University of CINCINNATI

INTELLIGENT SYSTEMS

EECE 6036

ASSIGNMENT -4

Submitted on:

12/04/2018

Submitted by

Siva Sai Krishna Paladugu

1. Problem Statement:

To train sparse auto-encoder network using the MNIST data as in problem 1 from HW 3 and to obtain good set of features for representing them by considering **two regularizations- sparseness and weight decay.** Also evaluating the performance using J_2 loss function to quantify the error. Also need to plots for cost for both training and test set and to plot features of hidden neurons in hidden layer. Also comparing the features obtained by this network with that of network obtained from problem 2 in home assignment 3.

System Description:

Given dataset contains 5000 records and distribution of each digit in dataset is as shown in table below. Data has been split into train and test dataset in such a way that each class/digit will have distribution in the ratio of 80:20 i.e., train-4000, test-1000 points. Also, again train data has been split to make validation dataset in such a way that train_new-3500 and validate-500. The distribution each digit into these groups can be seen below. The reason for this type of split is so that network will not be bias to any particular digit**. Here the main difference the output will be same as input i.e., we have to consider features of data only as target will be same as input.**

Digits	Original	Train	Test	Train_new	Validation
0	460	368	92	322	46
1	571	457	114	400	57
2	530	424	106	371	53
3	500	400	100	350	50
4	500	400	100	350	50
5	456	365	91	319	46
6	462	370	92	324	46
7	512	409	103	358	51
8	489	391	98	342	49
9	520	416	104	364	52

I am using same code by adding regularizations terms to update delta changes i.e., same class function which is used in implementing the Auto Encoder network is used just by adding weight decay and sparse penalty terms and also following **mini-batch approach**.

Strategy for SPARSENSS: Calculating **rho_j_cap** for **mini-batch** dataset points initially in an epoch and using **same** value for updating delta for hidden layer for **all points in mini-batch**. Again for second set of points will be considered and rho_j_cap is calculated for that and will be used in calculating penalty term to update delta and so on.

Neurons in Hidden Layer: As mentioned in question I am using same number of hidden neurons here in training the sparse auto encoder also. i.e., 150

Activation Functions: Tried with different combinations – (sigmoid, sigmoid), (relu, relu) and (relu, sigmoid). In problem 2 of HW 3, I have used **(Relu, Sigmoid)** combination which is giving less cost in less epochs and so considered it for final network but here I have used **(sigmoid, sigmoid)** as it is giving less cost when compared to other combination and also helps in avoiding **rho_j_cap** to become **zero** for small size of batch size.

Weights Initialization: Weights are chosen **which are giving good results from initial epochs** in such a way that they are

Uniform between (-a, +a) where **a = $\sqrt{(6/(N\_s+N\_t))}$**

N_s = number of neurons in source layer, Nt = number of neurons in target layer

Also started same set of initialization weights which are used for auto encoder initially so that to compare results.

*Xavier Glorot, Yoshua Bengio, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.

Learning rate: When using same learning rate i.e., 0.9 which is used in PB2-HW3, it is taking more time to get low cost and so I have changed my learning rate to 0.9 which is giving minimum cost with few hundreds of epochs. I thought to increase further like 10, but it is not giving less cost instead more cost and getting overflow due to overfitting as rho_j_cap tends to boundary values and so **chosen 0.9 as final**.

Momentum: When I used same momentum of PB2-HW3, it is taking more epochs or clearly error is starting from high value and so considered **as 0.9** which is starting with decent error. This is due to that momentum is considering the previous weights as weightage in calculating error. I have tried with 0.3, 0.7 and 0.9 and got **decent cost** with less epochs with **m=0.9**

Sparseness target (ρ): I have tried with different values like 0.03, 0.05, and 0.08 and observed that when ρ =0.03 and 0.08 there is possibility of getting overflow due to pho_j_cap reaching boundary conditions and so used 0.05 which is giving good results but taking more epochs as it controls the average number of activations on hidden layer and also getting good sparseness in features if we are taking **0.05** and chosen it as final.

Lagrange multiplier- Beta (β): I have tried with different values like 2,4,10 and observed that error obtained when β=2 is more when compared to error obtained when β=4 right from initial epochs. And when β=10, I am getting overflow errors as rho_j_cap tends to boundary values. So I have used **β=4 as final.**

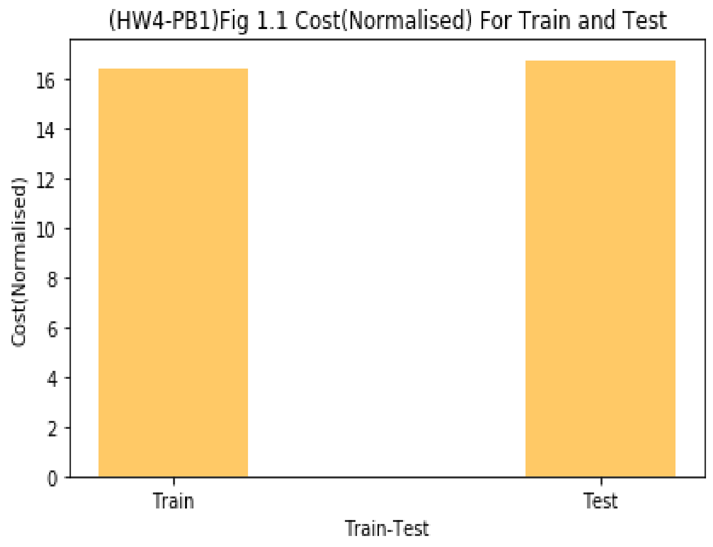
Lagrange multiplier- Lambda (λ): I have tried with two values 0.0001 and 0.001 and observed that weight decay with **λ=0.0001** is making some of the weights move towards zero in less epochs and so chosen as final.

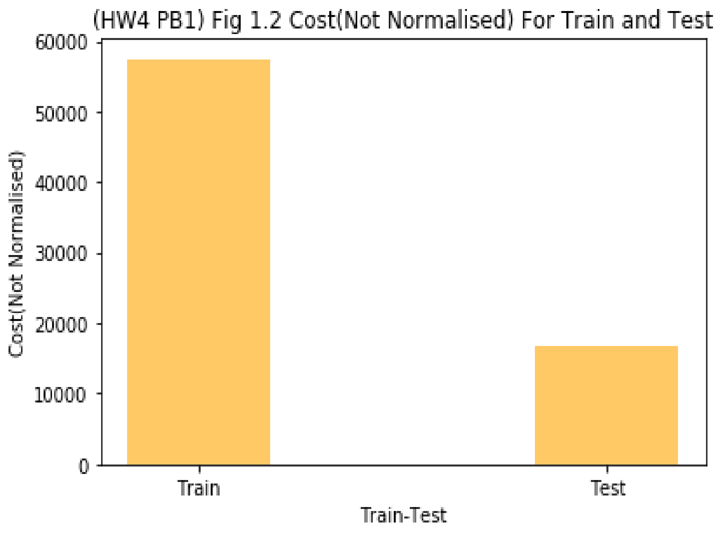
Rule for Stopping: Initially, I run without condition for 500 epochs and observed that there is decrease in all the errors but after few epochs there are more fluctuations but still they are decreasing and so ran for 500 epochs is considered for getting weights to plot sparse features of hidden neurons.

