MACHINE LEARNING LAB 3

SUBMITTED BY-
KARTIK VISHWAKARMA(2017CSM1001)
SAGARIKA SHARMA(2017CSM1012)

EXPERIMENT-1

- In this lab, experimentation with multi-layer perceptron is done. Experimentation is done with a simple 2 dimensional 3 class classification problem to visualize decision boundaries learned by a MLP.
- Aim is to study the changes to the decision boundary and the training error with respect to parameters such as number of training iterations, number of hidden layer neurons and finally the learning rate.
- Assumption is made for cross entropy as the error function, sigmoid as the activation function for the hidden layer units and softmax function for the output layer units.
- The output of the next work is the probability of classification.

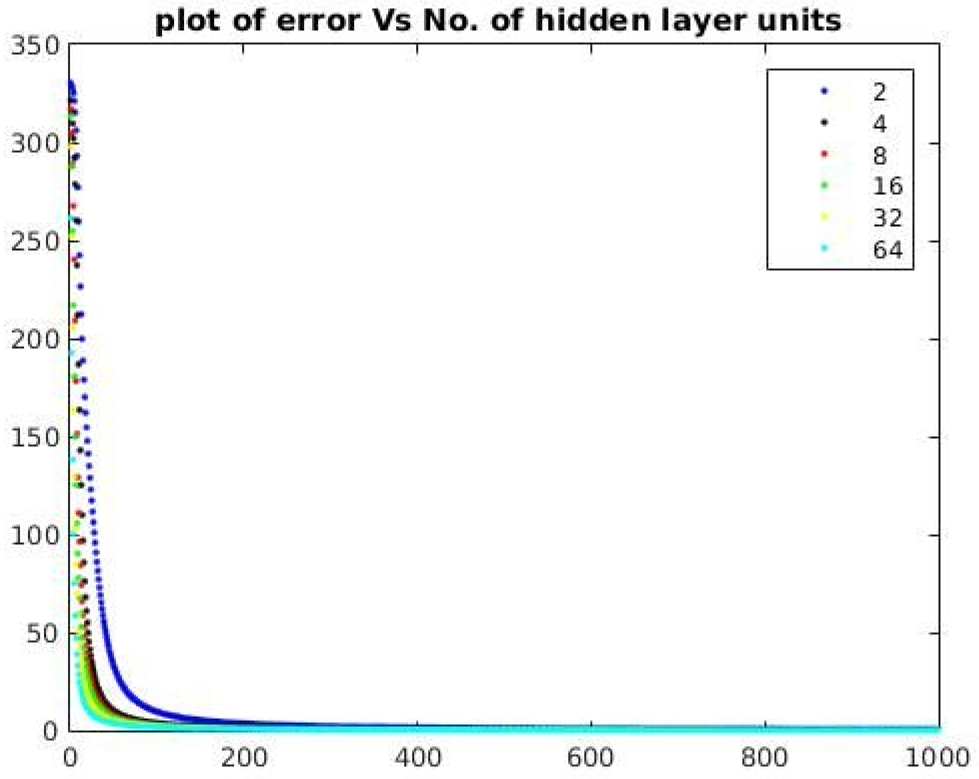
Observations:

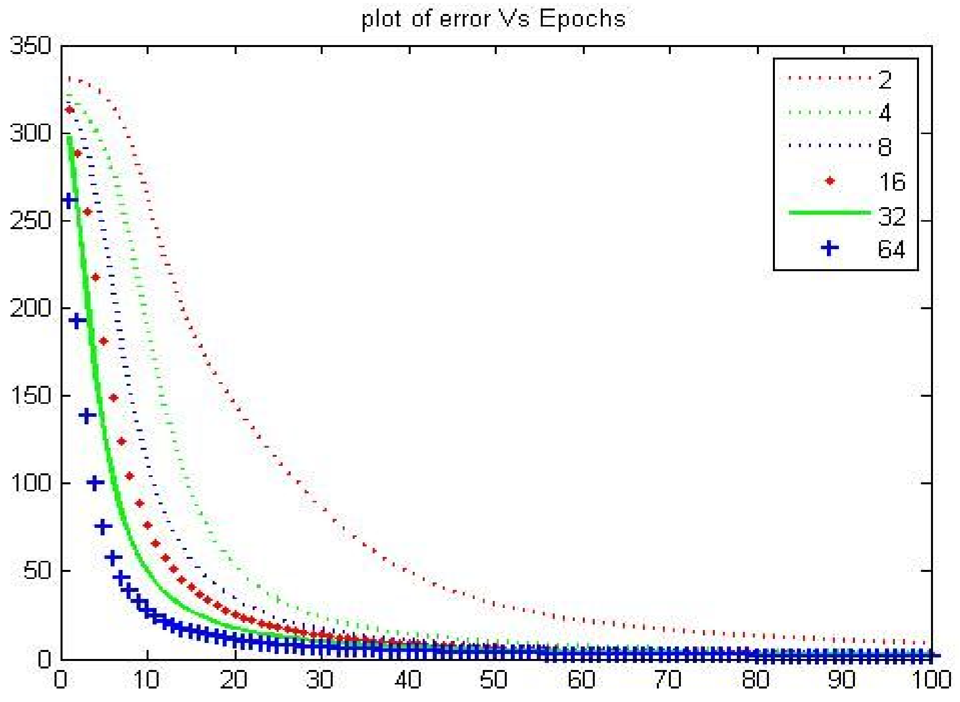
➔Varying the Learning Rate

❖ The number of epochs are fixed to be 1000, the learning rate to be 0.01 and the relation between the training error and the number of hidden layer units is observed by varying the number of hidden layer units in power of 2 starting from 2 and going till 64. Following are the observed results:

S.No	No. of Hidden Layer Units	Error
1	2	9.7934
2	4	4.3809
3	8	2.9844
4	16	2.5983
5	32	2.0992
6	64	1.7098

❖ Following is a single graph that includes the plots for training error(sum of squared error) against number of epochs for the different choices of hidden layer units:

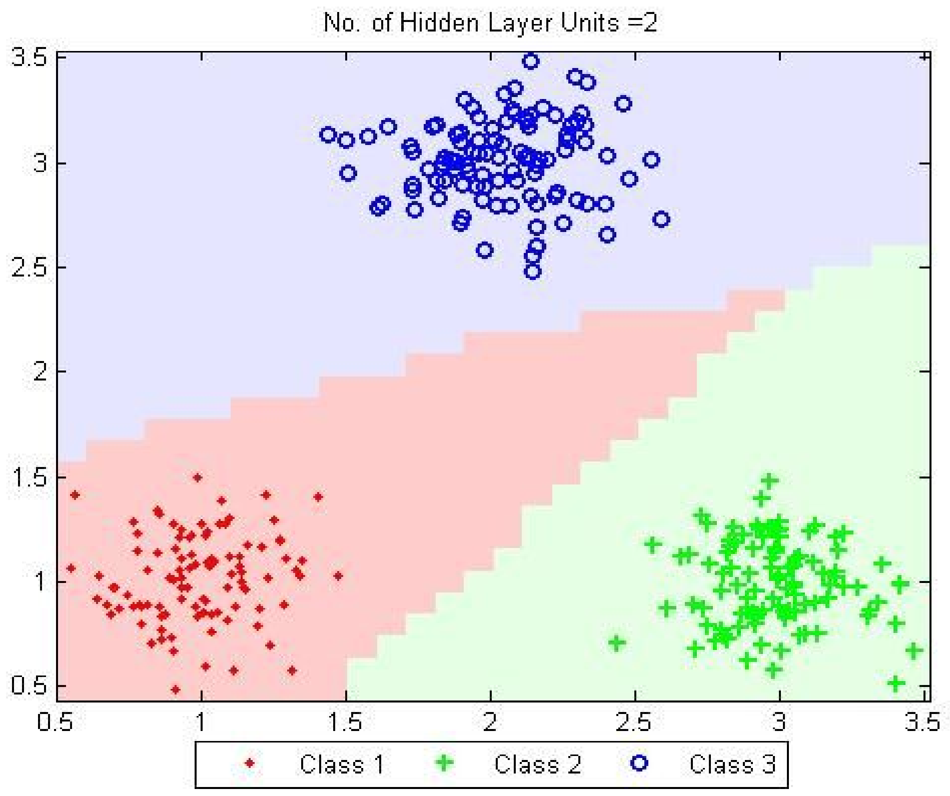

plot of error Vs No. of hidden layer units
350
300
250
200
150
100
50
0
0
200
400
600
800
1000
2
4
8
16
32
64

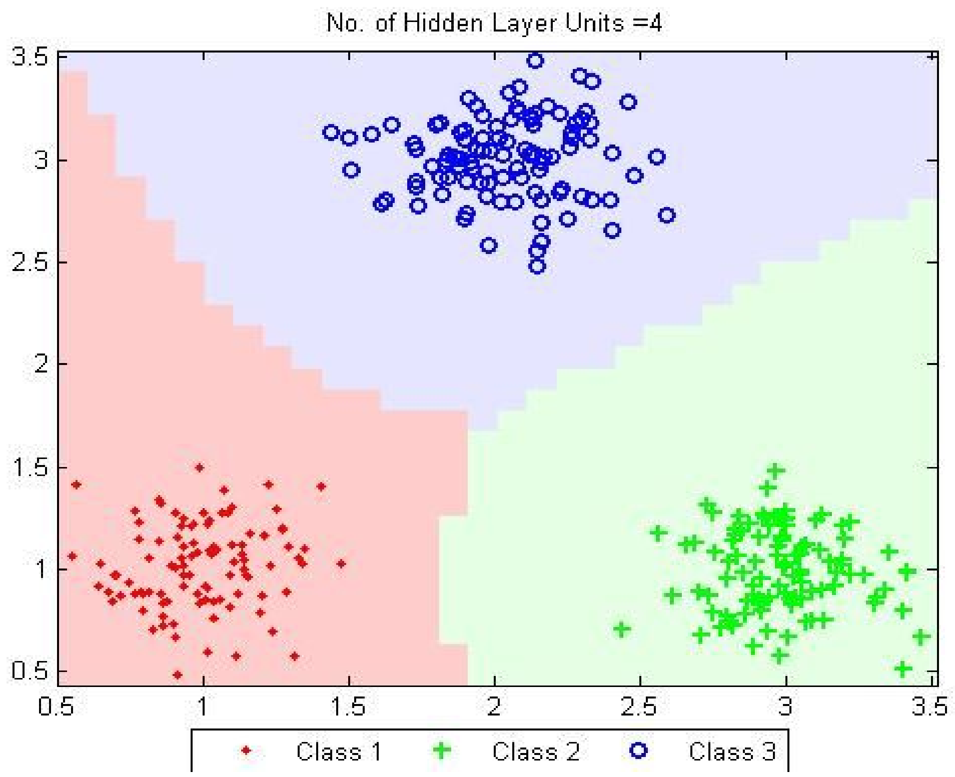

plot of error Vs Epochs
350
300
250
200
150
100
50
0
0
10
20
30
40
50
60
70
80
90
100
2
4
8
16
32
64

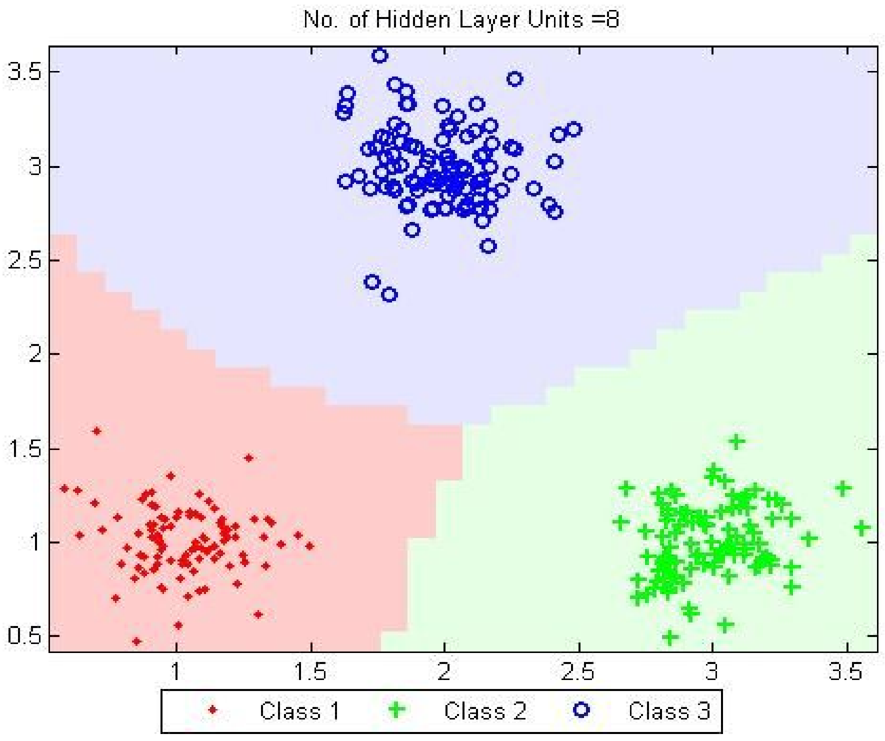
- ❖ Dark blue curve represents error plot with 64 hidden layer nodes.
  Green curve represents error plot with 32 hidden layer nodes.
  Dark blue curve represents error plot with 16 hidden layer nodes.
  Red curve represents error plot with 8 hidden layer nodes.
  Light blue curve represents error plot with 4 hidden layer nodes.
  Pink curve represents error plot with 2 hidden layer nodes.

