ASSIGNMENT-3

NEURAL NETWORKS

- **Pre-processing the data**

  The given data was first separated into the design matrix and the label vector. The design matrix was then range normalised. The label vector was one-hot encoded and thus converted into a (n x 10) matrix, where n is the number of training examples and 10 comes because of the number of labels.

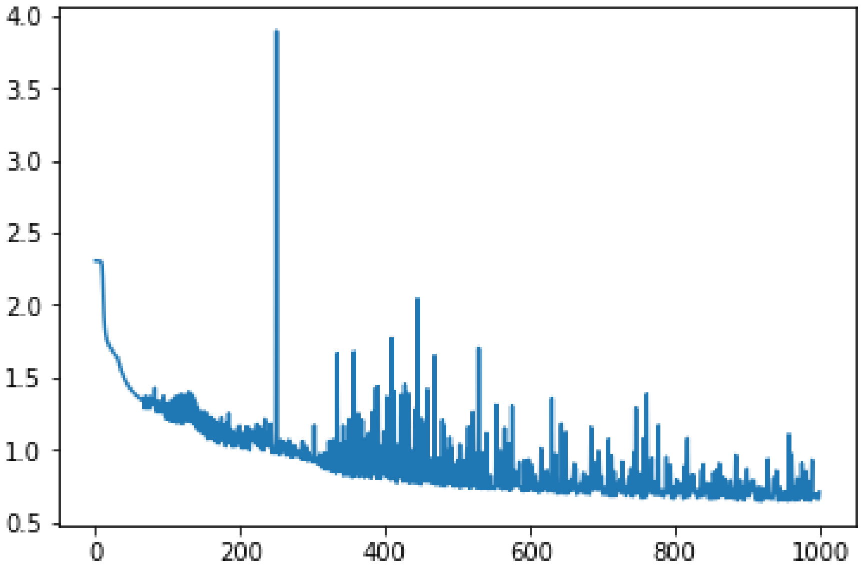
- **Coding the neural-network**

  Data pre-processing done, now neural network with backward propagation was built. Forward propagation allowed the use of various activation functions, namely sigmoid, tanh, relu and leaky relu. Finally, the output layer was generated through softmax activation function. Backward propagation used gradient descent with batch-size, learning rate and number of iterations variable. Finally, above all was wrapped in validation function with 80:20 train:test split. Cross-entropy was used to build the loss function. Weights were initialised using He-initialisation.

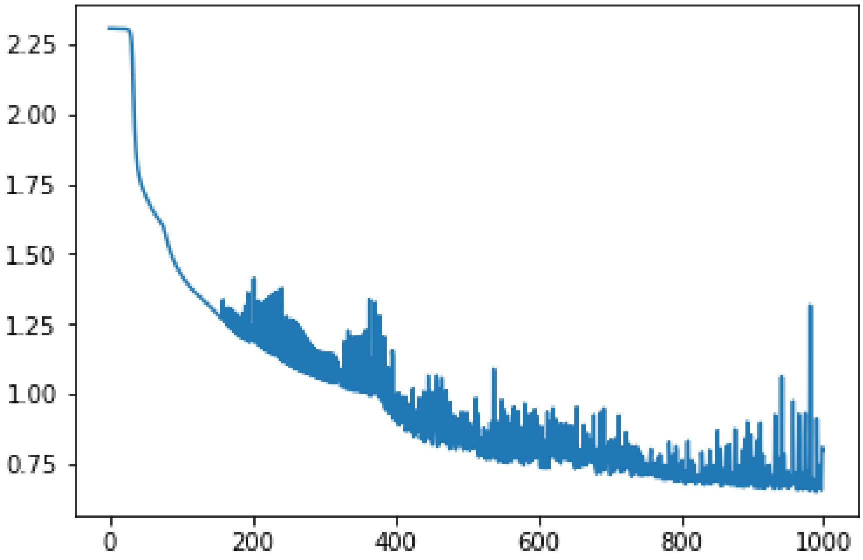
- **Setting gradient descent hyperparameters**

  First, learning rate was tuned. Different learning rates were tried and the loss was plotted vs the number of iterations. The plots are shown below. For this exercise, the number of iterations was set to 1000.