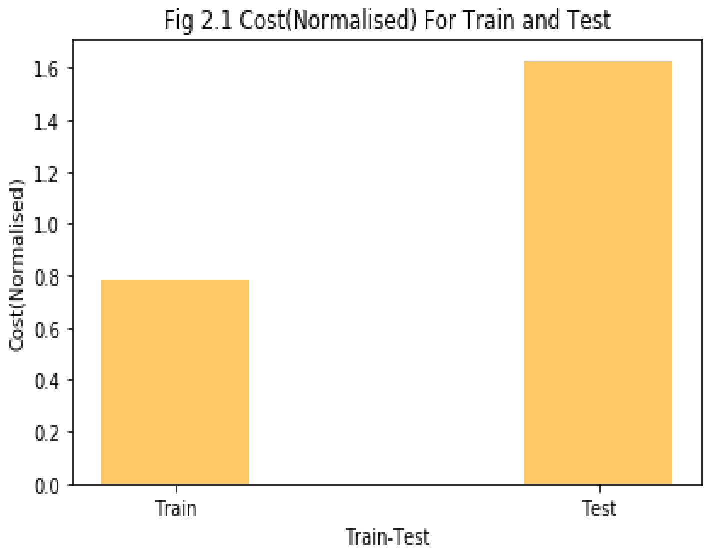
Results

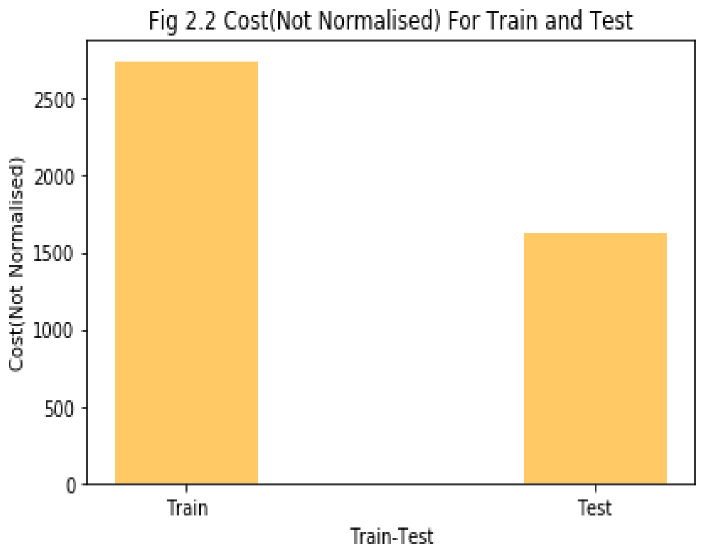
Plot (Required):

(HW4-PB1)Fig 1.1 shows the cost calculated for training and testing set such that overall function is normalized and observed test cost is very slightly higher than train cost and (HW4-PB1)Fig 1.2 shows the cost calculated in such a way that it is not normalized (not dividing by corresponding number of data points). Also figures 2.1 and 2.2 are corresponding figures from problem 2 of home assignment 3. Here we observed that in fig 1.3, the normalized train and test errors are almost similar making the **network not to over fit** and also as we considered limited epochs the training error is high compared to that of HW3-PB2.



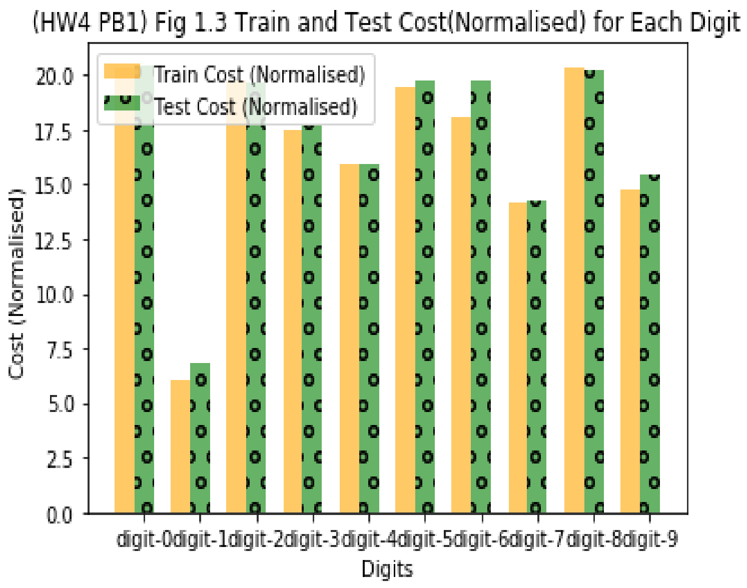


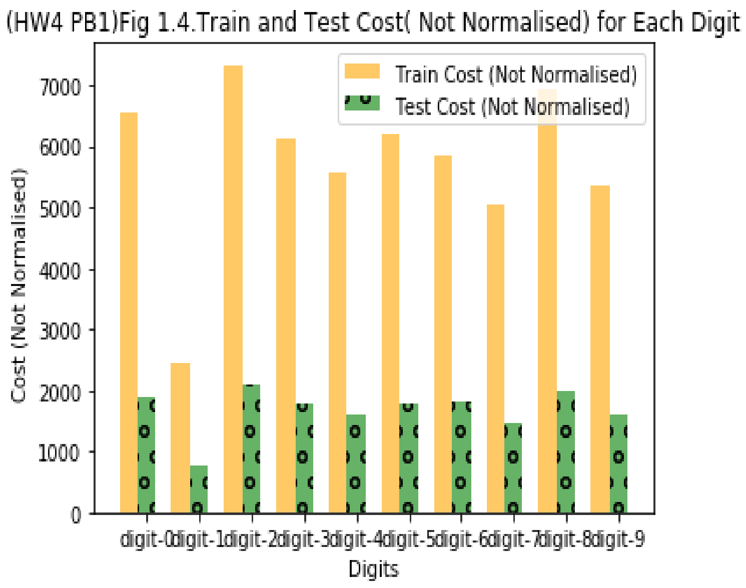


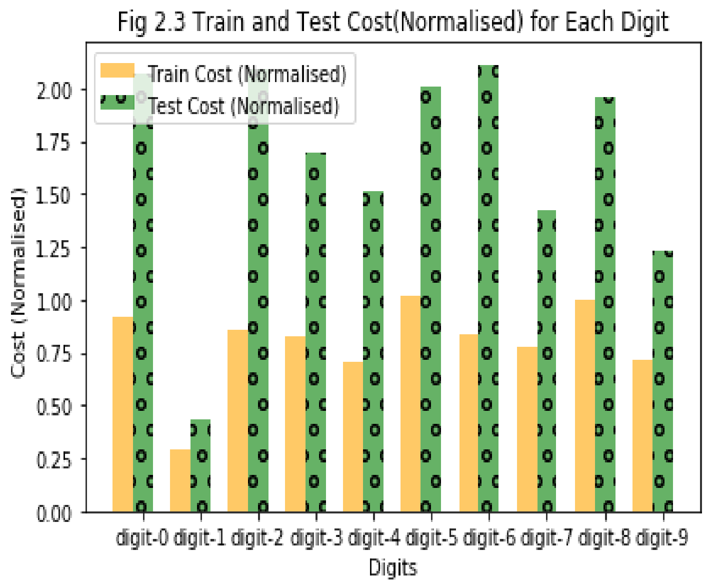


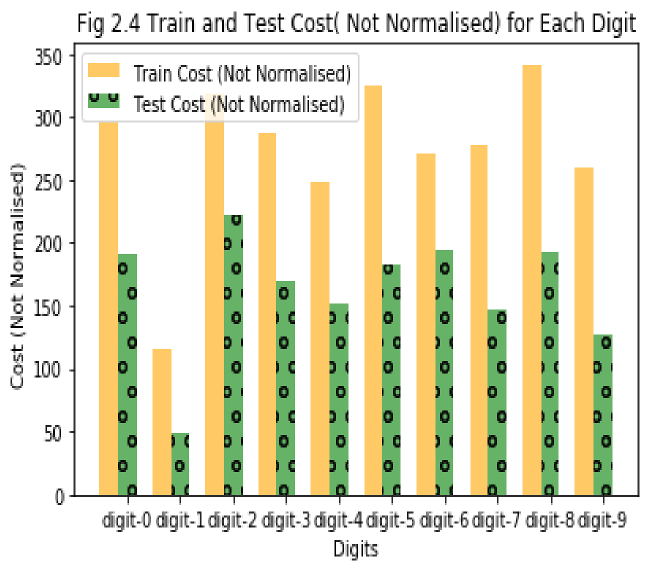
Plot (Required)

