**AUTOENCODER WITH SPARSITY AND WEIGHT DECAY REGULARIZATIONS**

**System Description:**

* We introduce certain constraints on the learning such as sparsity and weight decay for regularizing the learning.
* 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.
* The amount of weight decay (lambda) was defined to be 0.0001 for each batch of size 20 as we could see that values greater than this would fluctuate weight change too much.
* For this network, the initial weight matrices were multiplied with 0.1 as the output from them was becoming large which would put the sigmoid output near 1 and hence no learning was happening (gradient vanishing).
* The value of ‘Beta’ which was sparsity intensity that we apply to our model was selected to be ‘3’ as it gave good sharp features for multiple runs of the program.
* The ‘rho’ value was taken to be ’0.1’ which describes the averaging intensity that we need to apply. This was also referred from the stanford material provided.
* 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 ran this program for batch of 20 for 200 such epochs in each iterations and the number of iterrations were 10. So on average, we ran through each datapoint 10 times.
* 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:**

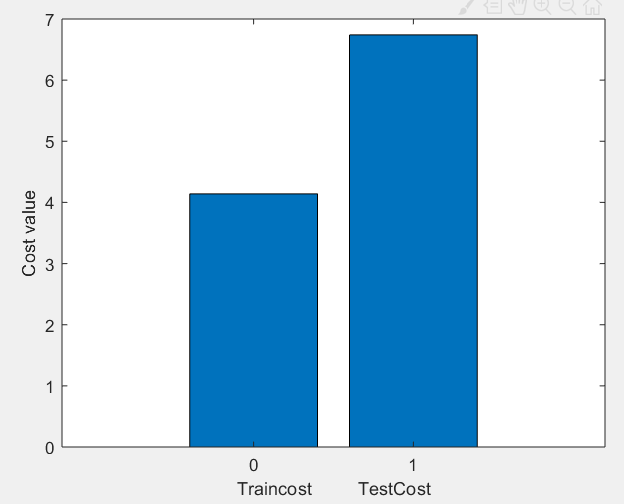


Figure 1-Final cost on training vs Testing Dataset.

Above is the final cost on training and testing datasets. We could see that the difference between the cost of training and testing sets is minimized compared to the network without sparsity and weight decay constraints. Though the test and train costs have been increased slightly than the vanilla network.

Fig.2 below shows the bar-graph for the training and testing data for each digit separately.

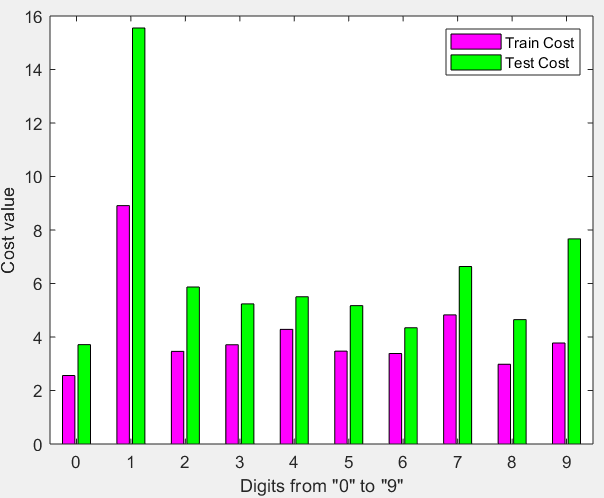


Figure 2-Digit-wise cost value(Train and test)

-Here we could see that the digit on which the error is mostly maximum is ‘1’ which is completely different from the one from without sparsity code where it was very less on 1. Except this, we could see that the cost has reduced compared to vanilla version on each of the other digit. Also there is around difference of 20 percent of train cost, when test and train costs are compared.