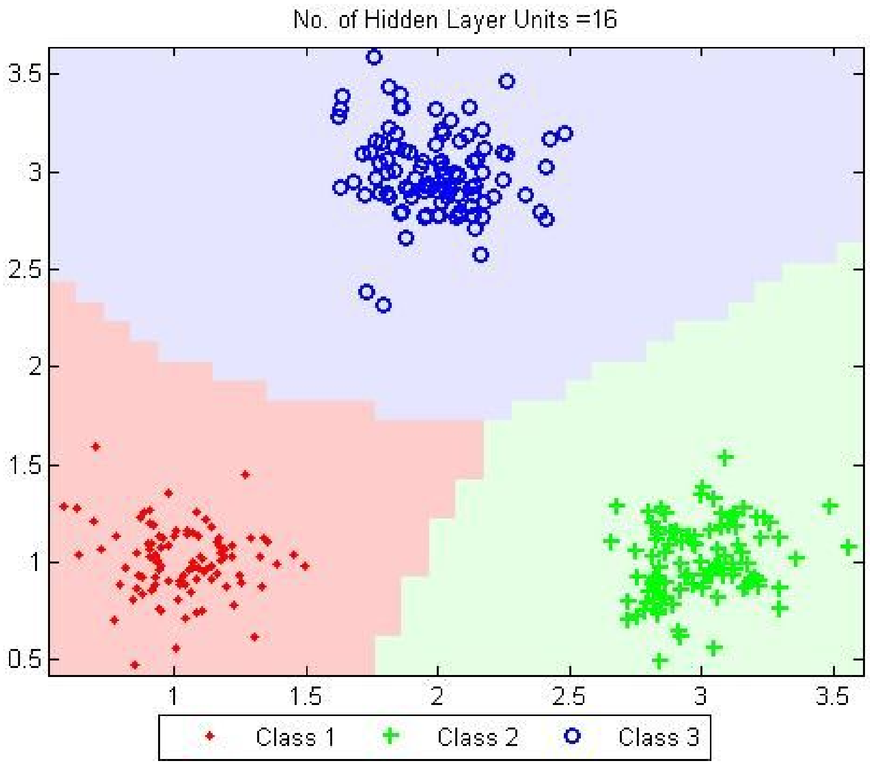
❖ It can be observed that for this model,for a given learning rate and fixed number of epochs, the error reduces on a faster rate as the number of hidden layer nodes increase from 2 to 64.

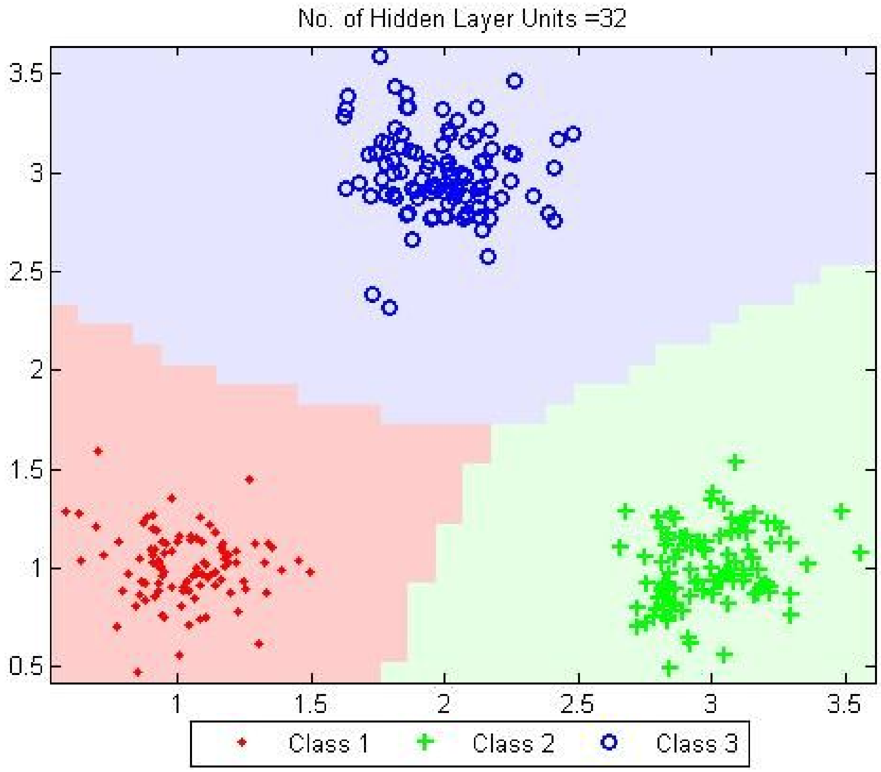
❖ Following is the plot for decision boundary for different number of hidden layer units:

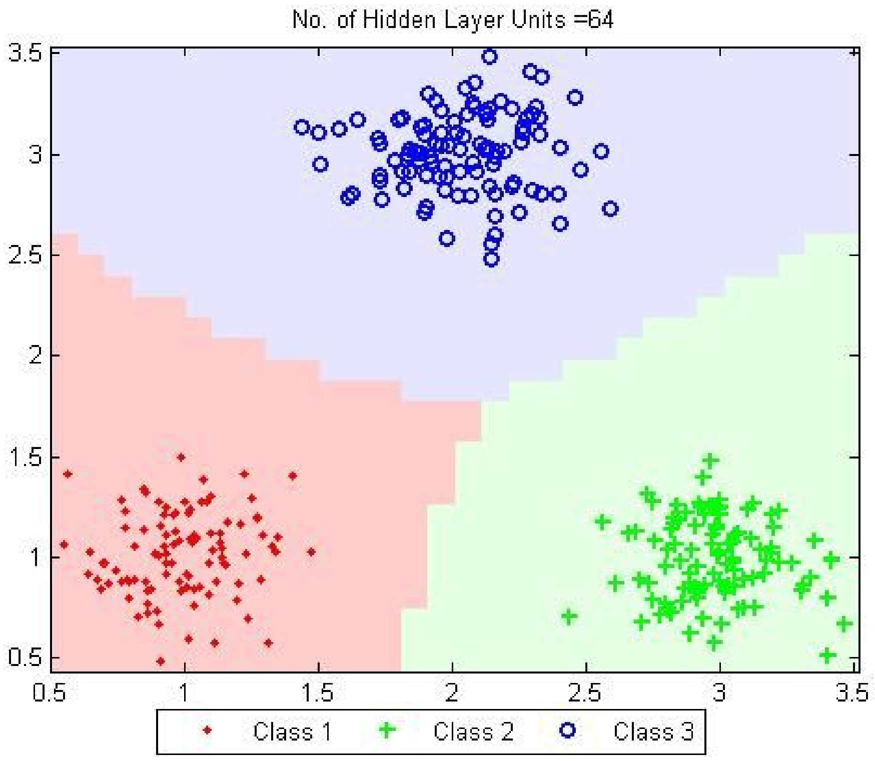

No. of Hidden Layer Units =2
3.5
3
2.5
2
1.5
1
0.5
0.5
1
1.5
2
2.5
3
3.5
Class 1
Class 2
Class 3