Fig 1.3 shows the cost calculated for each digit from training and testing set such that overall function of each digit is normalized and Fig 1.4 shows the cost calculated for each digit in such a way that it is not normalized (not dividing by corresponding number of data points). Also figures 2.3 and 2.4 are corresponding figures from problem 2 of home assignment 3. Here we observed that the patter it followed is same i.e., loss contribution by each digit pattern is almost similar in two home works.





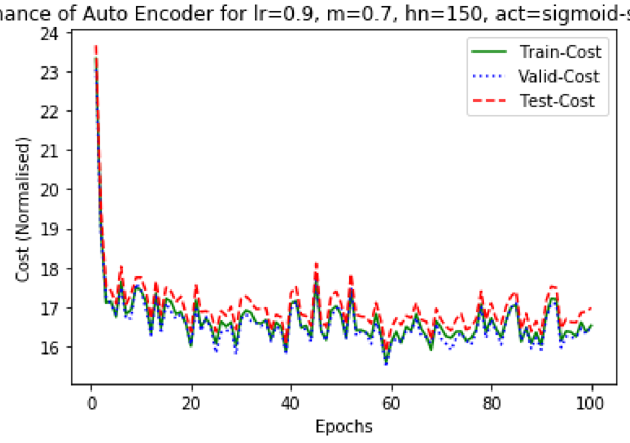




Plot (Required)

Fig 1.5.a shows the training error for train, test and validation considering every epoch values up to 100 epochs.Fig 1.5.b shows the training error for train, test and validation considering every 10th epoch values up to 500 epochs. (To get fit and good view, it plotted in 100 intervals but considering all 10th epoch values). We can see fluctuations but still decreasing with epochs. Also Fig 2.5 is corresponding graph from problem 2 of HW3.

(HW4 PB1)Fig 1.5.a Performance of Auto Encoder for lr=0.9, m=0.7, hn=150, act=sigmoid-sigmoid, bs=32,beta=4,rho=0.05



(HW4 PB1)Fig 1.5.b Performance of Auto Encoder for lr=0.9, m=0.9, hn=150, act=sigmoid-sigmoid, bs=32,beta=4,rho=0.03

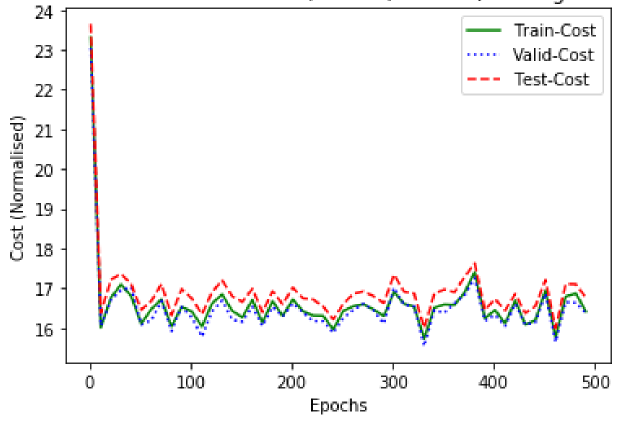
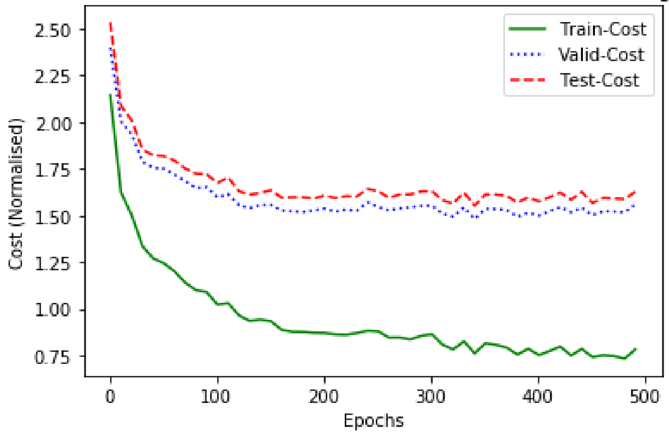
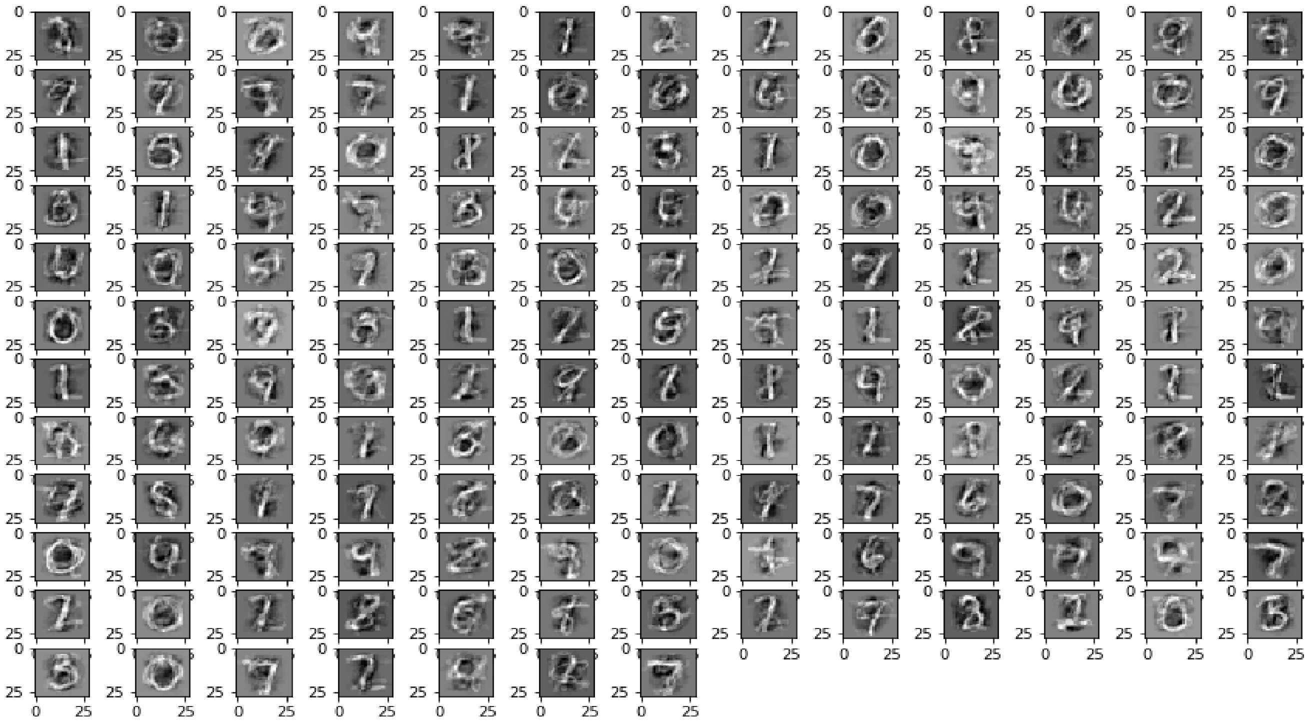