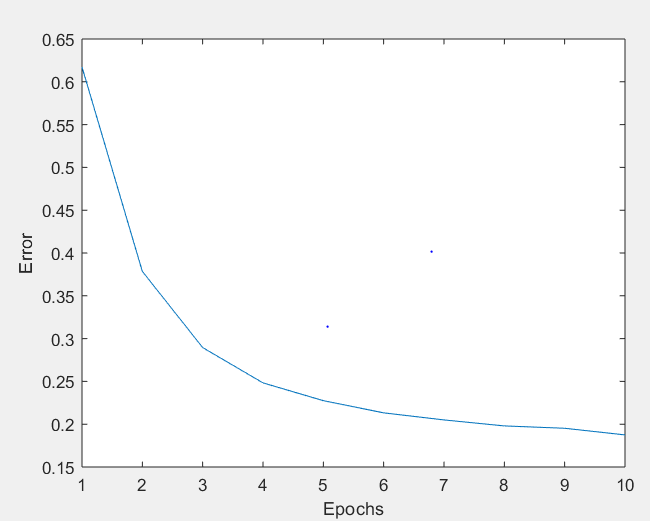


Figure 3-Error vs epoch

-The above graph shows that the error is decreasing speedily at first and the training converges near an error of around 0.17. After 10 epochs it reaches our threshold and the code stops to train further.

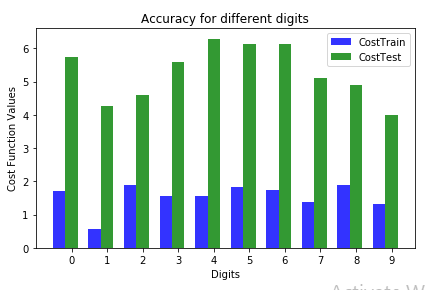
. 

Figure 4-Homework 3-digitwise cost function

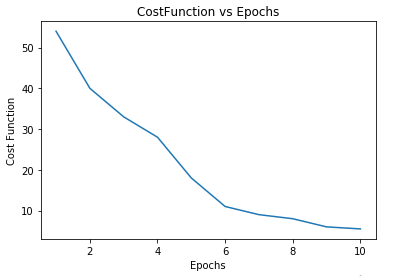


Figure 5-Homework3-Cost function vs Epochs

-Above are the graphs from homework 3 -problem 2 which is the vanilla version of the autoencoder.

-The major difference we could see that is the cost function starts from a huge value without the constraints.



Figure 6-Features with plain Autoencoder

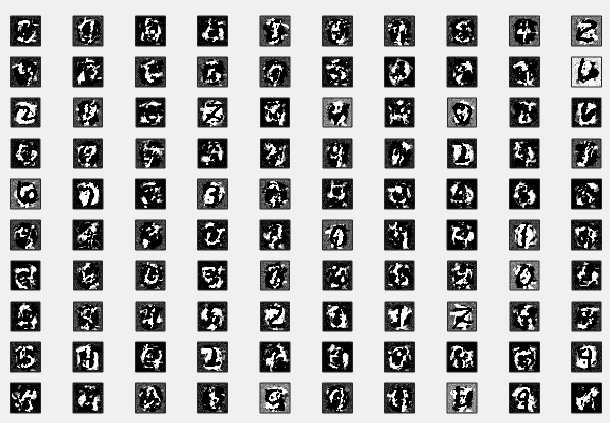


Figure 7-Autoencoder features with sparsity and weight decay constraints.

-In above figure 6 and figure 7 seems to be the most noteworthy difference. All the features in figure 6 seems to be blurry but in figure 7, features are clean and all different. Thus each of the neuron is learning different features about the data which we want to achieve here.

-This we could remove the redundancy in the system and make the network more efficient by adding these constraints as above. Each neuron will get tuned to a particular digit and we get a robust system.

**Appendix:**

* This code has 2 dependencies. Two files namely 'MNISTnumImages5000.txt' and 'MNISTnumLabels5000.txt' should be present in the system.
* Once you run the file HW4\_a\_collect\_features\_with\_sparsity.m the code should give all the figures as mentioned above except the figures 4,5 and 6 as those are taken from assignment 3.