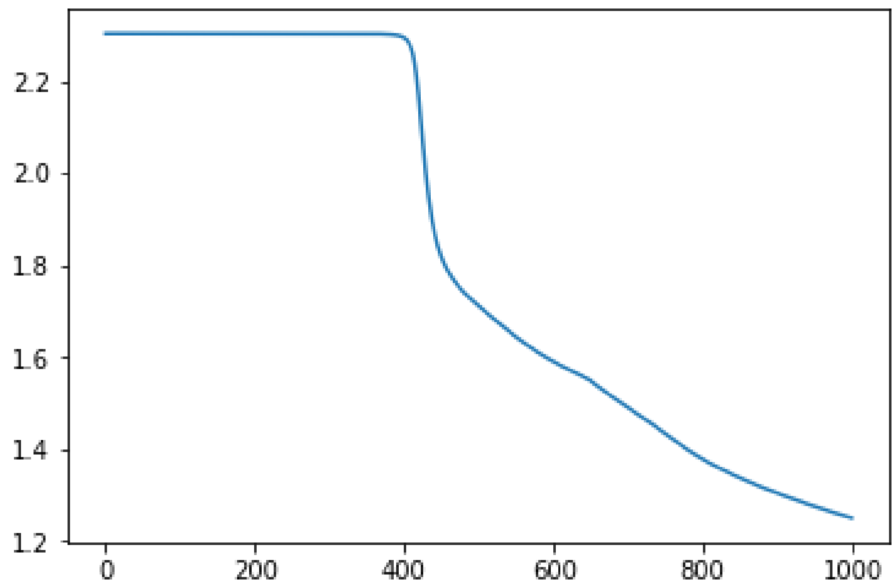
Learning-rate = 1



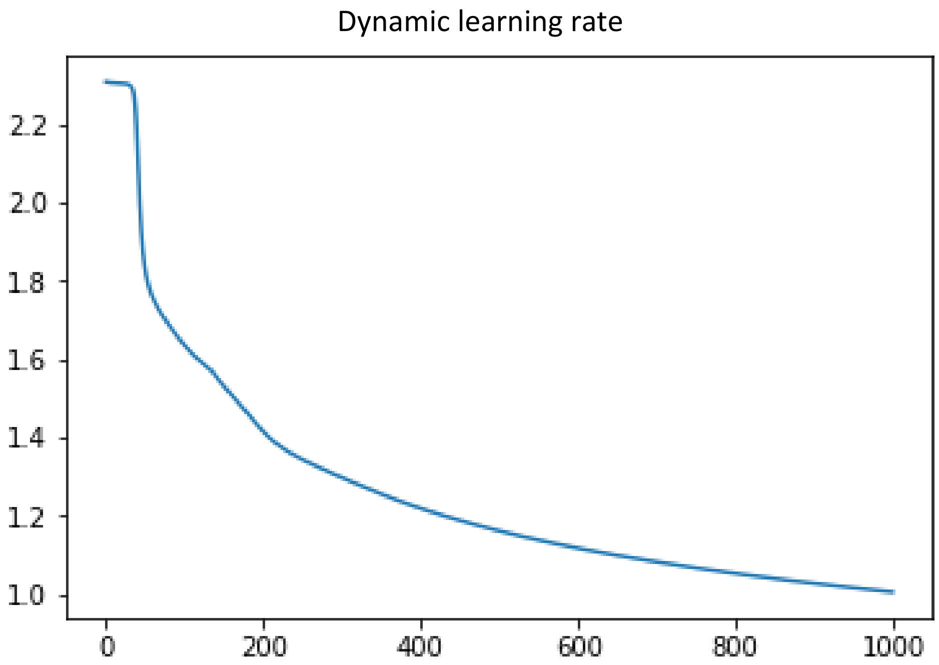
Learning-rate = 0.5



Learning-rate = 0.1

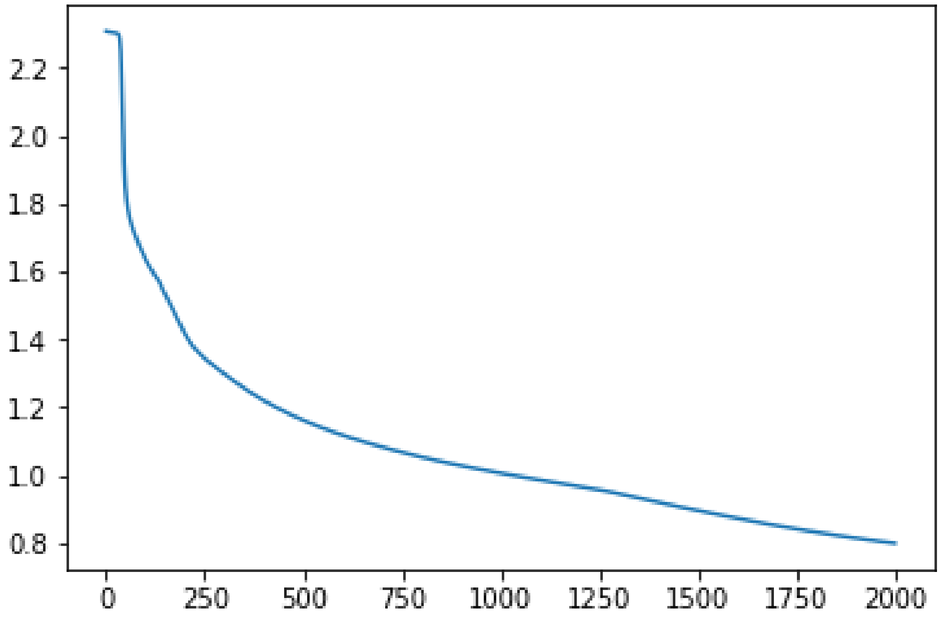


From above graphs, we see that alpha = 0.1 is a good choice, but is still not optimum. So, a dynamic learning rate was adopted starting from 1 and its value was changed every iteration by dividing it by cube-root of the current iteration number. For example, if nth iteration is going on, the value of learning rate would be 1.0/(n^(1/3)). The result was:

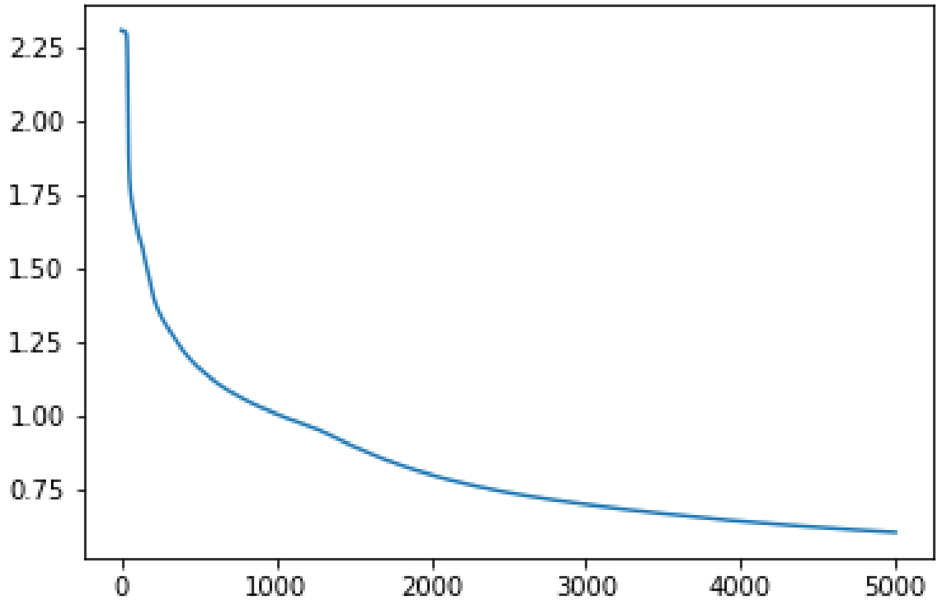


Much better! Now we move onto number of iterations and batch-size.

