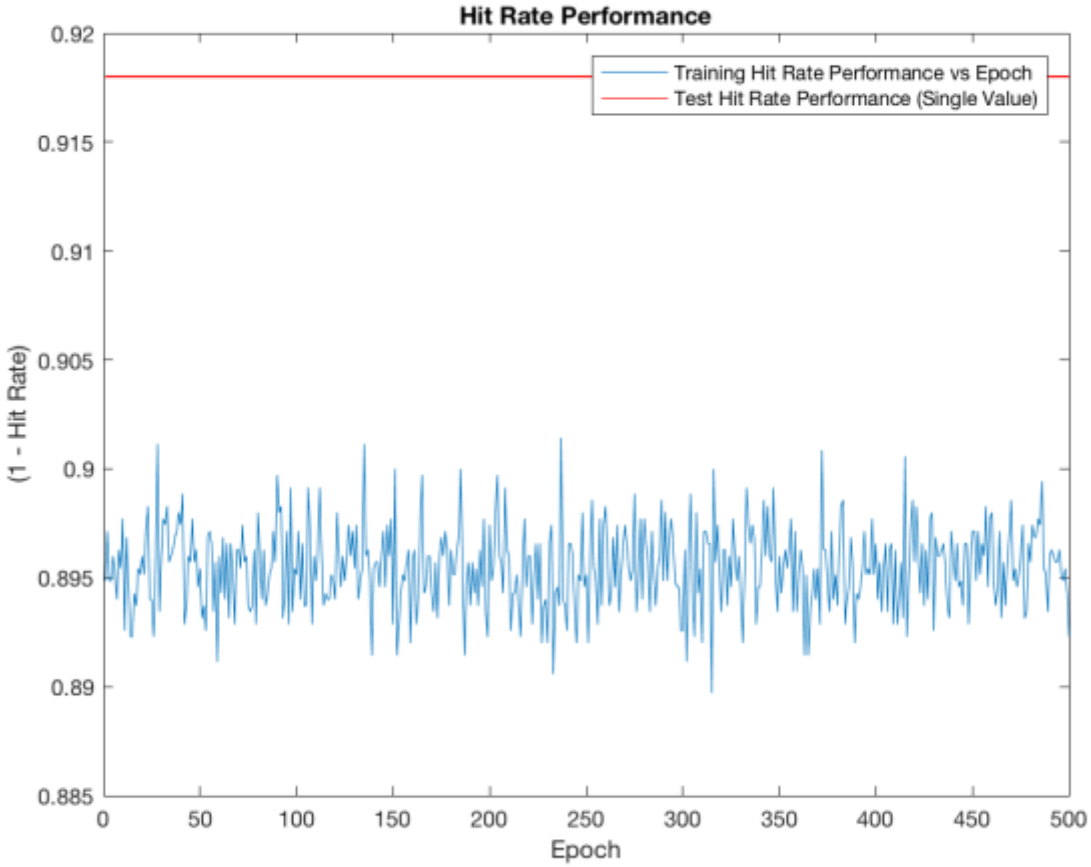


*Figure 5: Hit Rate Performance for Training and Testing Data for Network A*

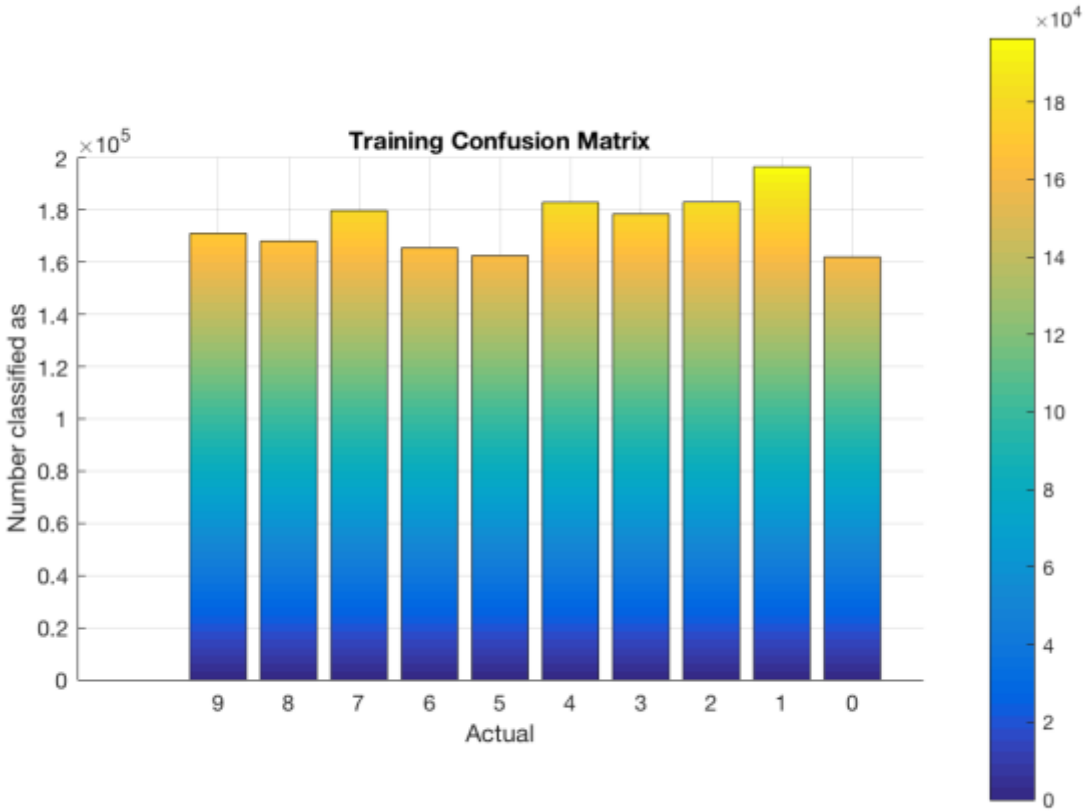


*Figure 6: Training Confusion Matrix for Network A*

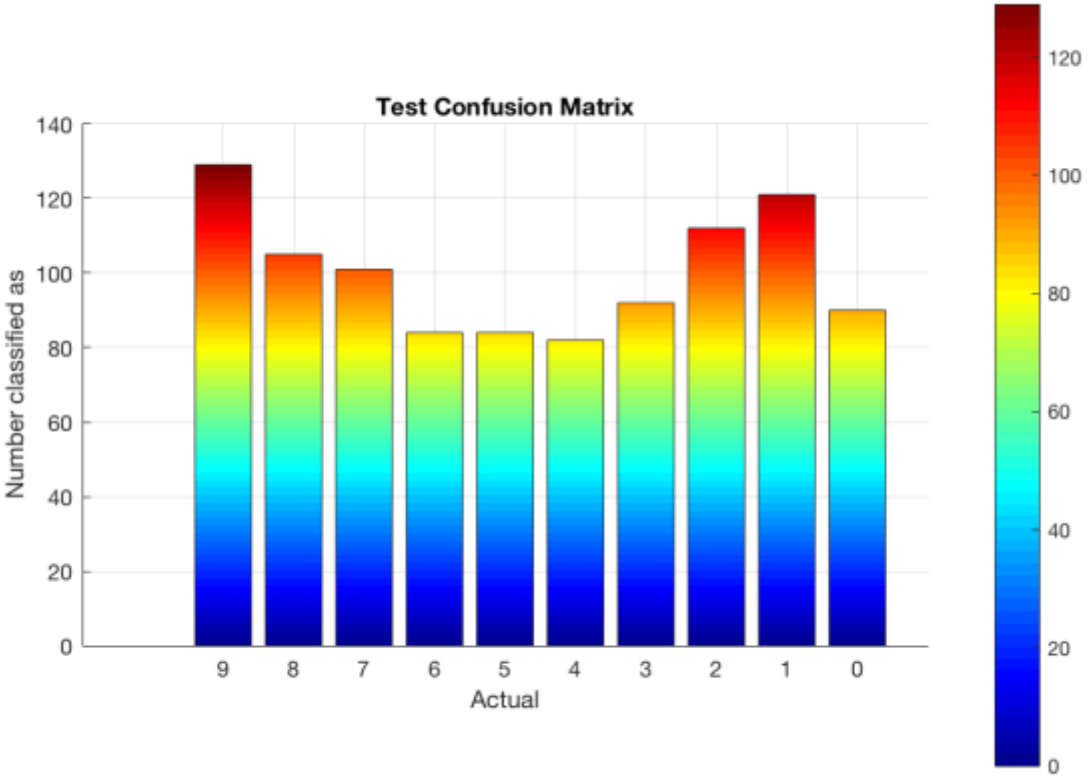


*Figure 7: Test Confusion Matrix for Network A*

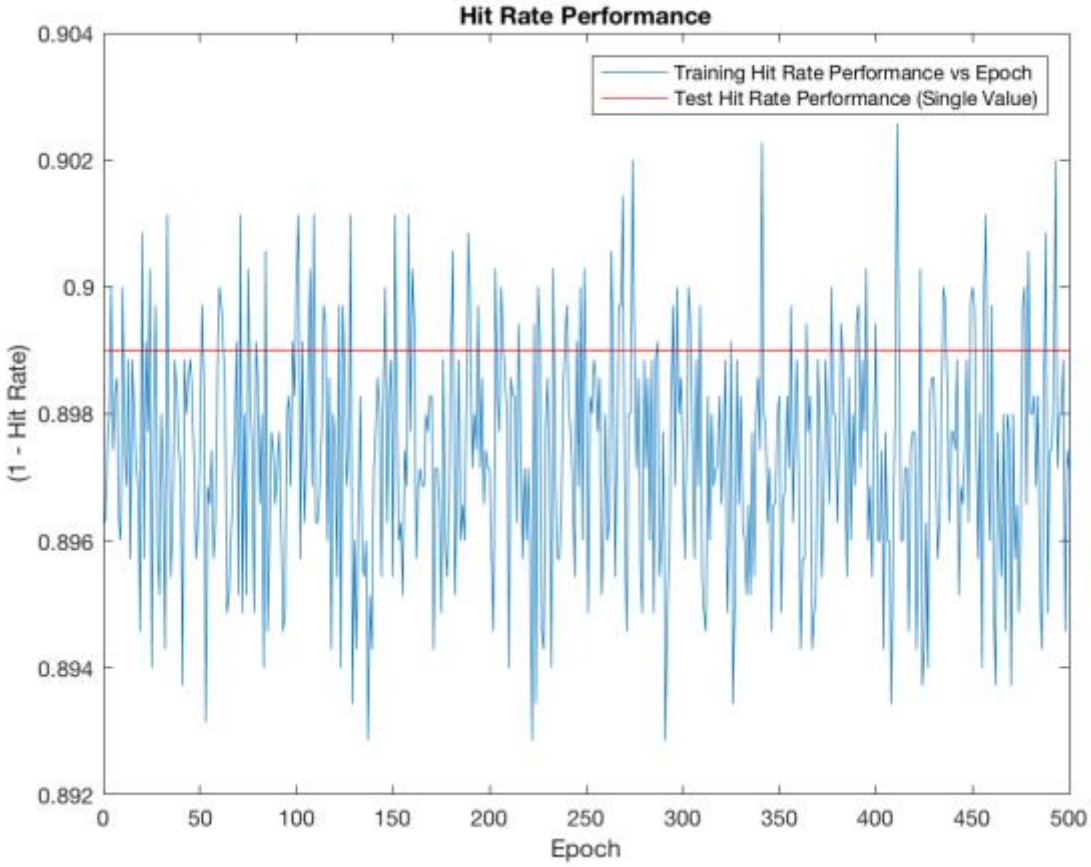


*Figure 8: Hit Rate Performance for Training and Testing Data for Network B*

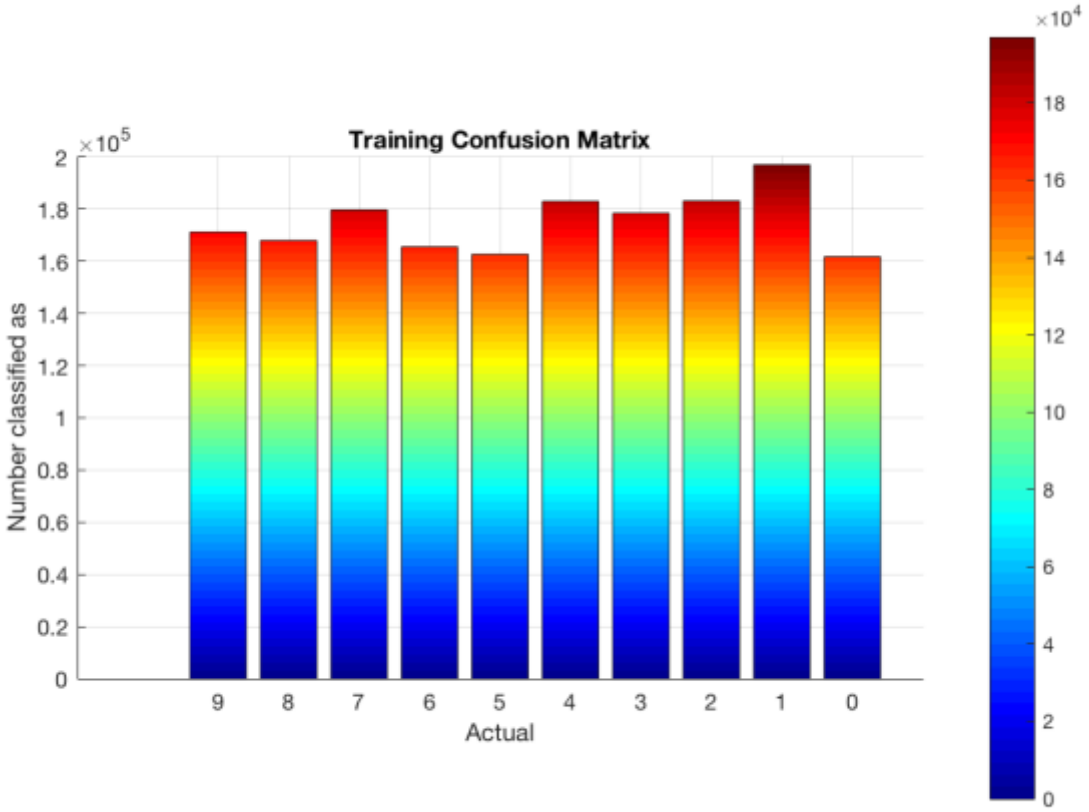


*Figure 9: Training Confusion Matrix for Network B*

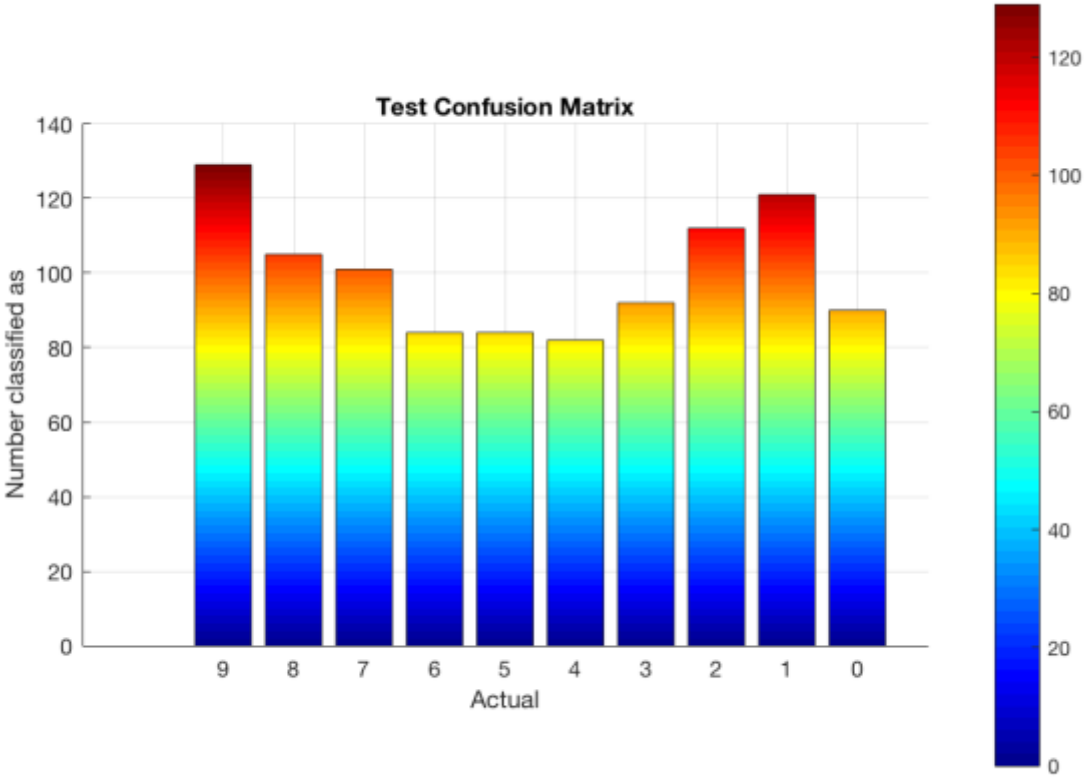


*Figure 10: Test Confusion Matrix for Network B*

**Analysis of Results:**

Figure 5: Hit Rate Performance for Training and Test Data for Network A