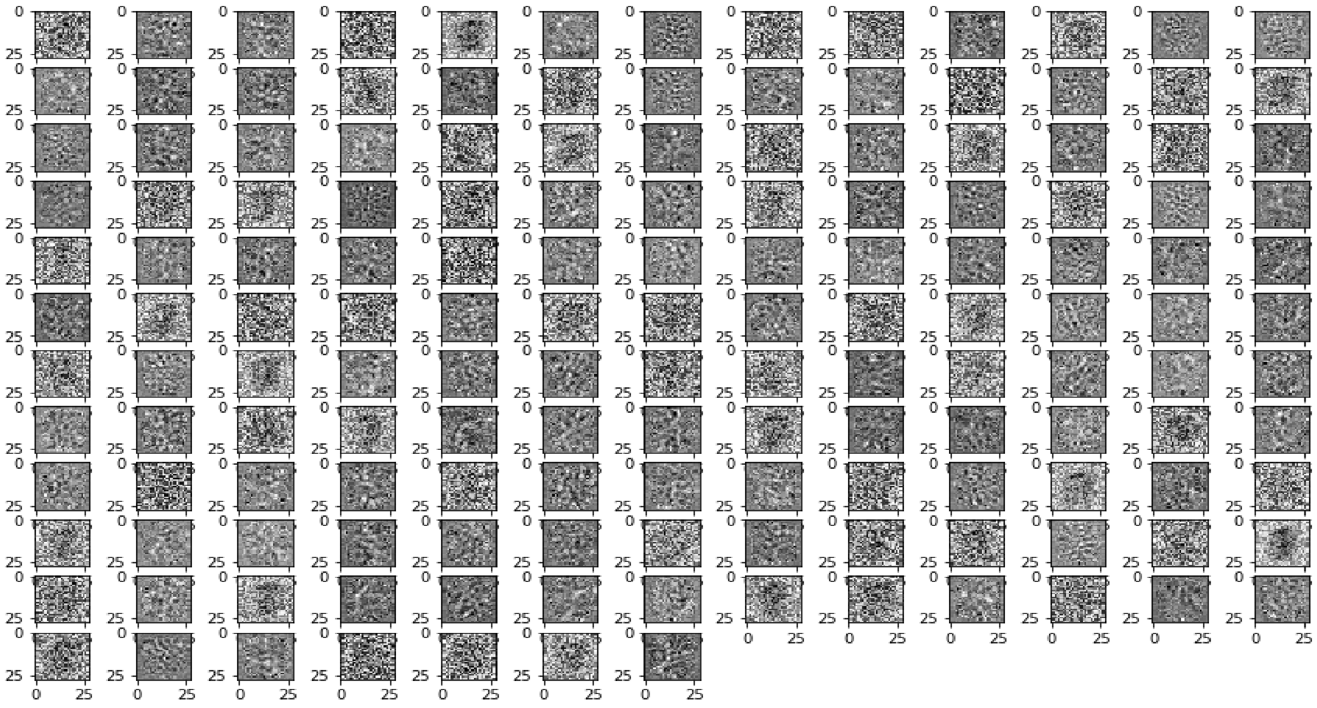
Fig 2.5 Performance of Auto Encoder for lr=0.1, m=0.7, hn=150, act=relu-sigmoid, bs=32



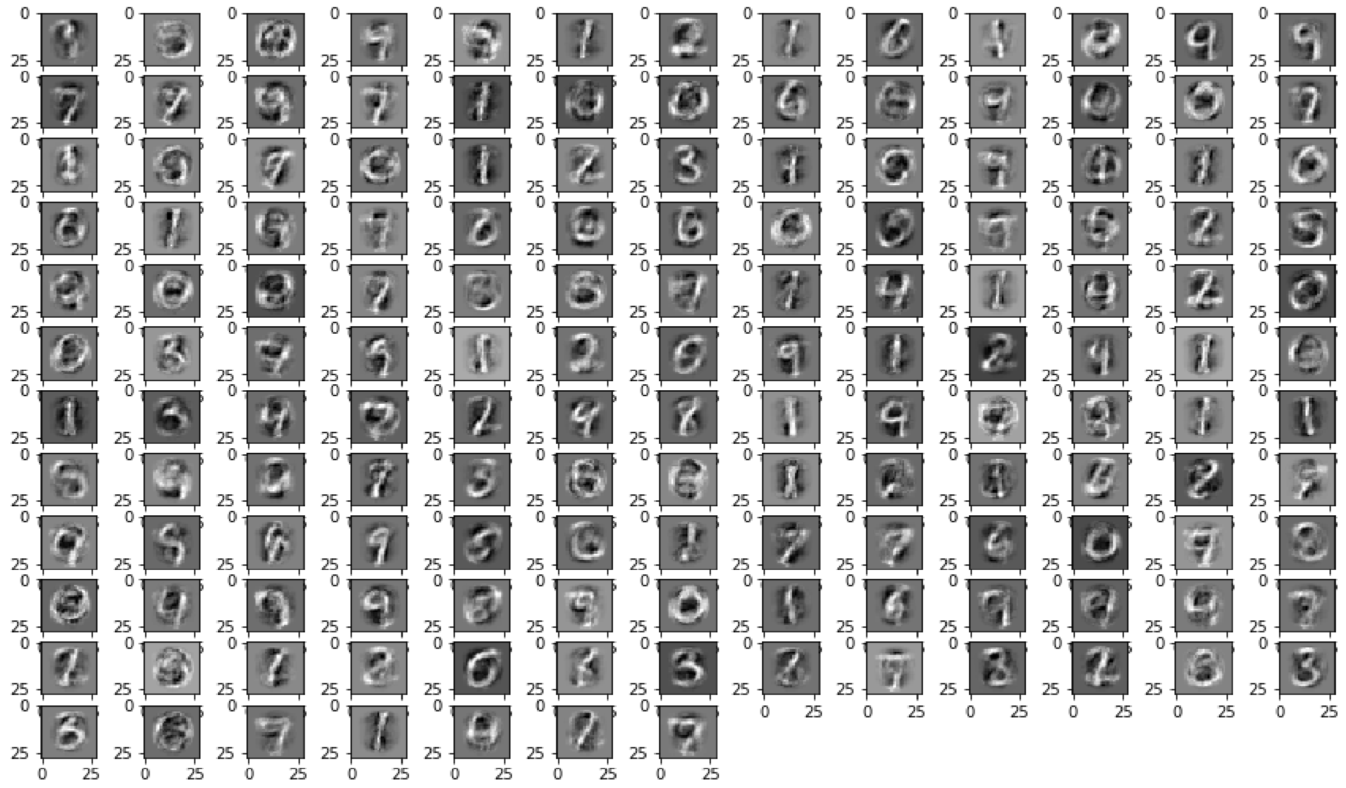
(Features) Plot (Required)
Here I have plotted the gray scale images such that considering for each neuron in hidden layer .i.e., for all 150 neurons, the incoming weights in a such a way that each neuron will have 784 weights from inputs and also outgoing weights in a such a way that each neuron will have to be connected to 784 output neurons and plots are as follows:
For weights that are from input to hidden layer in HW4-PB1:



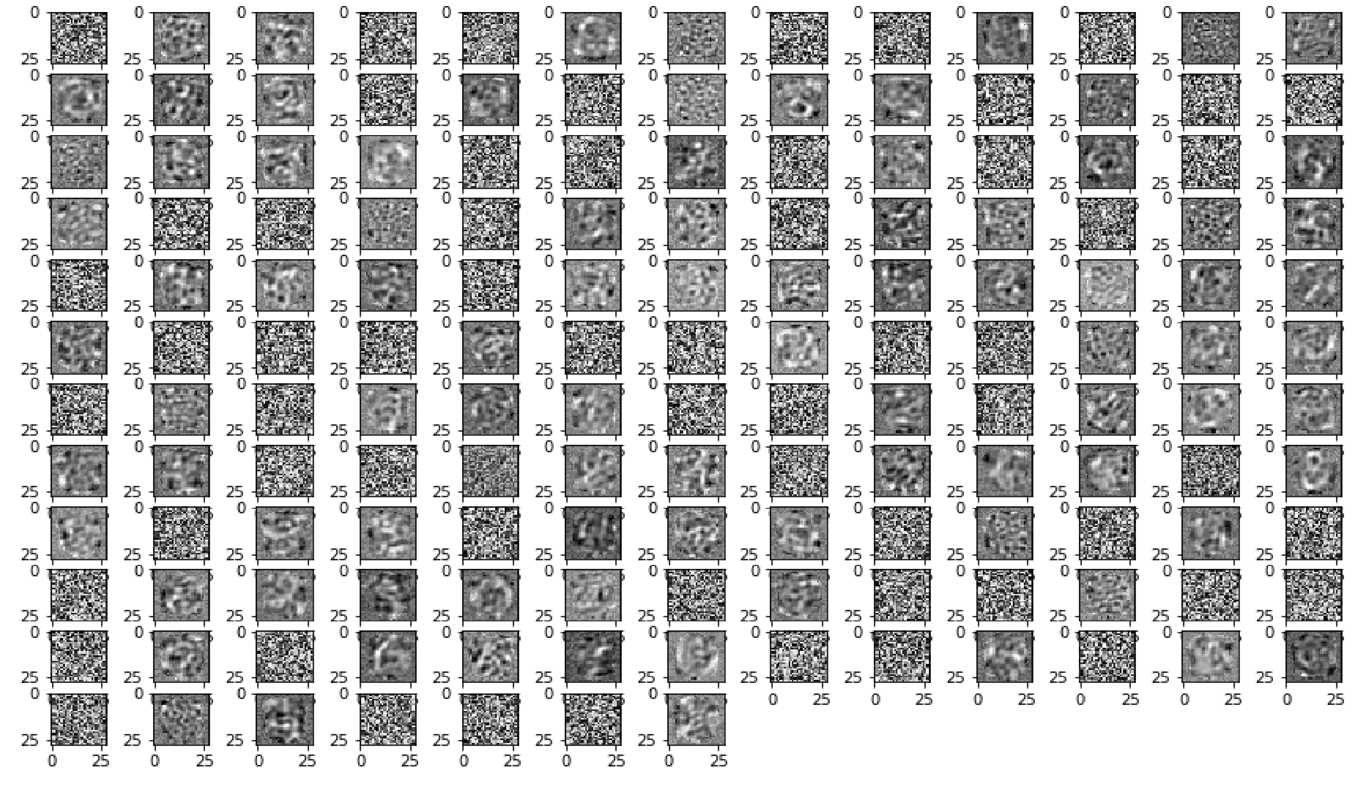
For weights that are from input to hidden layer in HW3-PB2:



For weights that are from hidden to output layer in HW4-PB1:



For weights that are from hidden to output layer in HW3-PB2



Analysis of Results:

From the results we can say that when Sparse Auto Encoder network trained with MNIST data, we are getting minimum loss function in more epochs but getting excellent sparse features as we are putting constraints on neuron in hidden layer and are **able to reconstruct** the input images by extracting features using weights. Also we have observed the tuning of parameters will lead to get decent cost function as explained in system description section. Also in any case the network is getting slow down to react after few hundreds of unless we change any parameter. We got costs more as compared to that of normal auto encoder in HW3PB2 without normalization though taken more epochs with lr=0.9, m=0.9, 150 hidden neurons and activation function as sigmoid-sigmoid for hidden and output layer. But if we ran more epochs the losses can be less than that of auto encoder's losses from HW3PB2. **From cost plot of each digit we can say that train data set can be reconstructed easily when compared to test set. Also from each cost of digit we can say that digit 1 can be easily reconstructed as it is having less cost (error) and so can be reconstructed easily. While digit 2 and 8 are having highest cost and so cannot be reconstructed easily**. **Also the pattern followed in both type auto encoders are almost similar.** Here we observed weights of each neuron in hidden layer are getting reacted some particular feature from the input. We have observed more sparsed features in case of sparse auto encoder. From features plot we can say that any digit having high pixel at the pixel where weight map is having high value will react to that pixel. **Here all weights are learning and we see no weight without any feature. But of few of weights are learning the same and so we can use dropout concept further if we better reconstruction error to eliminate those neuron which are learning the same.** While in HW3PB2, some weights are not learning and no use. Clearly we can see **sparse auto encoder extracted excellent features and all weights with some meaningful features compared to the auto encoder** implemented in HW3PB2.

Appendix: Program

I have written code in python 3 in jupyter notebook where we can see output for each cell there itself and also code file will be saved in .ipynb extension. To run code change .txt extension of M12992211_Assignment4_Q1_code(ipynb).TXT to .ipynb and open in jupyter where we can all cells at once or each cell at a time. Here we can see results (graphs) in notebook file itself.

2. Problem Statement:

To implement two networks, A and B which will use the weights or feature detectors from both auto encoders for classification problem. Network A will use weights or features obtained in PB2HW3 from input to hidden. Network B will use weights or features obtained in PB1HW4 from input to hidden layer. Also evaluating the performance of both networks with confusion matrix and comparing features of both networks and network obtained from PB1HW3.

System Description:

Given dataset contains 5000 records and distribution of each digit in dataset is as shown in table below. Data has been split into train and test dataset in such a way that each class/digit will have distribution in the ratio of 80:20 i.e., train-4000, test-1000 points. Also, again train data has been split to make validation dataset in such a way that train_new-3500 and validate-500. The distribution each digit into these groups can be seen below. **The reason for this type of split is so that network will not be bias to any particular digit.**

Digits	Original	Train	Test	Train_new	Validation
0	460	368	92	322	46
1	571	457	114	400	57
2	530	424	106	371	53
3	500	400	100	350	50
4	500	400	100	350	50
5	456	365	91	319	46
6	462	370	92	324	46
7	512	409	103	358	51
8	489	391	98	342	49
9	520	416	104	364	52

I have changed the code of the class Model in which weights from input to hidden are set to given weights and weights from hidden to output are initialized randomly.

Neurons in Hidden Layer: As mentioned in question I am using same number of hidden neurons here in training both the networks also. i.e., 150.

Activation Functions: (sigmoid, sigmoid) combination is giving good hit rate in less epochs and so considered it for both network A and B. (But in HW3PB1, I have used relu-relu combination which has given very good accuracy)

Weights Initialization: For Network A, input to hidden weights are set to features obtained in HW3PB2. For Network B, input to hidden weights are set to features obtained in HW4PB1.