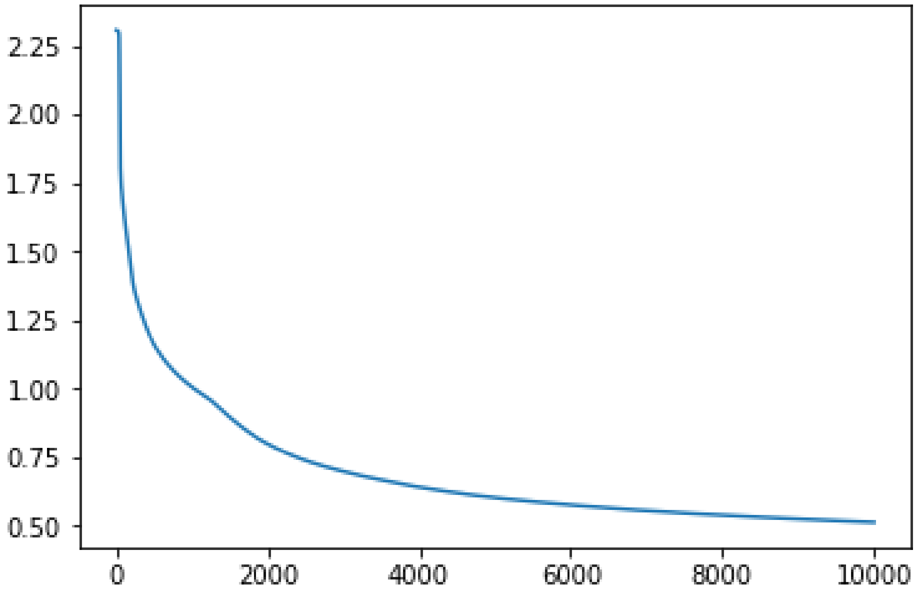
Iterations = 2000



Iterations = 5000



Iterations = 10000

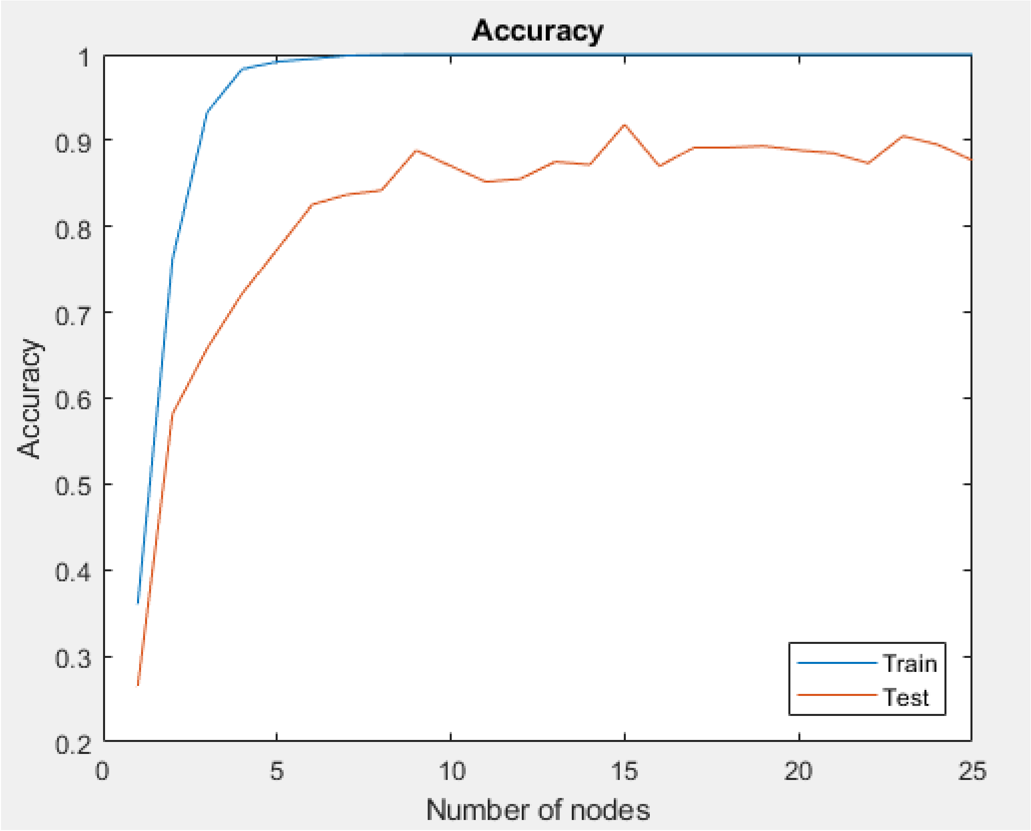


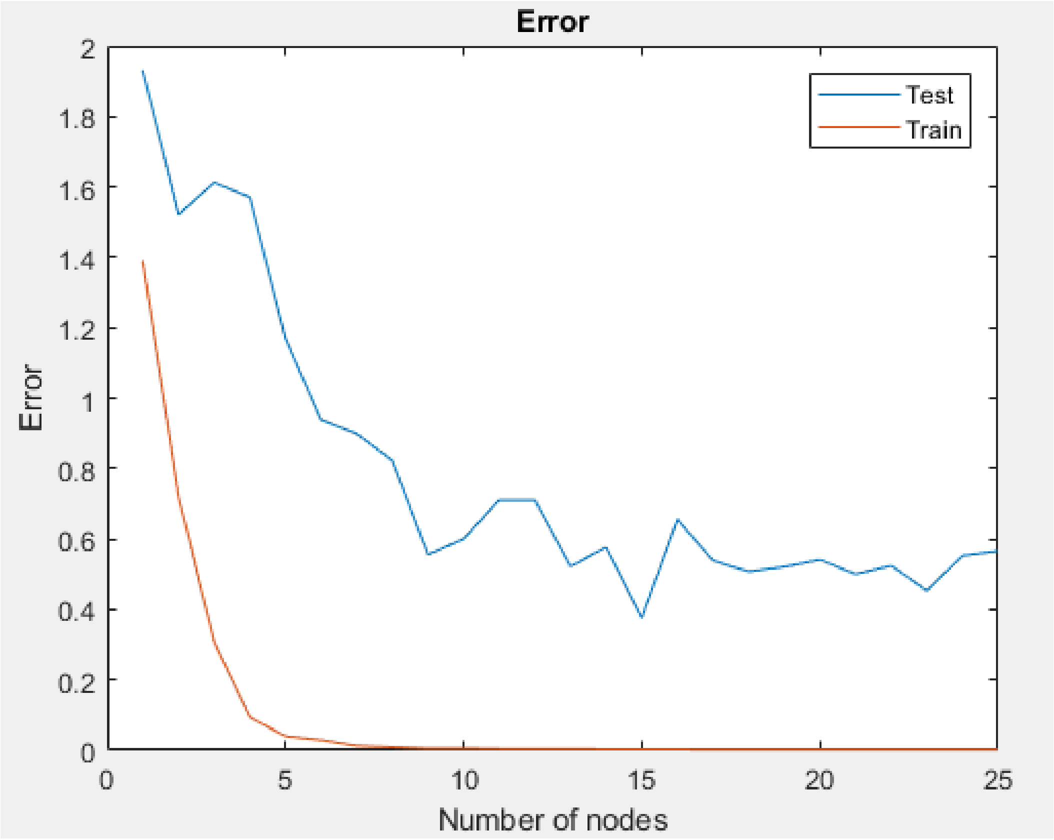
At 5000 iterations, the change in loss becomes of the order of 10^(-3). The optimal batch-size was found out to be 100, keeping in mind the speed of convergence and the final error after given iterations.
Hence, the final gradient descent parameters are:
Learning-rate = **Dynamic starting at 1.0 and decaying as the cube-root of the number of iteration currently going on**
Number of iterations = **5000**
Batch-size = **100**

