Intelligent Systems

Homework 3

Problem 1

System Description:

Number of hidden neurons – The number of hidden neurons for one hidden layer will be a number between the number of input neurons and output neurons. The chosen number is 200 neurons as this is a fairly large number of neurons to provide a balanced number of weights between the input and the hidden layer, and the hidden and the output layer.

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 the term that updates the weights.

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.

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. K

Results:

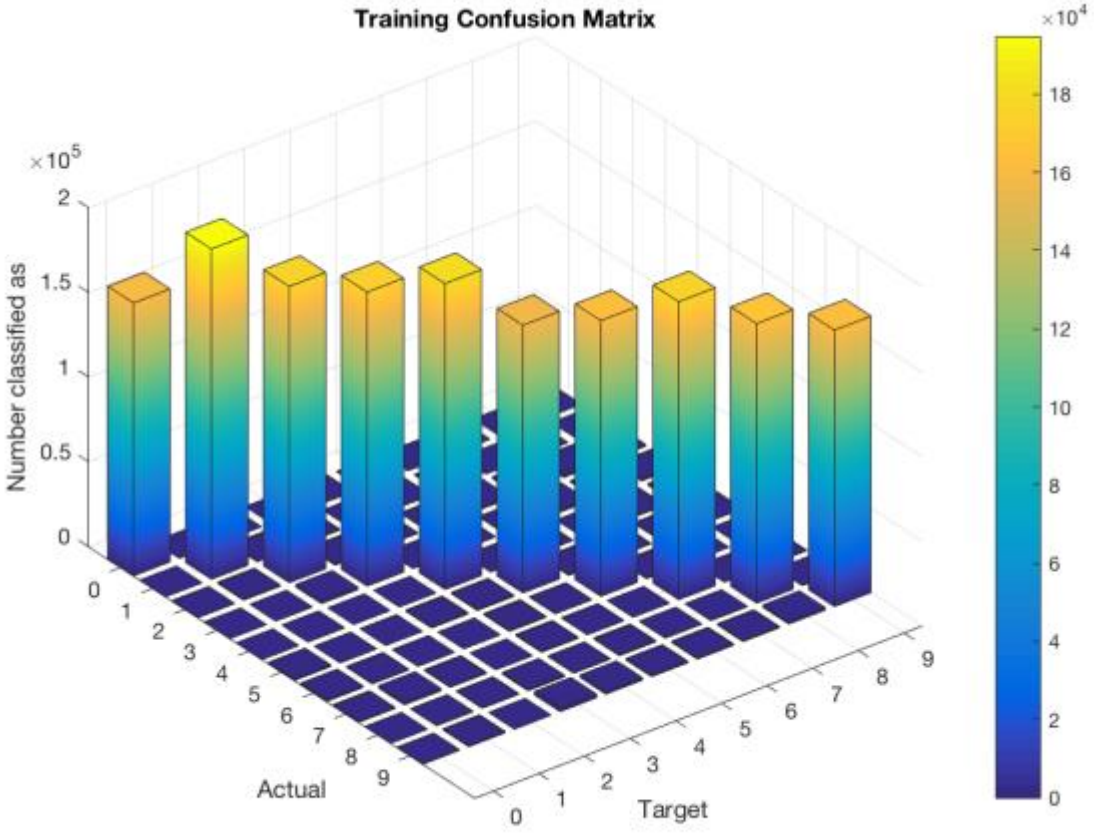

Figure 1: Training Confusion Matrix 3D Plot

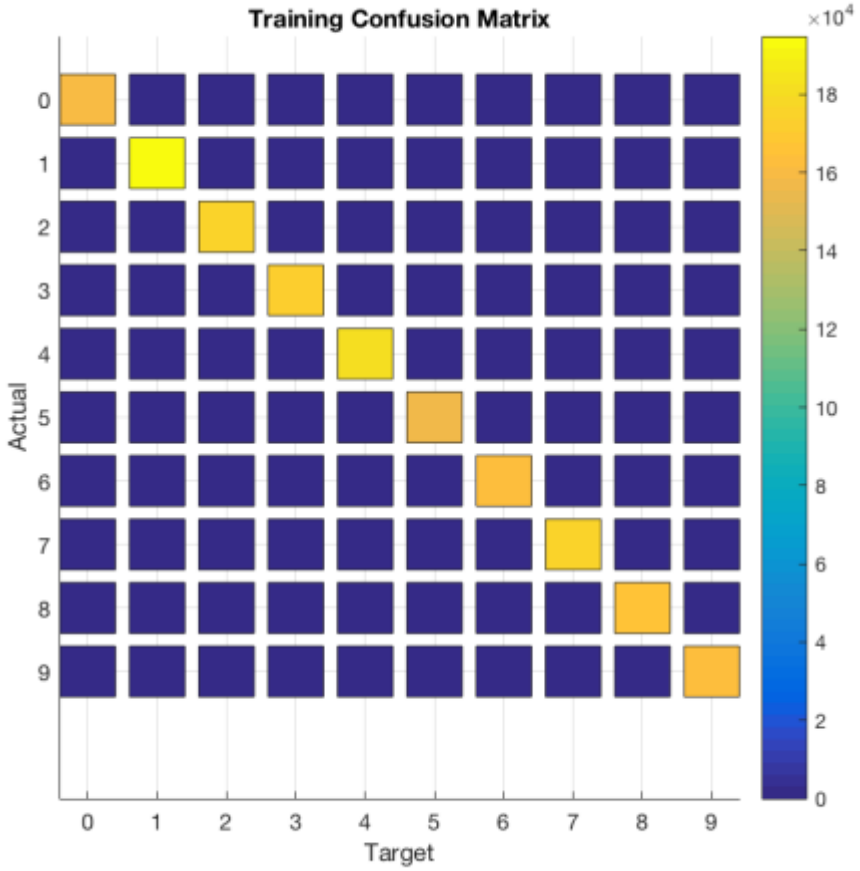

Figure 2: Training Confusion Matrix (Top View)