For weights from hidden to output, First I have tried with random numbers between 0 and 1 which are not giving good results as it is taking more epochs to get significant change in hit rate. So I have chosen weights **which are giving good results from intial epochs** in such a way that they are

Uniform between (-a, +a) where $\mathbf{a = \sqrt{(6/(N\_s+N\_t))}}$

N_s = number of neurons in source layer, Nt = number of neurons in target layer

*Xavier Glorot, Yoshua Bengio, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, PMLR 9:249-256, 2010.

Learning rate: With decreasing learning rate, I have observed that training time is taking more time to get good hit rate i.e., slowly learning takes place. Also I have taken very high learning rate which is giving me very low hit rate and so I have made my choice as **0.01 for Network A and 0.1 for Network B.**

Momentum: I have tried without giving momentum at first, and observed that to get decent rate it took more epochs. When I used in momentum, there is decent hit rate with less epochs. This is due to that momentum is considering the previous weights as weightage in calculating error. I have tried with 0.3 and 0.7 and got good hit rate with less epochs with **m=0.9**

Rule for Stopping: Initially, I run without condition for 500 epochs and observed that validation and train hit rates are increasing and approaching towards 1. So I tried to further increase to **1250**, then for network A, testing hit rate and validation hit rate are becoming constant. But for network B, testing hit rate is increasing but not significantly and so stopped for 1250 epochs. When the validation hit rate is almost constant, I am stopping the network to train further.

Choice of Ranges: Very beginning I tried with random weights and ranges as 0.75 and 0.25 while calculating hit rate and I am getting some points with more than one element less than 0.25 and so unable to assign label and so I used **max threshold** concept for evaluation of training set's hit rate which is in sync with evaluation of test set and validation set.

Results (For network A)

Actual
(0-to-9 left to right)

Predicted
(0-to-9 top-to bottom)

Confusion Matrix for Train	Confusion Matrix for Test	Confusion Matrix for Validation
[[314, 0, 4, 1, 1, 9, 7, 2, 4, 3],	[[90, 0, 1, 0, 0, 2, 3, 0, 0, 2]	[[45, 0, 2, 0, 0, 1, 1, 0, 0, 1],
[0, 391, 1, 1, 1, 2, 2, 5, 6, 4],	[0, 109, 0, 0, 0, 0, 2, 5, 3, 2	[0, 56, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 1, 338, 8, 0, 2, 2, 7, 0, 0],	[0, 1, 93, 0, 1, 0, 1, 2, 0, 0]	[0, 0, 45, 1, 0, 0, 0, 2, 1, 0],
[0, 0, 4, 316, 0, 9, 0, 3, 9, 6],	[0, 1, 2, 85, 0, 4, 0, 1, 3, 2]	[0, 0, 0, 44, 0, 5, 0, 0, 1, 1],
[0, 0, 5, 1, 330, 3, 2, 8, 9, 9],	[0, 2, 2, 1, 95, 3, 0, 0, 1, 6]	[0, 0, 2, 0, 47, 1, 0, 2, 1, 3],
[1, 0, 0, 6, 0, 274, 3, 0, 7, 2],	[0, 0, 0, 3, 0, 72, 0, 0, 1, 0]	[0, 0, 1, 0, 0, 34, 1, 1, 2, 0],
[3, 3, 4, 1, 4, 4, 306, 0, 3, 0],	[2, 0, 0, 2, 1, 4, 85, 0, 3, 0]	[1, 1, 0, 0, 1, 1, 43, 0, 0, 0],
[0, 1, 10, 10, 1, 3, 1, 326, 4, 8],	[0, 1, 2, 3, 0, 2, 0, 89, 2, 5]	[0, 0, 2, 3, 0, 1, 1, 44, 1, 2],
[4, 4, 3, 5, 1, 12, 1, 3, 297, 4],	[0, 0, 6, 4, 1, 4, 1, 1, 85, 2]	[0, 0, 1, 2, 0, 2, 0, 0, 43, 3],
[0, 0, 2, 1, 12, 1, 0, 4, 3, 328]]	[0, 0, 0, 2, 2, 0, 0, 5, 0, 85]	[0, 0, 0, 0, 1, 1, 0, 2, 0, 42]]

Results (For network B)

Actual
(0-to-9 left to right)

Predicted
(0-to-9 top-to bottom)

Confusion Matrix for Train	Confusion Matrix for Test	Confusion Matrix for Validation
[[316, 0, 8, 0, 0, 20, 9, 2, 8, 6],	[[90, 0, 4, 0, 1, 7, 5, 2, 1, 3]	[[44, 0, 3, 0, 0, 3, 1, 0, 0, 2],
[0, 395, 0, 0, 1, 6, 2, 5, 3, 5],	[0, 104, 0, 0, 0, 0, 1, 5, 2, 2	[0, 54, 1, 0, 2, 0, 0, 1, 0, 1],
[0, 0, 305, 1, 0, 0, 0, 3, 0, 0],	[0, 0, 87, 0, 0, 0, 0, 3, 0, 0]	[0, 0, 39, 0, 0, 0, 0, 1, 1, 0],
[0, 2, 21, 340, 0, 30, 1, 5, 18, 9],	[0, 5, 5, 90, 0, 18, 0, 2, 7, 3	[1, 3, 2, 47, 0, 6, 0, 0, 2, 2],
[0, 1, 13, 2, 342, 16, 4, 10, 8, 39],	[0, 2, 2, 1, 95, 3, 3, 3, 5, 16	[0, 0, 3, 0, 47, 4, 0, 2, 1, 5],
[0, 0, 0, 0, 0, 218, 0, 0, 2, 3],	[0, 0, 1, 1, 0, 53, 0, 0, 1, 1]	[0, 0, 0, 0, 0, 28, 1, 0, 2, 3],
[2, 2, 4, 0, 2, 6, 307, 0, 2, 2],	[2, 0, 0, 1, 1, 5, 83, 0, 3, 0]	[0, 0, 2, 0, 1, 1, 44, 0, 0, 0],
[1, 0, 12, 5, 0, 7, 0, 328, 3, 13],	[0, 0, 2, 4, 0, 0, 0, 87, 1, 5]	[0, 0, 1, 2, 0, 1, 0, 45, 1, 3],
[3, 0, 7, 1, 1, 15, 1, 4, 296, 8],	[0, 3, 5, 3, 3, 4, 0, 0, 78, 3]	[1, 0, 2, 1, 0, 2, 0, 1, 42, 3],
[0, 0, 1, 1, 4, 1, 0, 1, 2, 279]]	[0, 0, 0, 0, 0, 1, 0, 1, 0, 71]	[0, 0, 0, 0, 0, 1, 0, 1, 0, 33]]

Plot (Required) (Network A & B):

From both the figures in Fig 2.1(HW4PB2), I have observed that for network A, the testing and validation error rates are becoming constant after some epochs while for network B the testing and validation error rates are decreasing but significantly and so considered only for 1250 epochs. Also even though learning rate for network B is high it took more epochs to get significant constant values.