No. of Hidden Layer Units =4
3.5
3
2.5
2
1.5
1
0.5
0.5
1
1.5
2
2.5
3
3.5
Class 1
Class 2
Class 3


No. of Hidden Layer Units =8
3.5
3
2.5
2
1.5
1
0.5
1
1.5
2
2.5
3
3.5
Class 1
Class 2
Class 3


No. of Hidden Layer Units =16
3.5
3
2.5
2
1.5
1
0.5
1
1.5
2
2.5
3
3.5
Class 1
Class 2
Class 3


No. of Hidden Layer Units =32
3.5
3
2.5
2
1.5
1
0.5
1
1.5
2
2.5
3
3.5
Class 1
Class 2
Class 3



❖ Following is the error for different number of hidden layer units:

S.No	No. of Hidden Layer Units	Error
1	2	9.7934
2	4	4.3809
3	8	2.9844
4	16	2.5983
5	32	2.0992
6	64	1.7098

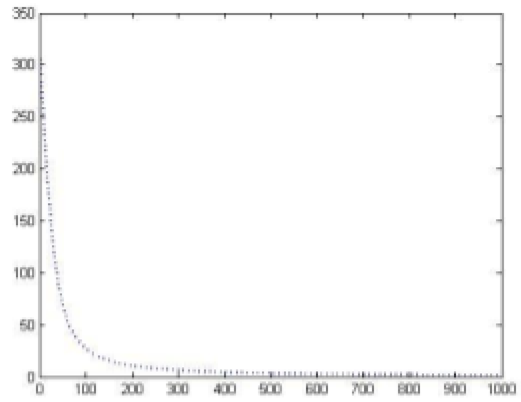
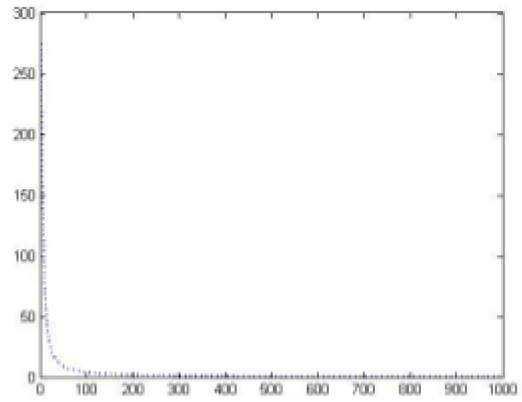
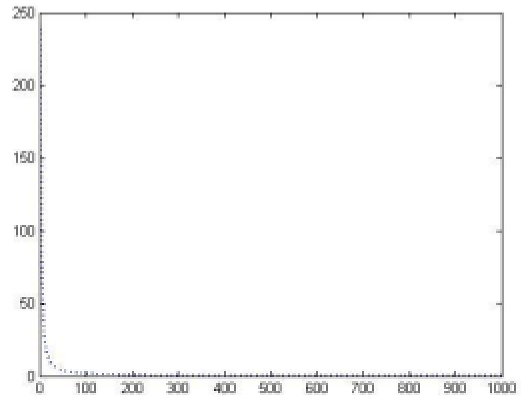
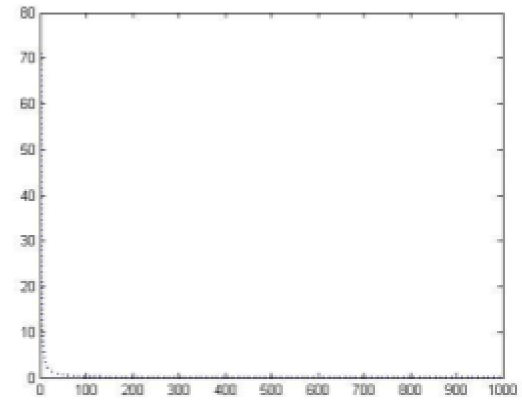
Varying the Learning Rate

- Following observations can be made by varying the learning rate for a model run on 1000 epochs and with 64 hidden layer units:

S.No	Learning Rate	Error
1	0.001	1.6364
2	0.005	0.3724
3	0.01	0.1758
4	0.05	0.0311

- It can be observed that for same number of epochs(1000), convergence is achieved faster on increasing the learning rate. Though high learning rate can also make the model converge to a local minima, this does not happen on increasing the learning rate from 0.001 to 0.05.

- Following are plots for mean squared error vs number of epochs for different learning rates:

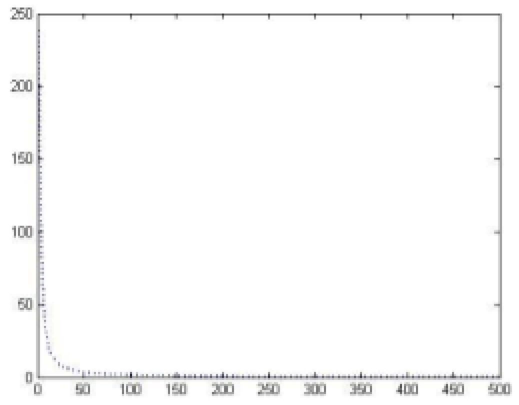
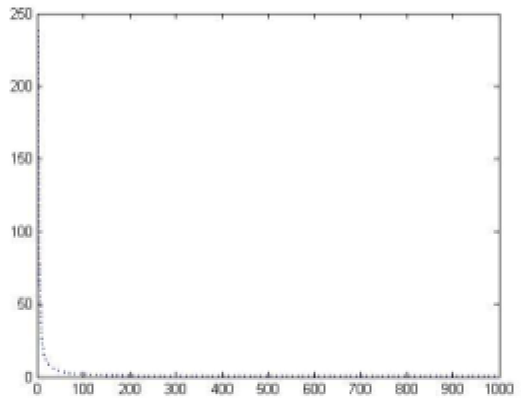
 Learning Rate 0.001	 Learning Rate 0.005
 Learning Rate 0.01	 Learning Rate 0.05

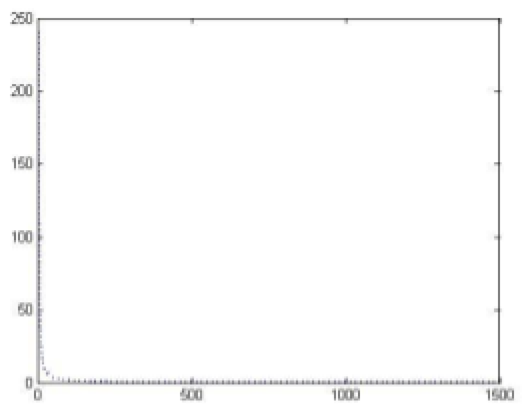
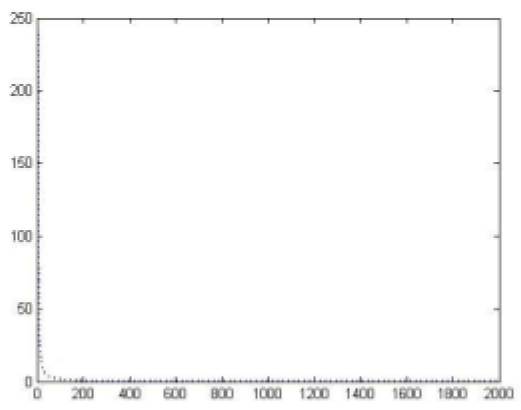
Varying the Number of Epochs

❖ Following observations can be made by varying the number of epochs for a model with learning rate of 0.01 and 64 hidden layer nodes. Error mentioned is mean squared error.

