**AUTO-ENCODER**

**System Description:**

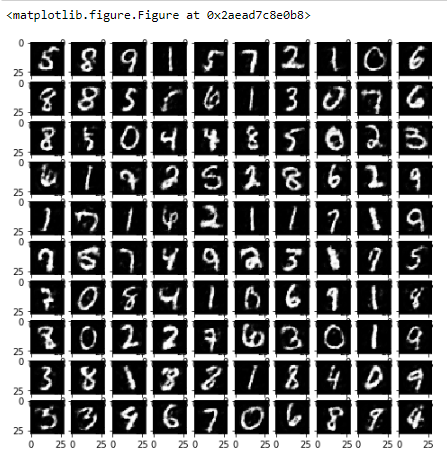
* The system designed here has 1 layer of compression and one layer of decompression. The system that has been designed consist of total 3 layers, the input layer of 784 neurons, the hidden layer of 100 neurons and the output layer of total 784 neurons. The sigmoid function has been applied at the hidden as well as the output layer to enable the function with non linearity.
* For a single hidden layer network, the 2 weight matrices Weight\_1 (784\*100) and Weight\_2 were choosen to be random between 0 and 1 because when tested with all zeros and all ones, all columns in the weight matrices were changing identically which means that there was a symmetry in the system.
* For this network, the initial weight matrices were multiplied with 0.01 as the output from them was becoming large which would put the sigmoid output near 1 and hence no learning was happening.
* After debugging multiple times, the Eta values for both networks were selected to be 0.1 which was suitable as the range in which the sigmoid output should fluctuate was good with these values. There was constant learning as the derivative value didn’t reach zero with these values of Eta.
* This was tried with different values of Momentum. The computation time and the hitrate were optimal for momentum of 0.1 so momentum of 0.1 was taken.
* We used the online method in this and it took around 10 epochs for the cost function to reduce to an average of 2 on each data point for training
* The cost function choosen here is the squared value of each (input-output) and averaging it over all the datapoints.
* The learning became negligible after 10 epochs and hence we stopped the training after 10 epochs. This has been implemented in the code.

**Results:**

After the data is compressed and decompressed again, we could get the following images with a very little loss in the actual data.

The left image is the actual output of data after encoding (Fig 1).

The right image is the image of all the weights coming out of the hidden neurons(Encoded data parameters) (Fig 2).



The following is the image of all the weights that have been set for hidden to output matrix.

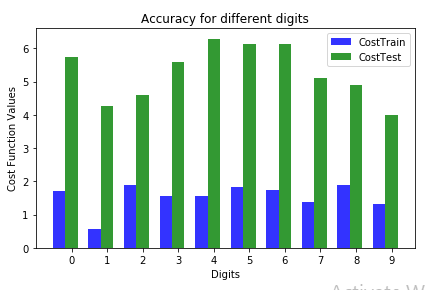
Each block in this matrix of images is a 28 by 28 weights coming out of each hidden neuron.

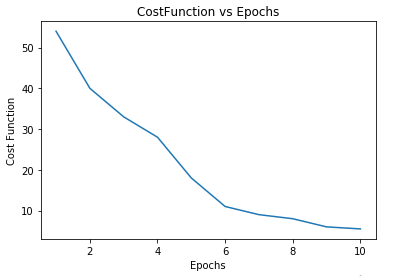
Looking at the following images, we could see that the 1st and 77th neurons are the neurons to which all the digits are sensitive to a large amount.

**Digit wise error distribution**

The following bar graph shows the digit wise distribution of the costfunction over the encoder data. This helps us determine which digits it can encode with less loss of data.

From the below results, we could say that it can encode ‘1’ with less loss of actual data.





**Review:**

* From the above graph, we could say that the cost function decreases with increasing number of epochs. We have used here online method instead of stochastic and hence the number of epochs is less.
* By looking at the bar graph, it could be said that the difference between training and testing costfunction is more and hence the model is little overfitted.
* As the input layer is of 784 neurons and there are 100 hidden neurons, the data could be compressed to about 784:100 times but with a little more loss as compared to traditional encoders.
* This could be implemented where the details of the image are not important and only classifying the data is necessary.

**Appendix:**

1. Below is the code for running a multi layer neural network (autoencoder) which takes input from the user for number of hidden layers and number of neurons in each layer.
2. Before running the code put the path of the Dataset file and Label file in place of “dataset path” and “label path” as highlighted in the code respectively.
3. When the code is run, it will give a prompt asking ‘'Please enter the number of neurons in each layer'’ . Fill the network requirements as ‘100,200,150’. This means the network will have 3 hidden layers, out of which first will have 100, 2nd will have 200 and 3rd will have 150 neurons.
4. The python code works as follows:
   * 1. It imports data file and labels file from the path mentioned as numpy matrices and all initializations of weight matrices, dell(error) matrices, activation of neurons at each level etc. happens.
     2. The feed forward code (activation of each layer) happens with the help of recursion function ‘forwardFirst’ and the cost function is calculated such that it is squared error of output subtracted from the input averaged over all the datapoints.
     3. The weight change calculation and weight change happens with help of recursion function ‘backwardFirst’.
     4. Then the 784 weight matrices from each of the 100 hidden neurons are plotted.
     5. Then the cost function are calculated digitwise for test and train data and plotted as bar graphs.
     6. At last we plot the Cost function of the complete model for training data that we had already captured during training in an array.

**Note:** Mnist digit dataset used for training. If other dataset used then number of input parameters needs to be changed accordingly.