Figure 6: Training Confusion Matrix for Network A

Figure 7: Test Confusion Matrix for Network A

Figure 8: Hit Rate Performance for Training and Test Data for Network B

Figure 9: Training Confusion Matrix for Network B

Figure 10: Test Confusion Matrix for Network B

***Description:***

I divided the data into train (4000 data points) and test (1000 data points). The initial weights between input and hidden layer are taken from the final network obtained in HW3 problem 2. The weights between hidden and output layer are initialized between negative 0.1 and positive 0.1 randomly with learning rate 0.2 and momentum 0.5, with bias weights at input and hidden layers (hence, w(ij) is a 200x785 and w(jk) is 10x201). The learning rate and momentum are increased as compared to the other problems as this increases the execution time of the code and the weights can learn faster, and hence give better results in less number of iterations of epochs. Ran the feed-forward and back-propagation on train data for 500 epochs so that the weights would learn to give a good hit-performance. The training is done a subset (3500 out of 4000) of training data points. In the feed-forward mechanism, calculated s and output (h) for hidden and output layers. The hit-rate performance train is incremented if output calculated is equal to the desired output. In the back-propagation algorithm, the delta weights are calculated using momentum term, which gives optimum learning while updating weights for hidden and output layers. For the test data set, it is run for the 1000 test data points for one epoch. The confusion matrix for train and test data is a 10x10 matrix such that the diagonal elements are the values which are found correctly with the algorithm and the remaining upper and lower diagonal elements are the errors just like mentioned in the problem statement.

In Network A, the confusion matrix for training data is better than that for test data in terms of accuracy of the hit rates. This can be observed from figures 6 and 7. Next, from figure 5, it is observed that the training data is not consistent as it learns. The hit performances keep varying with each epoch. It still does have a good hit rate performance, just that it's not consistent. The hit performance of the test data is between 91.5% and 92%, which is great.

In Network B, the confusion matrix for training data is better than that for test data in terms of accuracy of the hit rates. This can be observed from figures 9 and 10. From figure 8, it is observed that the training data is not consistent as it learns. The hit performances keep varying with each epoch. It still does have a good hit rate performance, just that it's not consistent. The hit performance of the test data is between 89.8% and 90%, which is great too.