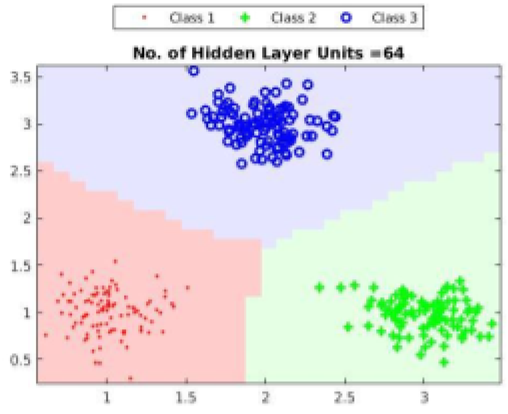
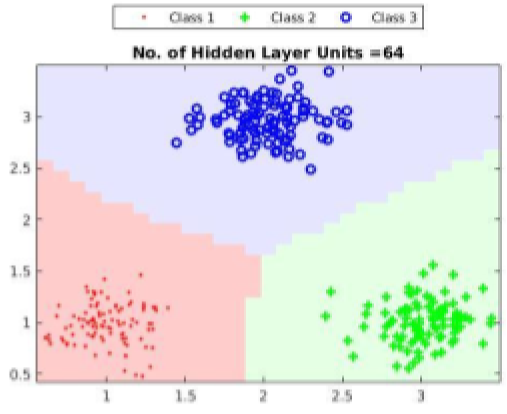
S.No	Number of Epochs	Error
1	500	0.2903
2	1000	0.2124
3	1500	0.1087
4	2000	0.0719

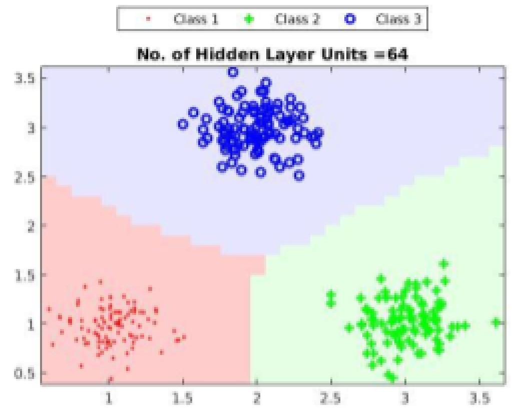
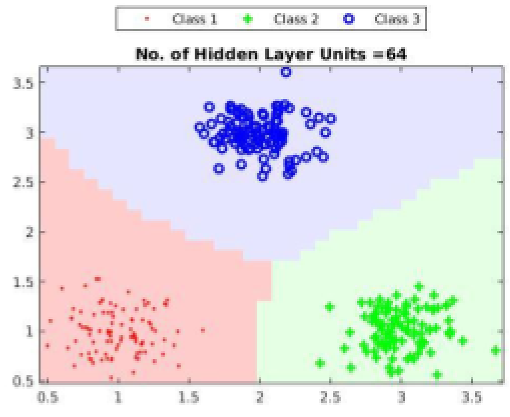
- It can be observed that for this model that error reduces as the number of epochs increases. The reason can be that the extent to which the model gets trained increases and hence the with the updated weights, accuracy on the training set increases.
- Following plots are mean squared error curves wrt number of epochs.

 No. of Epochs -500	 No. of Epochs -1000

 No. of Epochs1500	 No. of Epochs- 2000

❖ Following are plots for decision boundaries plotted by varying the number of epochs:

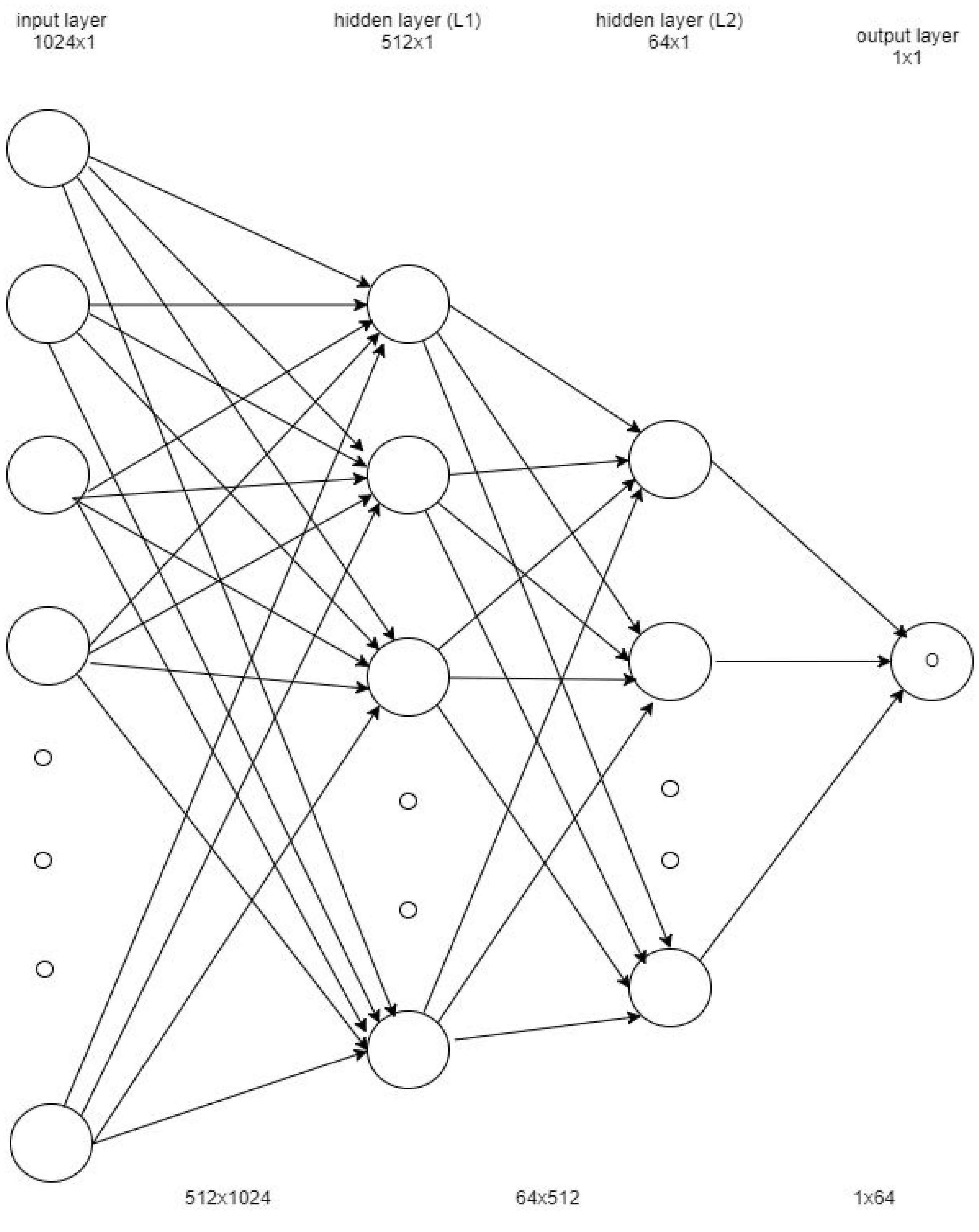
 Number of Epochs=500	 Number of epochs=1500