- **Setting the hidden layers and activation function**

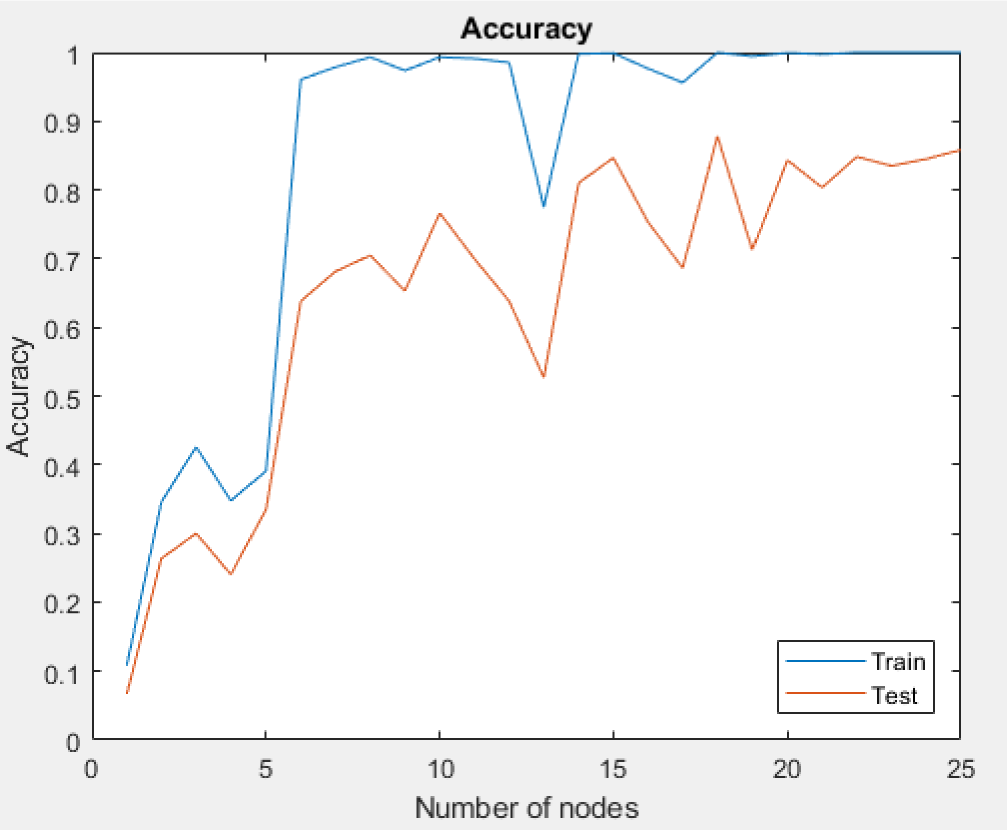
In this section, the number of layers and the nodes in each layer are tuned with different activation functions.
Firstly, only one hidden layer is used and the number of nodes in this layer are varied from 1 to 25. The accuracies and error are plotted below.