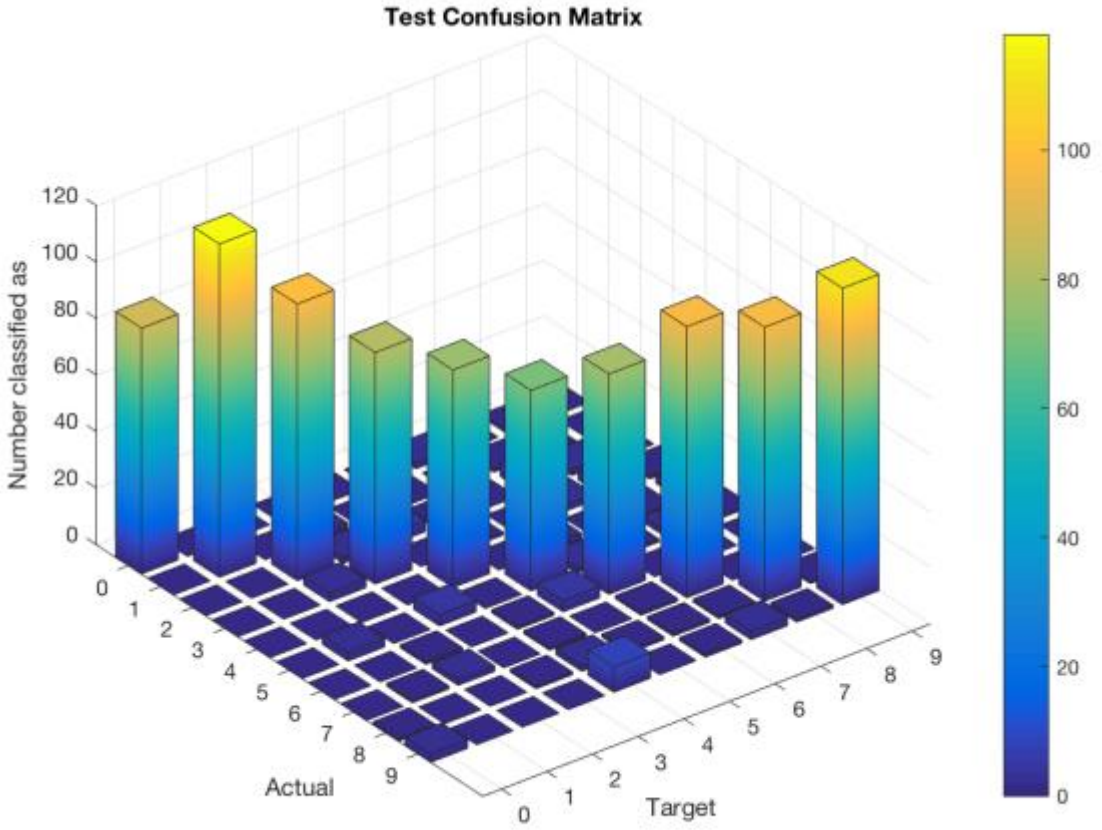

Figure 3: Test Confusion Matrix 3D Plot

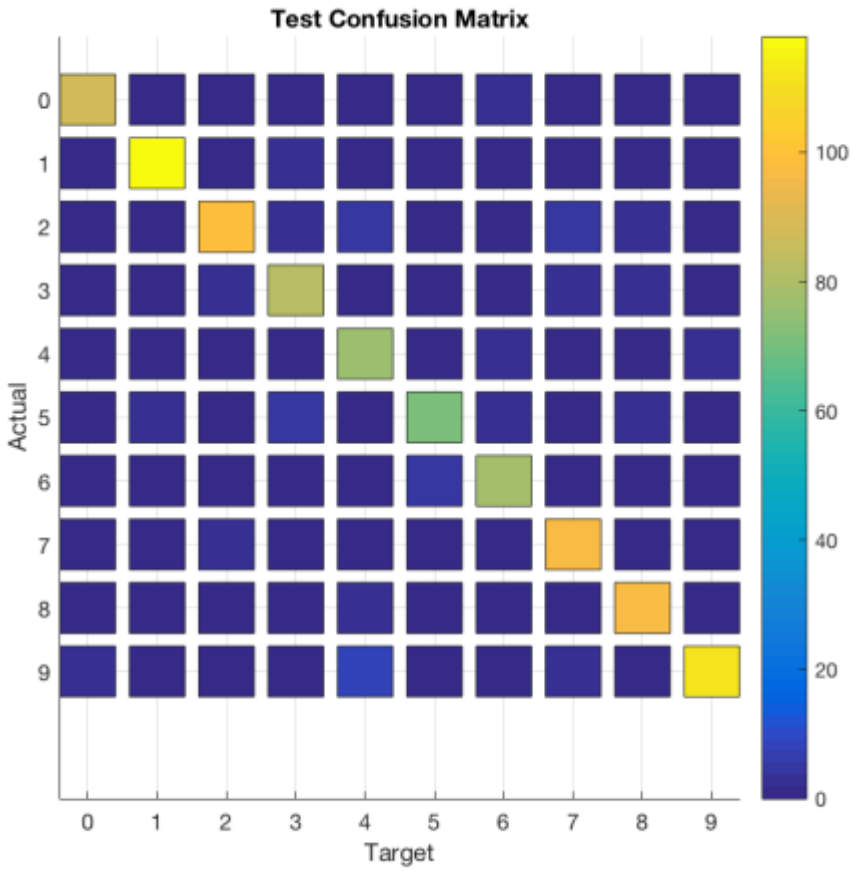

Figure 4: Test Confusion Matrix (Top View)