 Number of epochs=3500	 Number of Epochs=6000

❖ It can be observed that decision boundary becomes more refined as we increase the number of epochs.

EXPERIMENT-2

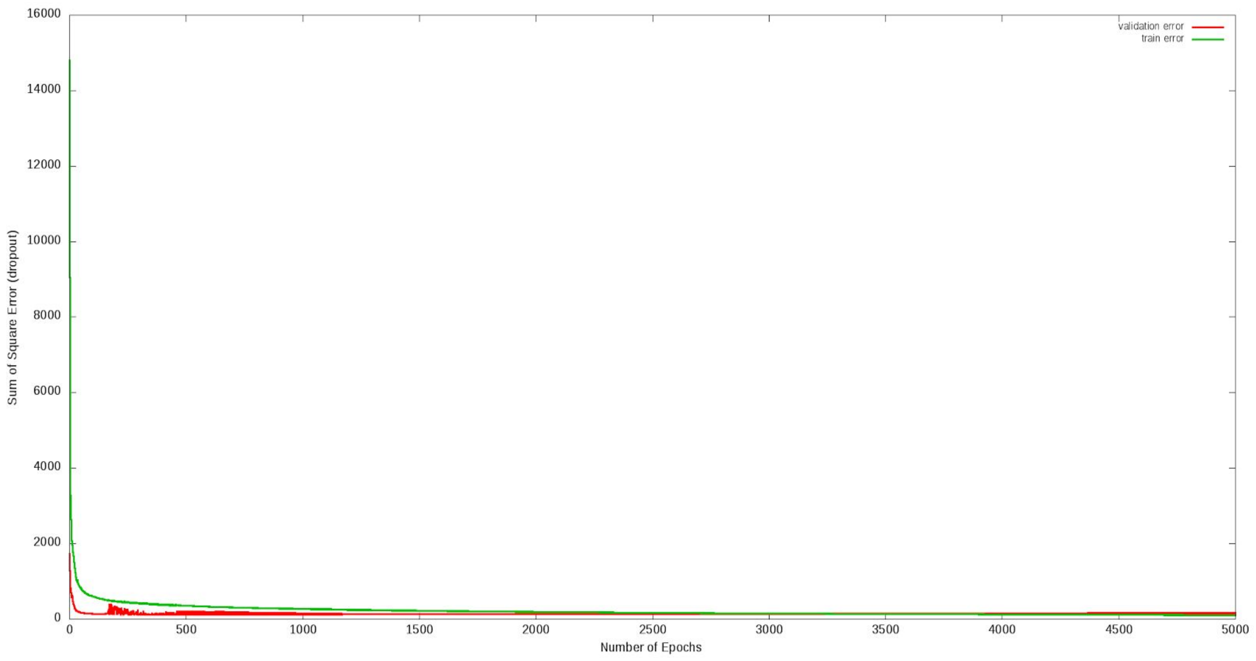
- In this experiment, a basic neural network is implemented to predict the steering angle of the road image for a self-driving car application that is inspired by Udacity's Behavior Cloning Project . This dataset has been compiled from Udacity's simulator.
- From the given data.txt file, the image identifiers and the corresponding angles of all the 22,000 images are stored in fields of a cell in matlab of dimension 1X2. This is done through textscan function in matlab.
- Each original image is converted into a grayscale image from a rgb image. This is part of preprocessing. Original image size is 32x32. Each image pixel values are then reshaped into a matrix of dimension 1024x1.
- Each image pixel value is then normalised by dividing it with the maximum pixel value in the image.
- The architecture employs 1 input layer, 2 hidden layers with number of units as 512 and 64 and an output layer.
- All the hidden layers employ sigmoid activation function, while the output layer does not have any activation function.
- Sum of squares error is used as the loss function in the output layer.
- The weights between the layers are initialized randomly from a normal distribution. The bias terms are initialized to zero.
- The given data set is split into training and validation sets according to 80:20 ratio.


input layer
1024x1
hidden layer (L1)
512x1
hidden layer (L2)
64x1
output layer
1x1
512x1024
64x512
1x64

OBSERVATIONS

1. Observation 1 :

- Sum of squares error is plotted for training and validation set against number of epochs.
- Learning rate is set at 0.01 and number of epochs go till 5000.
- Error is calculated after a mini batch of 64 images is trained by forward propagation and weights are updated by backward propagation.Dropout is not implemented.
- Following is the plot obtained for sum of square errors against the number of epochs:

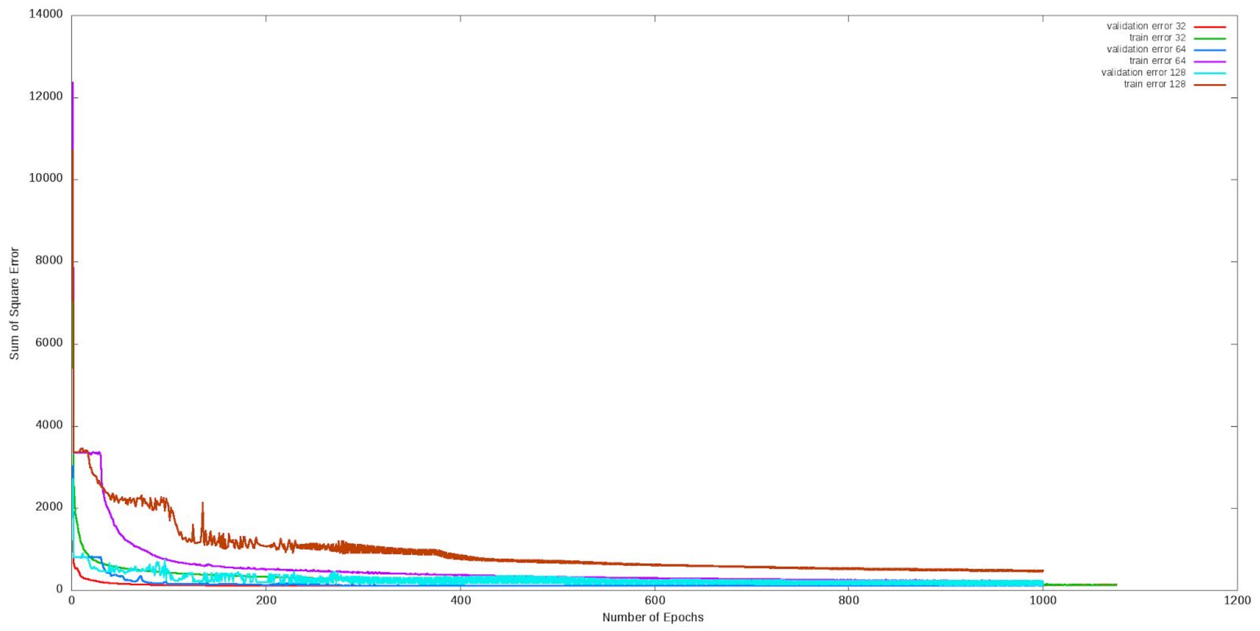


- The red curve represents the sum of squared error per epoch for validation data and the green curve for training data.

- It is observed that for a little more than 3000 epochs, the sum of squares error on validation is a little less than the training error. After that it either increases very slightly or remains equal to the training data.
- This can be due to the reason that the test cases during the validation belonged to the type of data that influenced the model the most while training or maybe because the test cases contain the type of data that our model is good at predicting. But since there is a very small difference, it could be random.
- Also since the validation error is less, the model is neither underfitted nor overfitted.
- The model improves with more epochs of training, to a point. After that hardly any change in accuracy is obtained as the model starts to converge.

2. Observation 2:

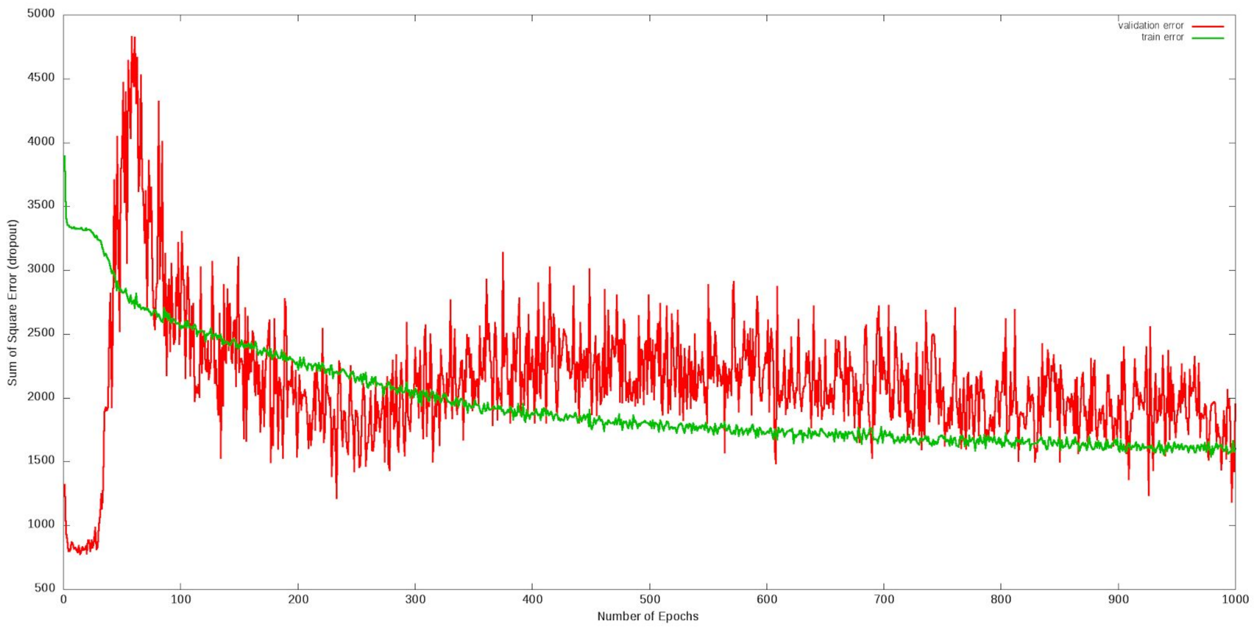
- Training and validation data is sent in mini batches of size 32,64 and 128.
- Learning rate is set at 0.01 and number of epochs go till 1000.
- Following is the plot obtained for sum of square errors against the number of epochs:



- Brown curve- train error on mini batch size 128
  Light blue curve- validation error on mini batch size 128
  Pink curve- train error on mini batch size 64
  Dark blue curve- validation error on mini batch size 64
  Green curve- train error on mini batch size 32
  Red curve- validation error on mini batch size 32
- For batch size 128,sum of squares error for training data is more than that for the validation data.This can be due to the reason that the test cases during the validation belonged to the type of data that influenced the model the most while training or maybe because the test cases contain the type of data that our model is good at predicting.
- For batch size 64,again error on training data is more than that on validation data.
- For batch size 32, again error on training data is more than that on validation data.
- Therefore it can be concluded that overall in the model, error on training data is more than that in validation error.
- In addition to the above mentioned points, reason can be that the training data has mostly “hard” cases whereas validation data has mostly “easy” cases.
- Due to low validation error, model is neither overfit nor underfit.
- It can also be observed that in this model, best performance is achieved with mini batch size of 32, then of 64 and then of 128. This can be due to the reason with mini batch size of 32/64, enough noise is injected to each weight update while achieving a relative speedy convergence.