Sigmoid

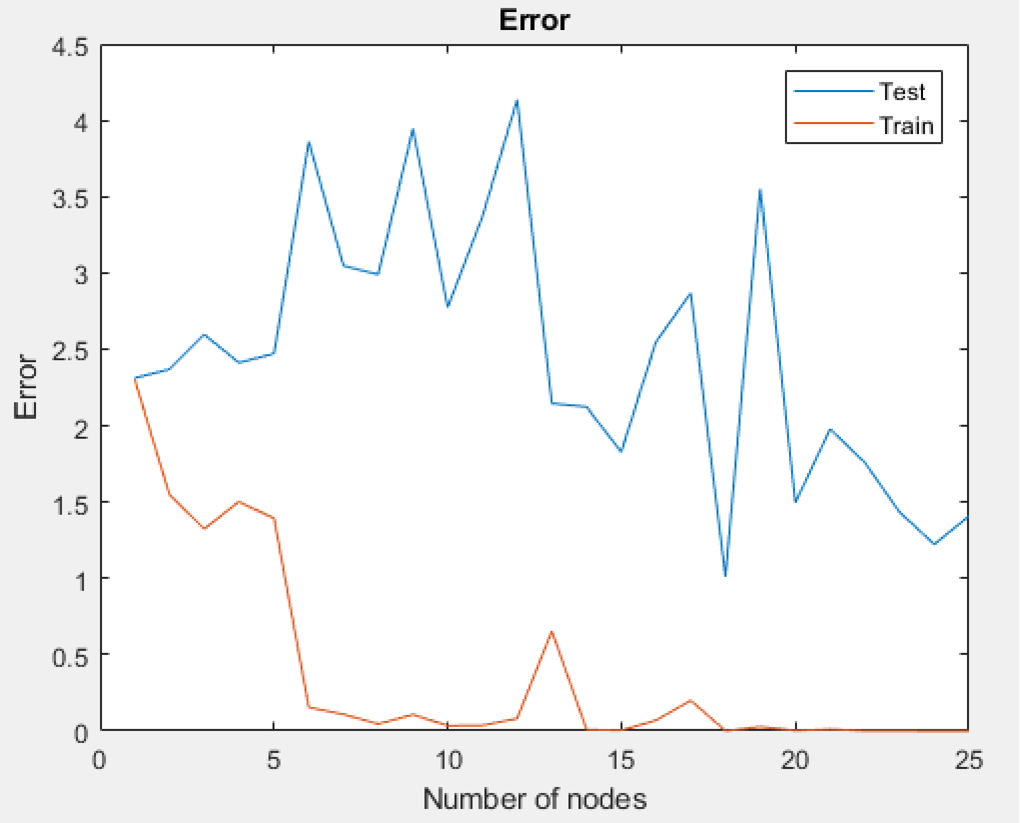

Accuracy
Accuracy
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
Number of nodes
Train
Test


Error
Error
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
Number of nodes
Test
Train

ReLu

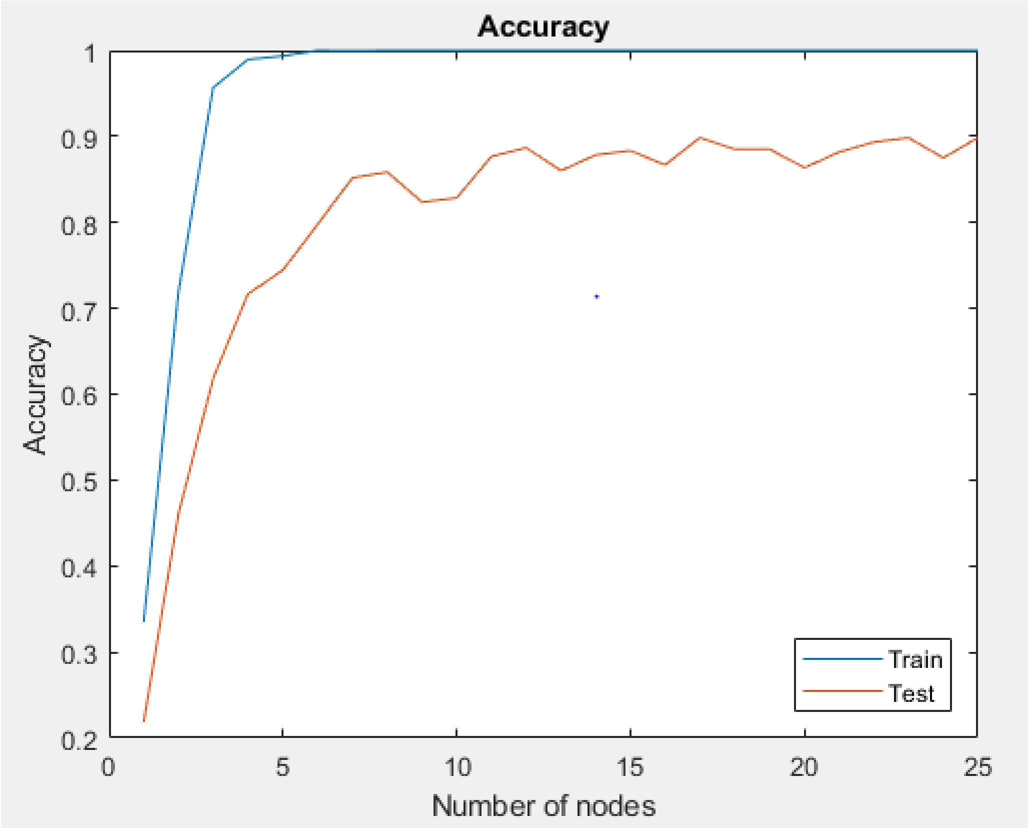


Accuracy
Accuracy
Number of nodes
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
0
5
10
15
20
25
Train
Test

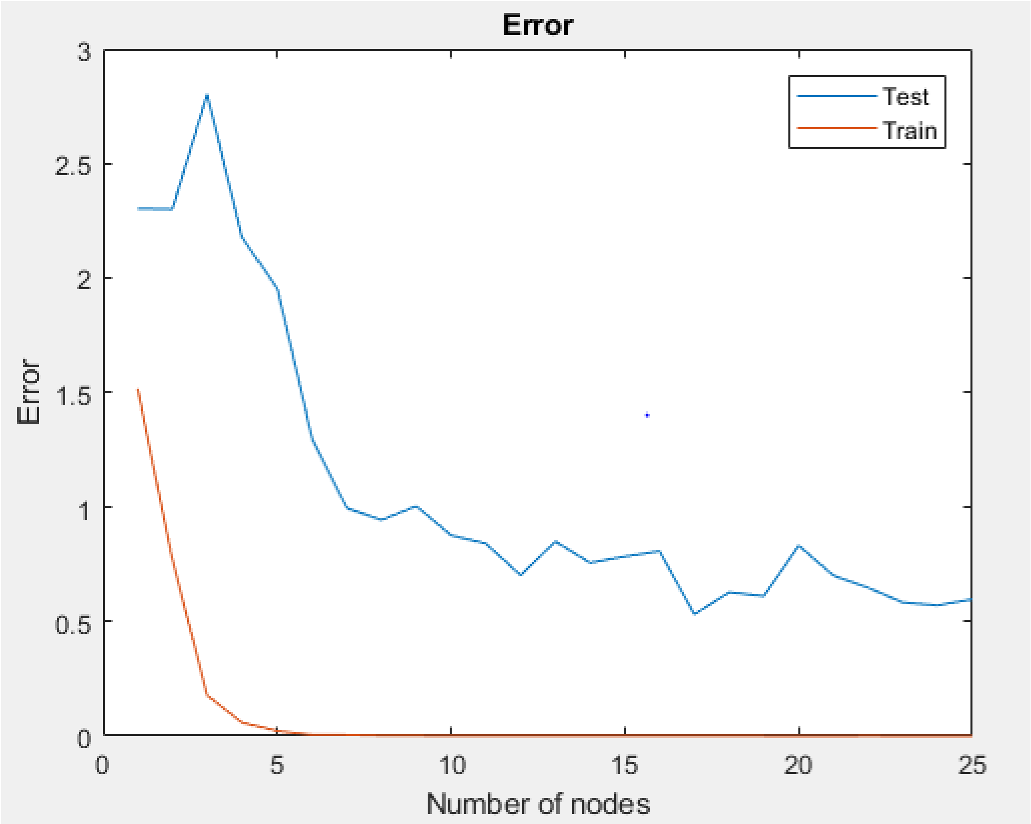


Error
Error
Number of nodes
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
5
10
15
20
25
Test
Train