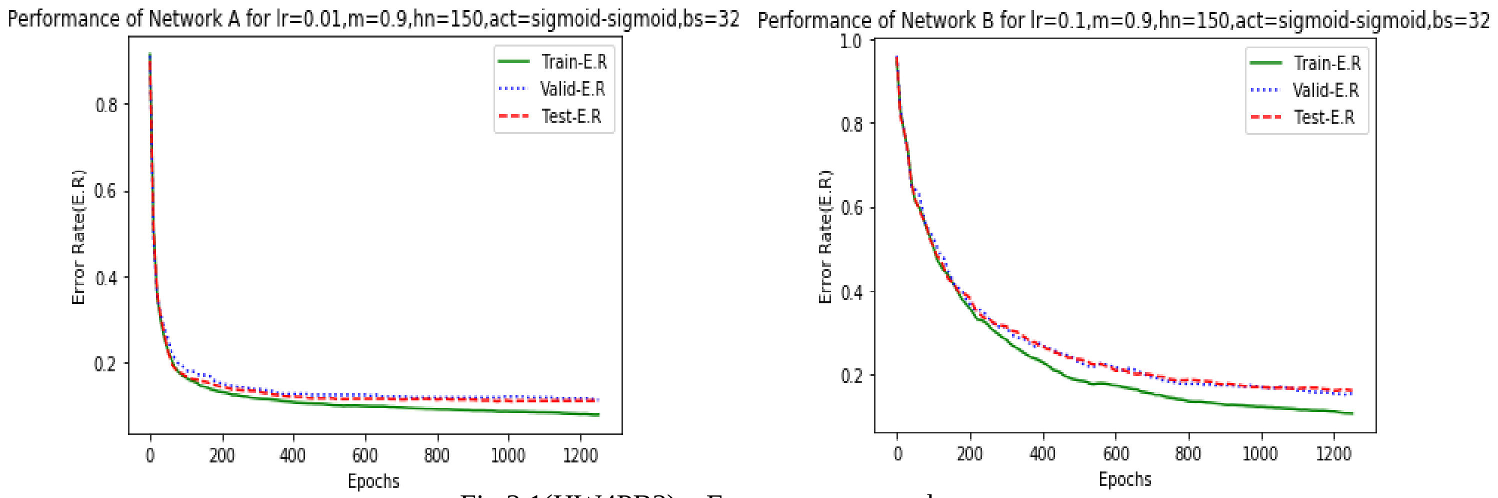
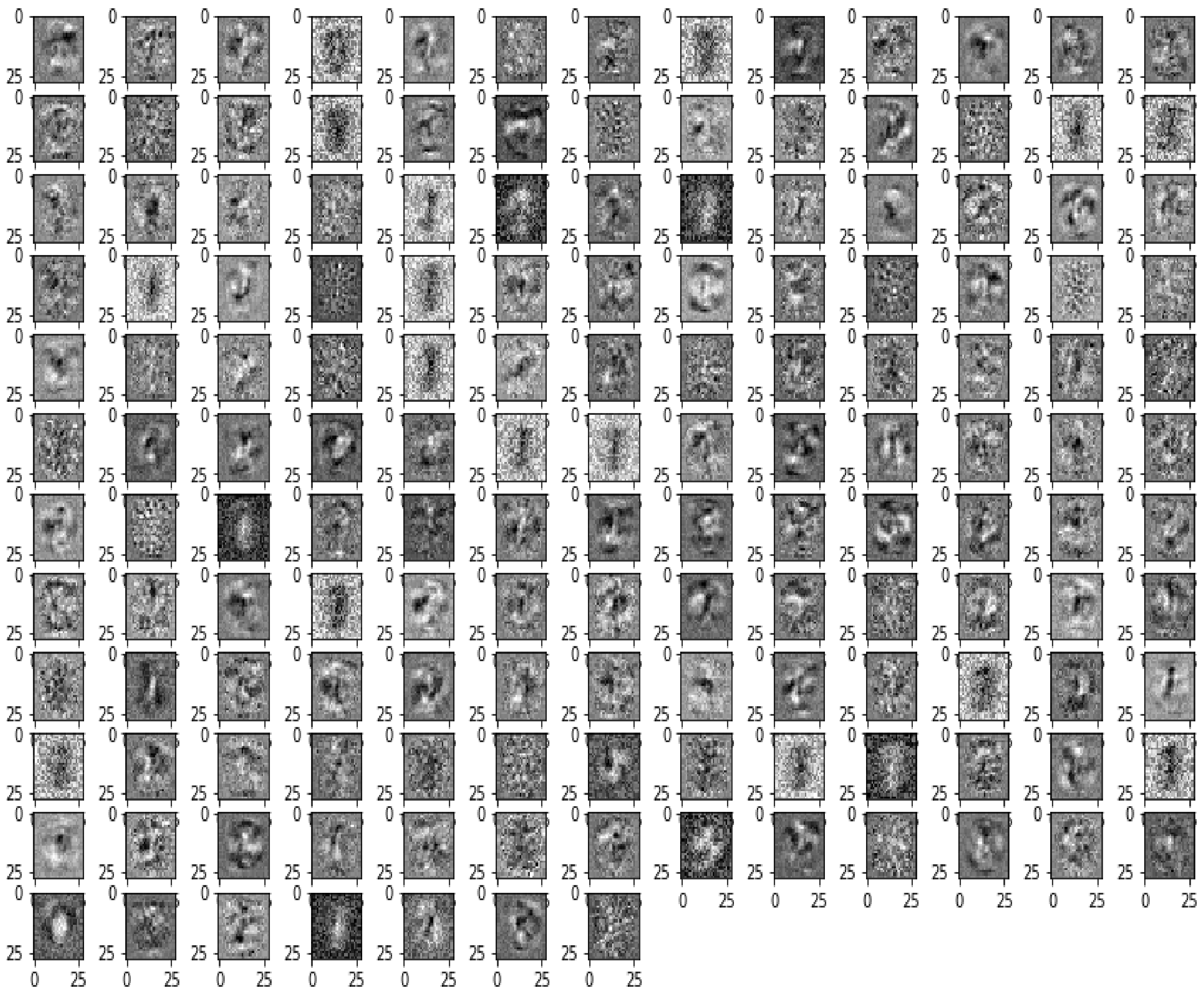