3. Observation 3 :

- Learning rate is set at 0.001 and number of epochs go till 1000.
- Dropout is implemented with probability of 0.5 in the first, second and third layer.
- Following is the plot obtained for sum of square errors against the number of epochs:

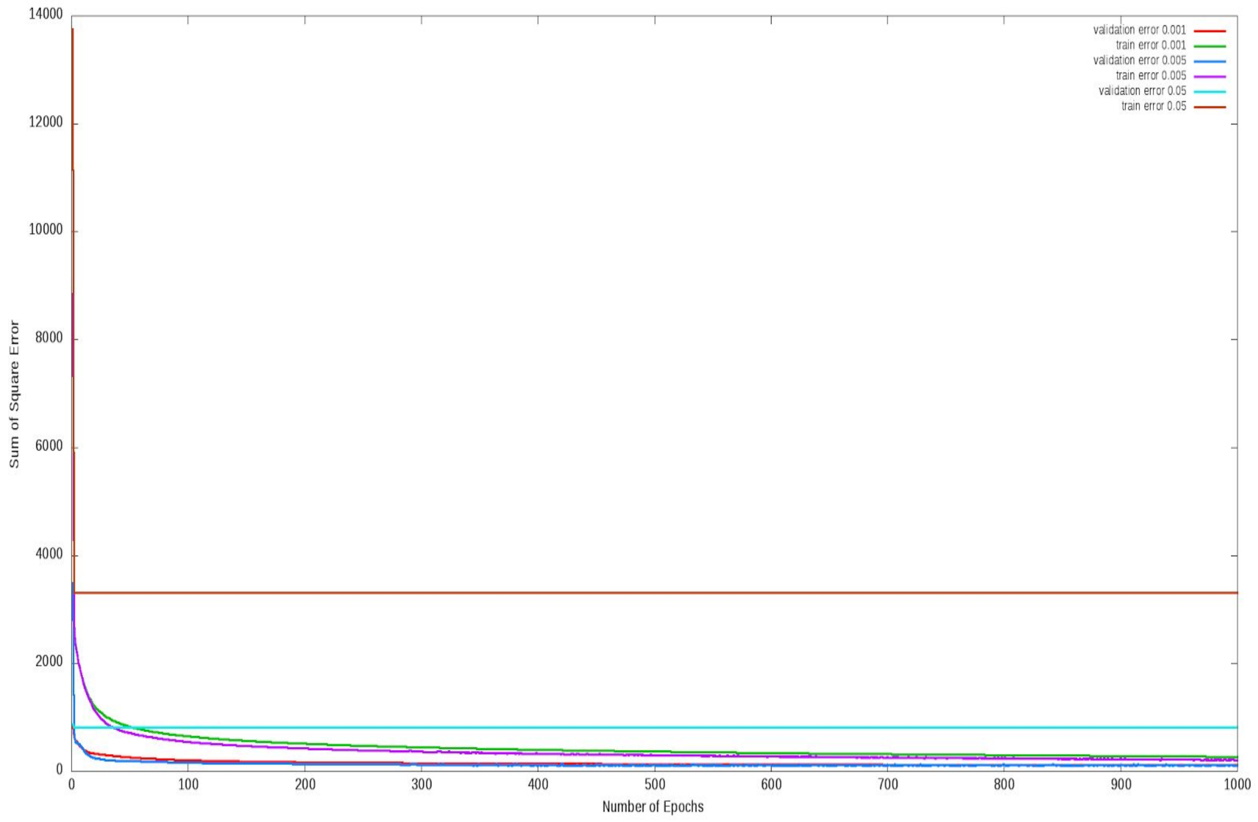


- It is observed that on implementing dropout, there is a lot of oscillation in the value of sum of square error with the increasing number of epochs.

4.Observation 4:

- Sum of squares error is plotted for training and validation set against number of epochs.
- Observation is taken for different learning rates , 0.01,0.005 and 0.05. Number of epochs go till 1000.
- Error is calculated after a mini batch of 64 images is trained by forward propagation and weights are updated by backward propagation.Dropout is not implemented.

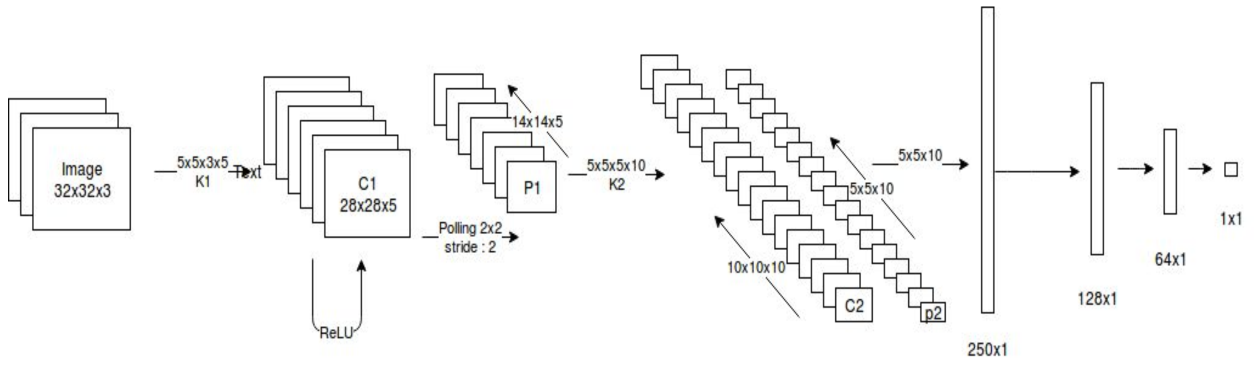
- Following is the plot obtained for sum of square errors against the number of epochs:



- Brown curve- train error with learning rate as 0.05.
  Light blue curve- validation error with learning rate as 0.05.
  Purple curve- train error with learning rate as 0.005.
  Dark blue curve- validation error with learning rate as 0.005.
  Green curve- train error with learning rate as 0.001.
  Red curve- validation error with learning rate as 0.001.

- It can be observed that with a high learning rate like 0.05, in case of training data, convergence is obtained very fast but it converges to a local minima and not a global minima. This can be a drawback of choosing a high learning rate.

Convolutional Neural Networks



We implemented CNN also. Following is the underlying architecture:

- Input image is of size 32x32x3.
- Convolution is applied on it with 5 filters of dimension 5x5x3 each. Therefore image size now becomes 28x28x5.
- After than Relu activation function is applied and size of image remains the same.
- Then max polling is done on the image with stride 2 and window size of 2x2. Image size will now become 14x14x5.
- Convolution is again performed on this image with 10 filters of size 5x5x5.Image size now becomes 10x10x10.
- Relu activation function is applied.
- Then again max polling is done on the image with stride 2 and window size of 2x2. Image size will now become 5x5x10.
- After that this image of size 5x5x10 is reshaped into a 1D matrix of dimension 250x1.
- After this 2 fully connected hidden layers are implemented with number of nodes being 128 and 64, which is followed by an output layer consisting of only 1 node.
- Sigmoid activation function is used in the hidden layers. No activation function in used in the output layer.

We are not using CNN for competition part since we were able to run it only for 100 epochs in which mean square error came out to be 1.091. Due to shortage of time, we couldn’t train it further and use it for competition.

For Competition Part

- The architecture employs 1 input layer, 2 hidden layers with number of units as 128 and 64 and an output layer.
- All the hidden layers employ sigmoid activation function, while the output layer does not have any activation function.
- Sum of squares error is used as the loss function in the output layer.
- The weights between the layers are initialized randomly from a normal distribution. The bias terms are initialized to one.
- Mini batch of 32 images is used.
- Learning Rate is 0.01.
- Number of epochs at which model is trained=
- The weights trained after 1000 epochs are used to generate the angles for the test file given.
- The weights used for this purpose are included in file competition.txt