Tanh

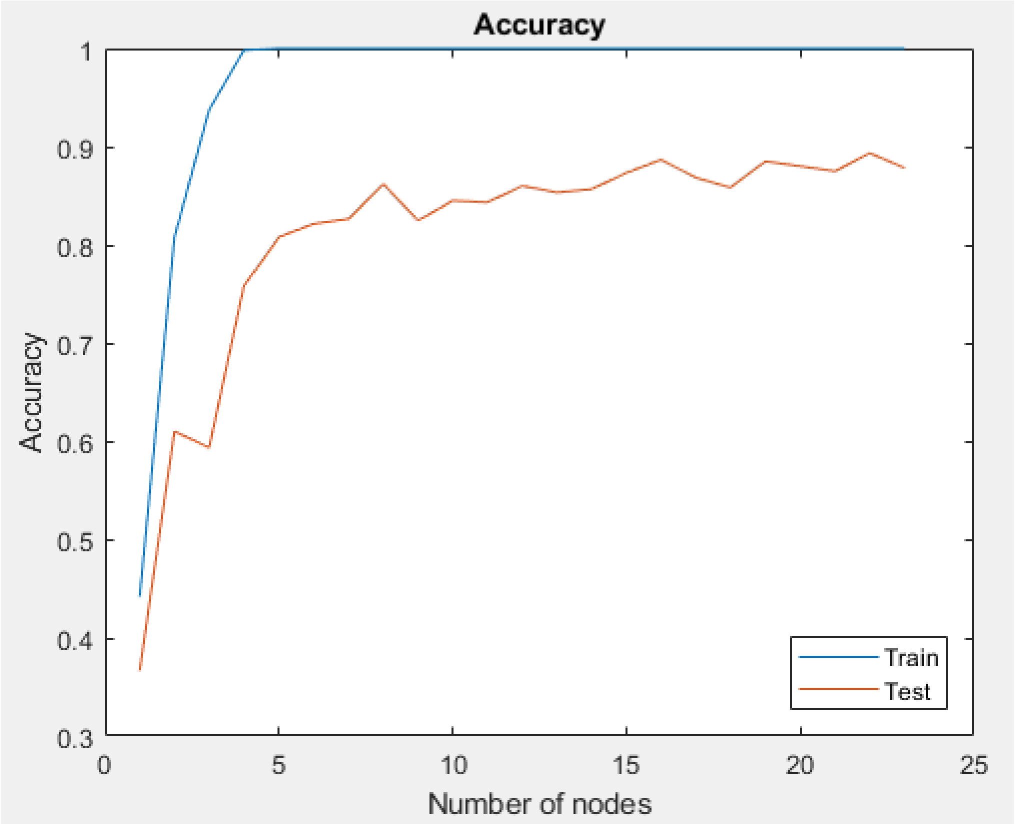


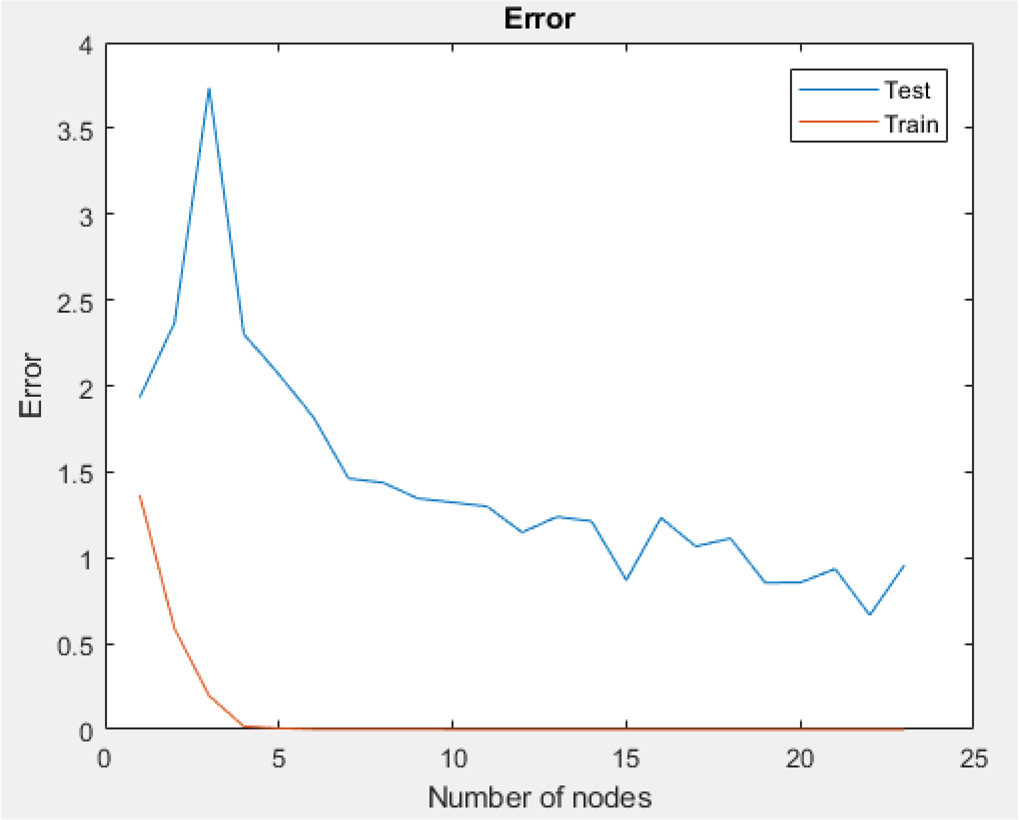
Accuracy
Accuracy
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
5
10
15
20
25
Number of nodes
Train
Test