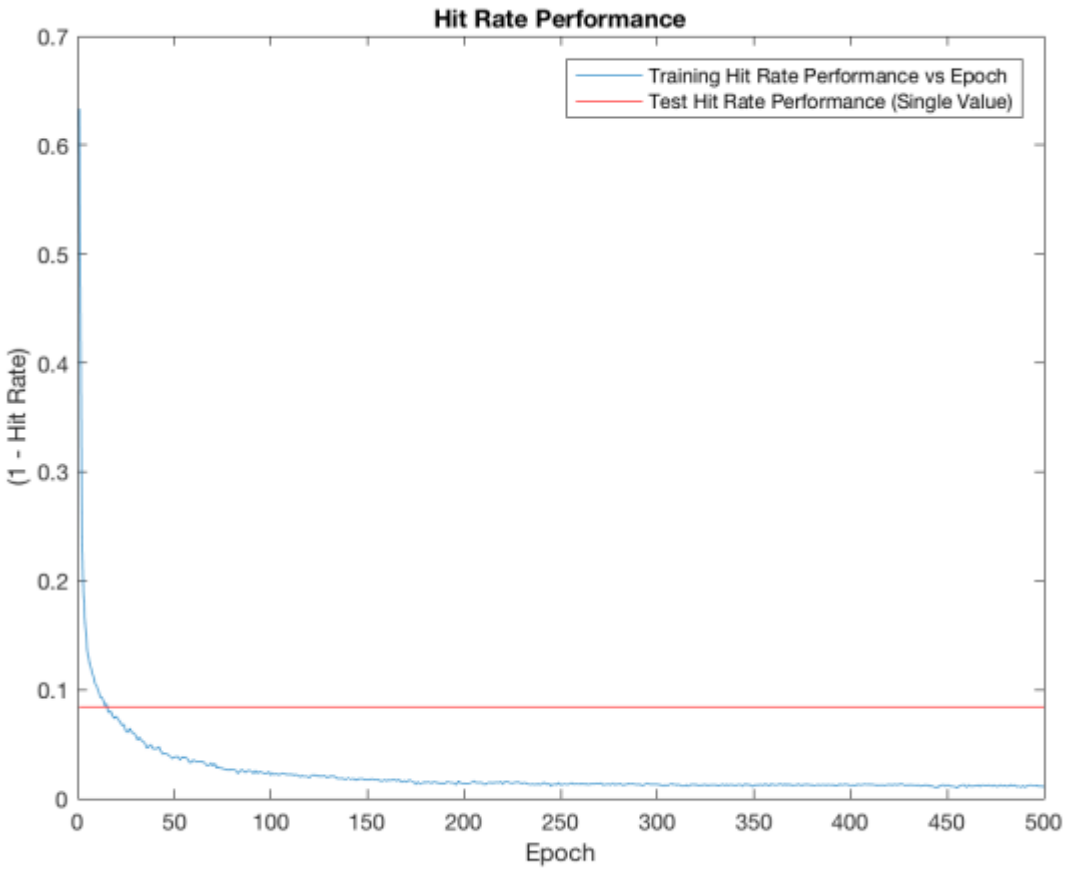

Figure 5: Hit Rate Performance

Analysis of Results:

Figure 1: Training Confusion Matrix 3D Plot

Figure 2: Training Confusion Matrix (Top View)

Figure 3: Test Confusion Matrix 3D Plot

Figure 4: Test Confusion Matrix (Top View)

Figure 5: Hit Rate Performance

Description:

I divided the data into train (4000 data points) and test (1000 data points). The initial weights are initialized between negative 0.1 and positive 0.1 randomly with learning rate 0.02 and momentum 0.2, with bias weights at input and hidden layers (hence, w(ij) is a 200x785 and w(jk) is 10x201). 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. From figure 1 and 2, it can be observed that the training data learns well with hit-rate performance reaching +99% for 500 epochs. The errors are significantly low (blue elements, i.e., everything except the diagonal elements). From figure 3 and 4, it can be observed that the testing data learns well too but not as good as the training data, because this runs only for one epoch. Still, the hit rate performance reached a good +98%. Figure 5 is a plot of number of epochs versus 1 – hit rate performance for training dataset. It is observed, that the neural network gradually learns from about 30% hit-rate performance for first epoch to about 99% hit rate performance for the 500^{th} epoch. Even when I ran the code for 50 epochs, the learning was good up to 87% hit-rate performance.

Problem 2

System Description:

Learning Rate, Momentum, Rule for choosing initial weights, Output threshold

These remain the same as Problem 1. As mentioned in the problem 2 statement, one hidden layer is considered as per the code from problem 1.

Results:

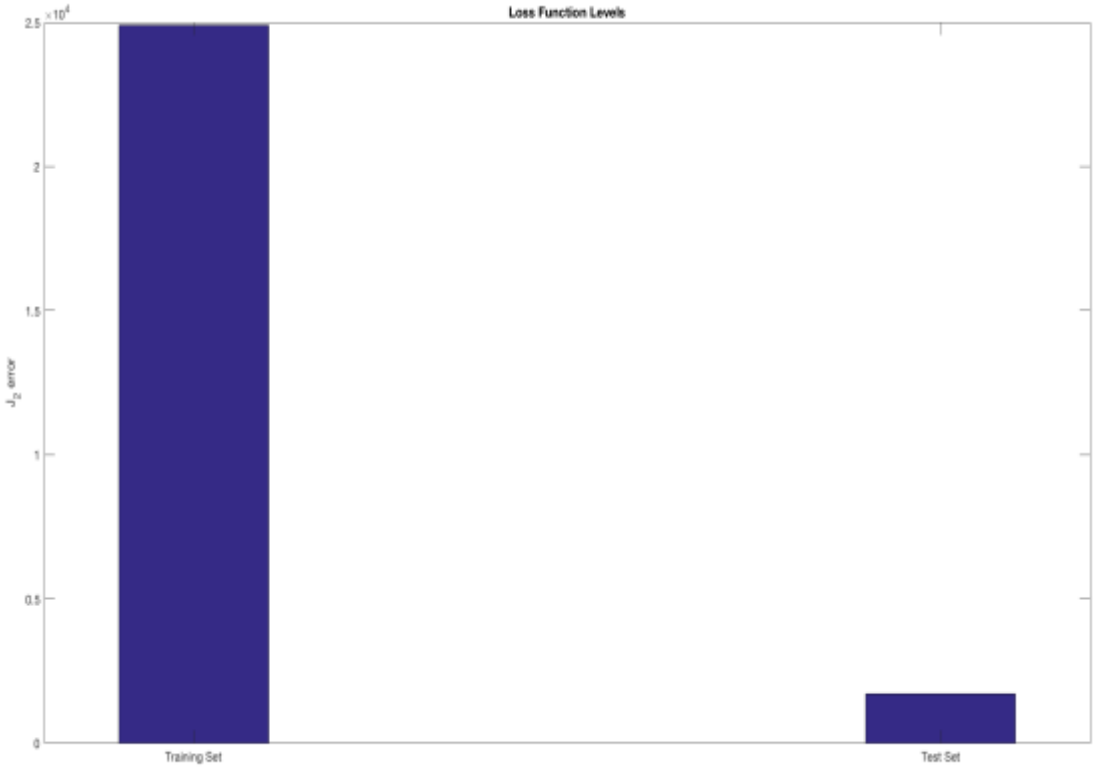

Figure 6: Loss Function Levels

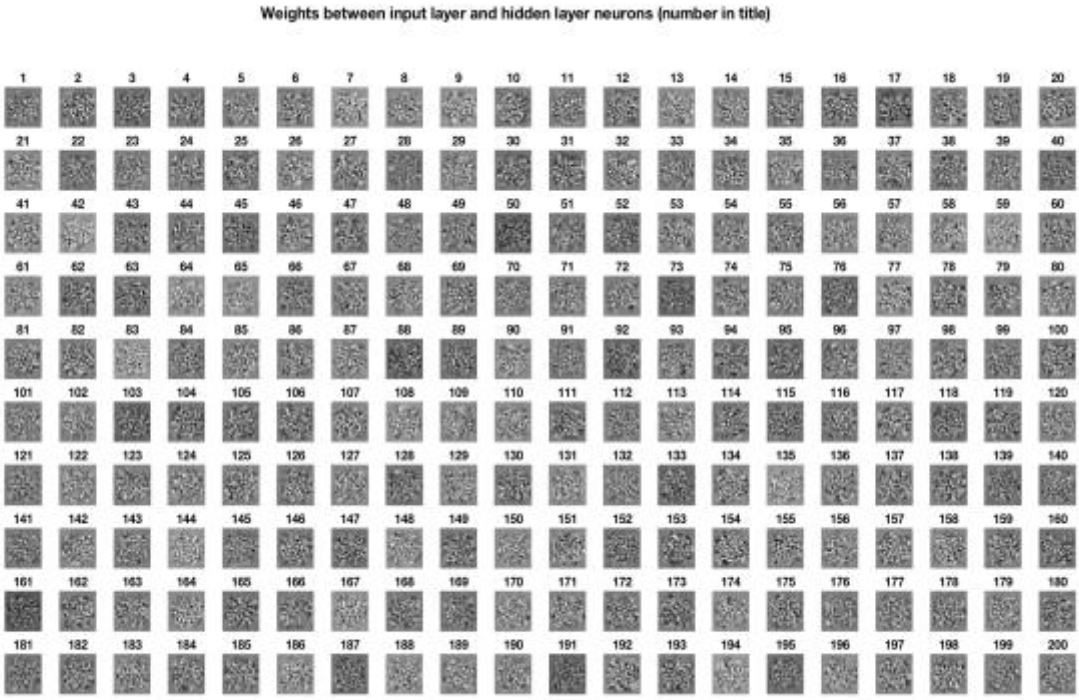

Figure 7: Weights between input and hidden neurons

Features:

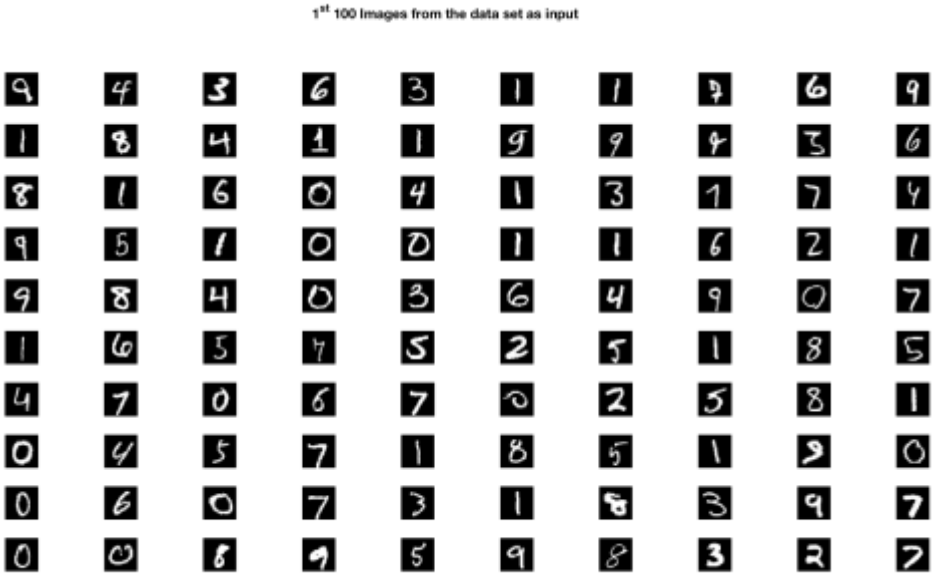

Figure 8: First 100 images from dataset as input

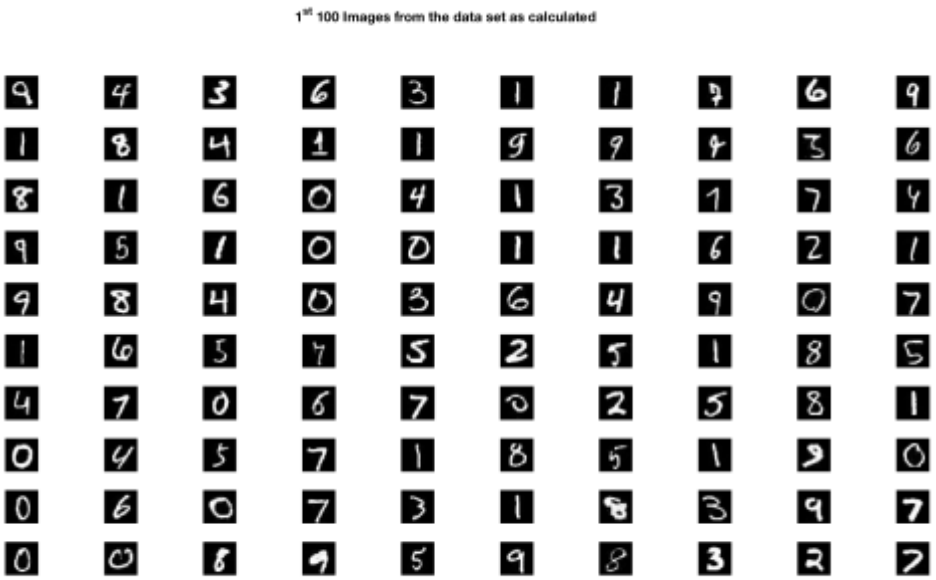

Figure 9: First 100 images from dataset as calculated (output)

Analysis of Results:

Figure 6: Loss Function Levels

Figure 7: Weights between input layer and hidden layer neurons

Figure 8: Plotting 1st 100 input test images

Figure 9: Plotting 1st 100 output test images

Description:

For the auto-encoder problem, I used the 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 Problem 1, sufficiently low). For problem we had, k (inputs) = 784, j (hidden layer) = 200 and i (output) = 10. For problem 2, we have k (inputs) = 784, j (hidden layer) = 200 and i (output) = 784. Ran the code for one epoch on the test dataset. Figure 6 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 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 7 is a plot of the weights between the input and hidden layer. Figure 8 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 9 This shows what input the hidden neuron is most responsive for. The