Fig 2.1(HW4PB2) – Error rates vs epochs

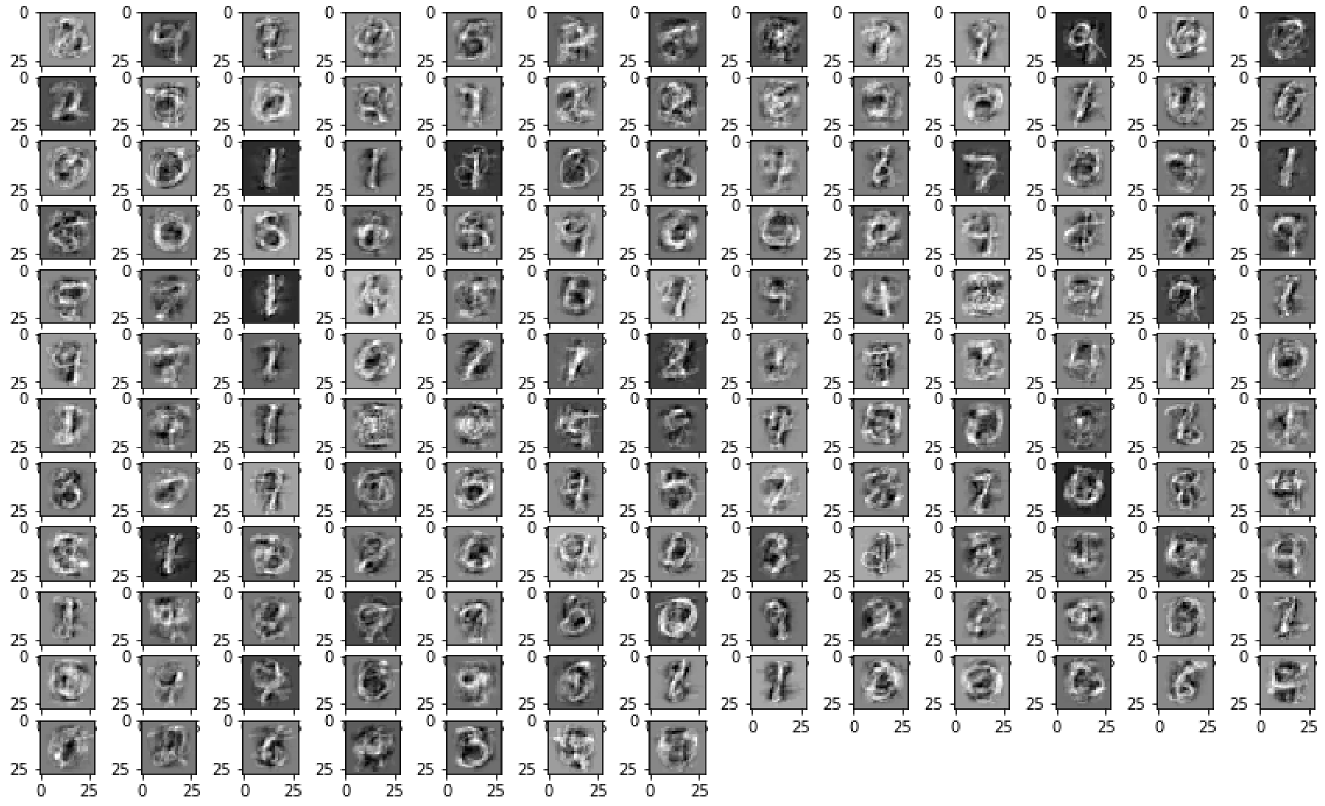
(Feature comparison) Plot (Required) (Network A & B & HW3PB1):

Here I have plotted the gray scale images such that considering for each neuron in hidden layer .i.e., for all 150 neurons, the incoming weights in such a way that each neuron will have 784 weights from inputs and plots are as follows:

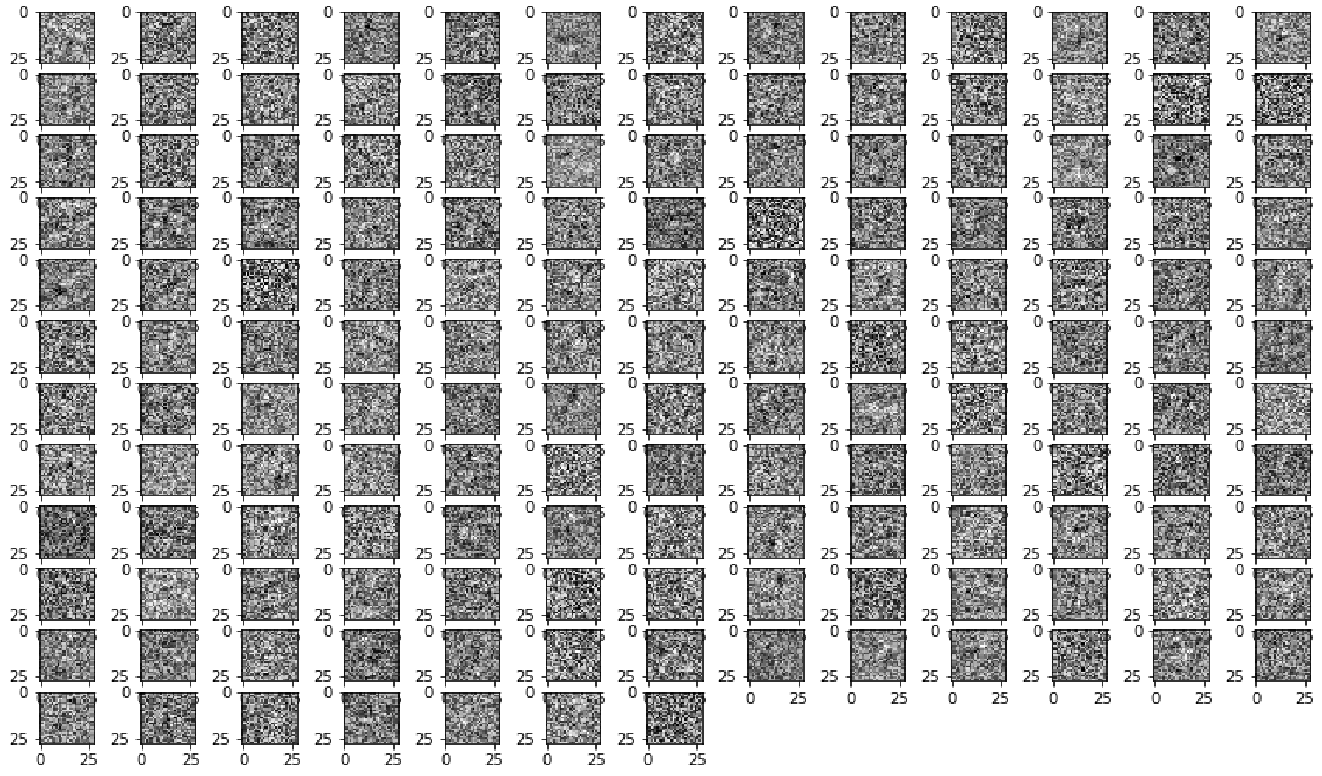
For weights that are from hidden to output layer for network A:



For weights that are from hidden to output layer for network B:



For weights that are from hidden to output layer for network obtained in PB1-HW3:



Analysis of Results:

From the figures of error rates of both network A and network B, we can say that network B is having slightly greater performance compared to network A. Also if we can run further network B is having improvement in performance but network A is having almost constant performance. This clearly states that **classification accuracy when using sparse auto encoder's features is high when compared to network that uses features from normal auto encoder (pb2hw3). Also we observed that classification accuracy of network from problem 1-HW3 is higher than the accuracies obtained using network A and network B. (Testing hit rate for network in problem-1-HW3 is (~0.92), and for network A (~0.87) and for network B (~0.89)). This clearly indicates that the network in PB-1-HW3 is exclusively designed for classification and we got more accuracy. Also if we train the sparse auto encoders for more epochs say in 2000-3000 we can get better features and that may lead to good classification accuracies in network used in HW4 compared to PB1-HW3 network accuracy.** Also from confusion matrices of both networks which follows same pattern almost, we can say that 0's and 1's are also almost classified correctly and 8 has more misclassification when compared with other digits. This behavior may be due to some samples of some digits are more in dataset and so trained accordingly.

From Feature plots of network-A, network B and network obtained in PB1-HW3 we can say that excellent features are extracted in network B compared to that network A and network obtained in PB1-HW3. Also features extracted in network A are better when compared with features that are obtained from network of PB1-HW3. This indicates that reconstruction can be better if we use sparse auto encoders when compared with other two. Also reconstruction with auto encoder i.e., PB2 HW3 will be better compared to network of PB1-HW3 which may not be case with classification accuracy as PB1-HW3 is exclusively trained for classification.

Appendix: Program

I have written code in python 3 in jupyter notebook where we can see output for each cell there itself and also code file will be saved in .ipynb extension. To run code change .txt extension of M12992211_Assignment4_Q2_code(ipynb).TXT to .ipynb and open in jupyter where we can all cells at once or each cell at a time. Here we can see results (graphs) in notebook file itself.
Also here two pickles files are used to set weights for network which are needed for running code for problem 2. Unable to upload them through blackboard but can be shared through mail if required.