Error
Error
3
2.5
2
1.5
1
0.5
0
0
5
10
15
20
25
Number of nodes
Test
Train

Leaky ReLu

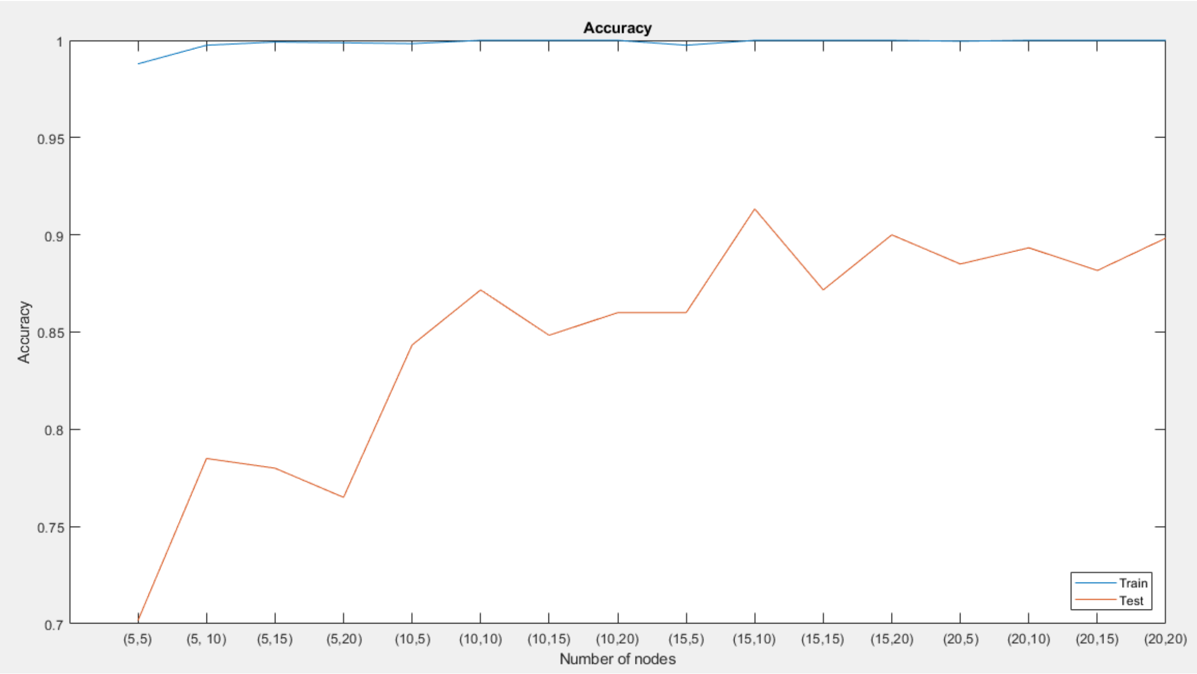

Accuracy
Accuracy
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
5
10
15
20
25
Number of nodes
Train
Test

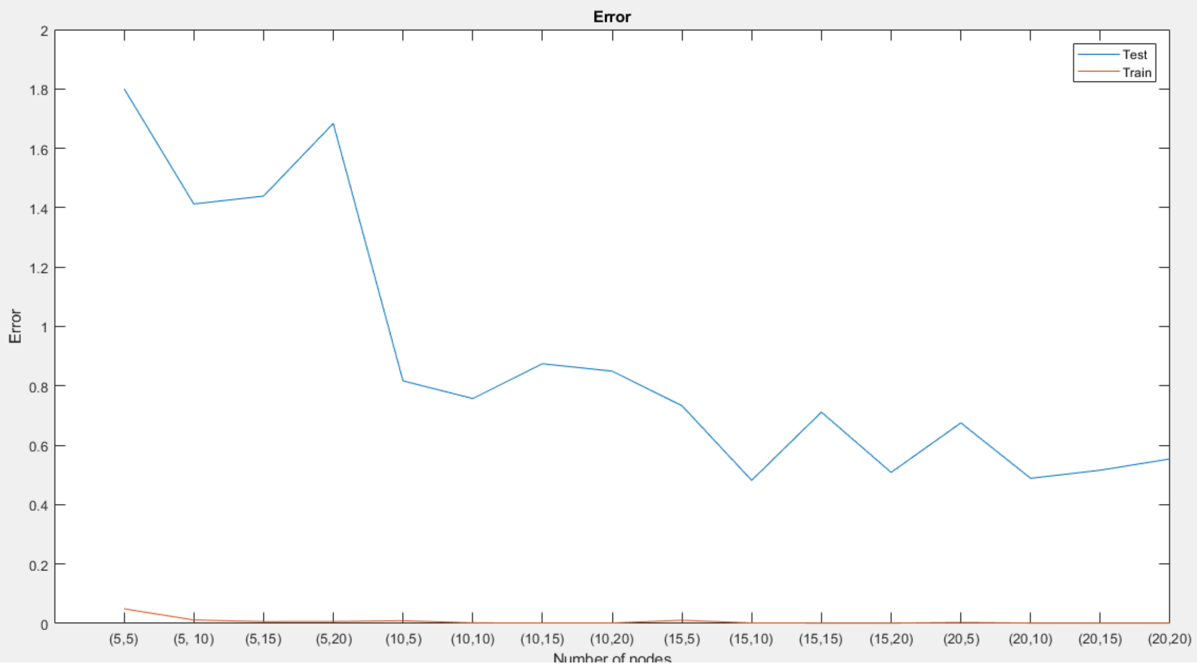

Error
Error
4
3.5
3
2.5
2
1.5
1
0.5
0
0
5
10
15
20
25
Number of nodes
Test
Train

To summarise, we see that highest validation accuracy is attained by **sigmoid** activation function with **15** nodes, the value of which is **0.918333** and the corresponding error is **0.37548608**. The time taken to run each hyperparameter setting was **152** sec on average.

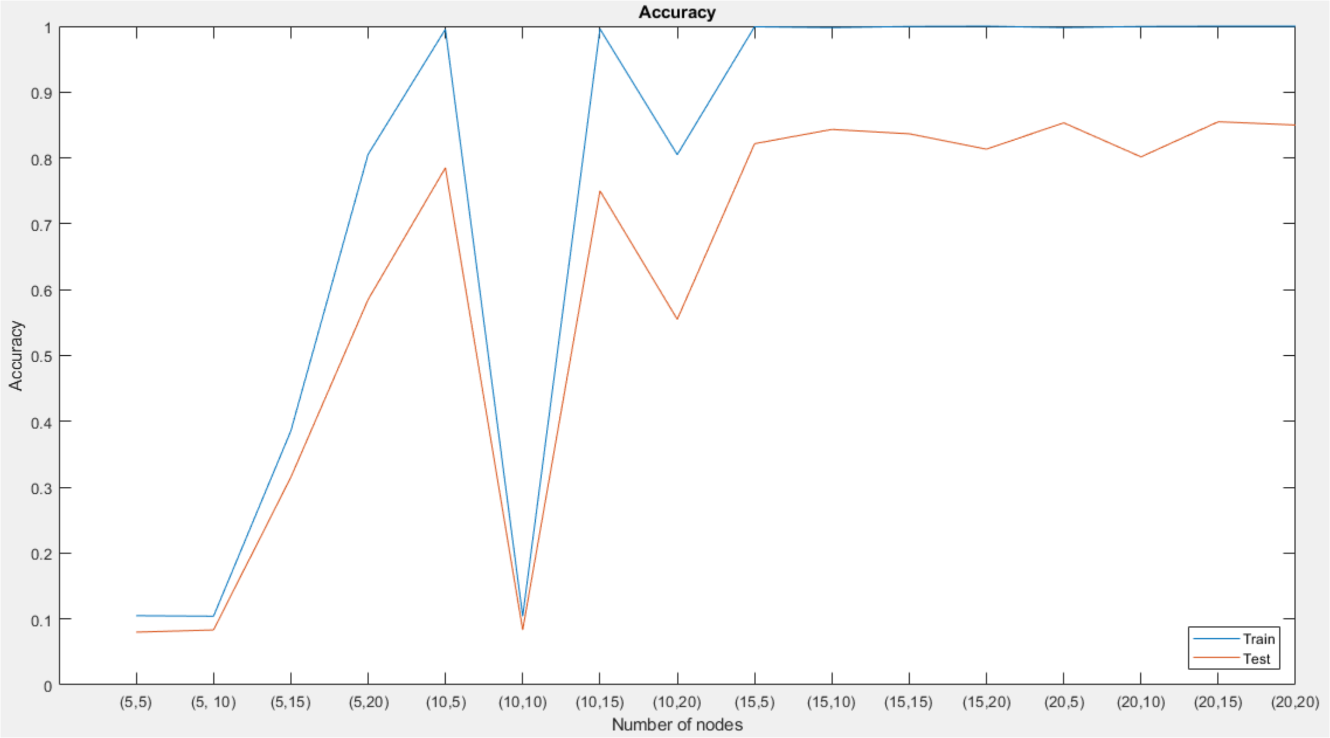
Now we move onto two hidden layers.

Sigmoid

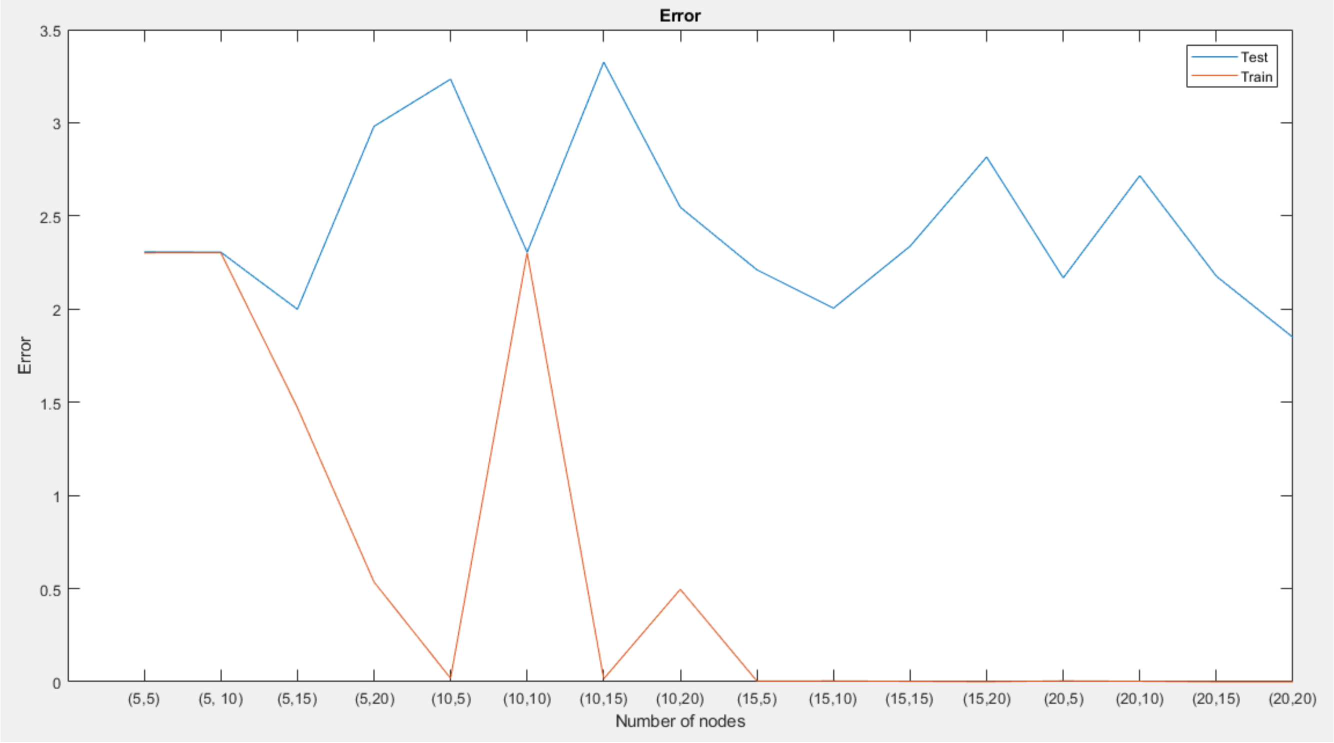




ReLu

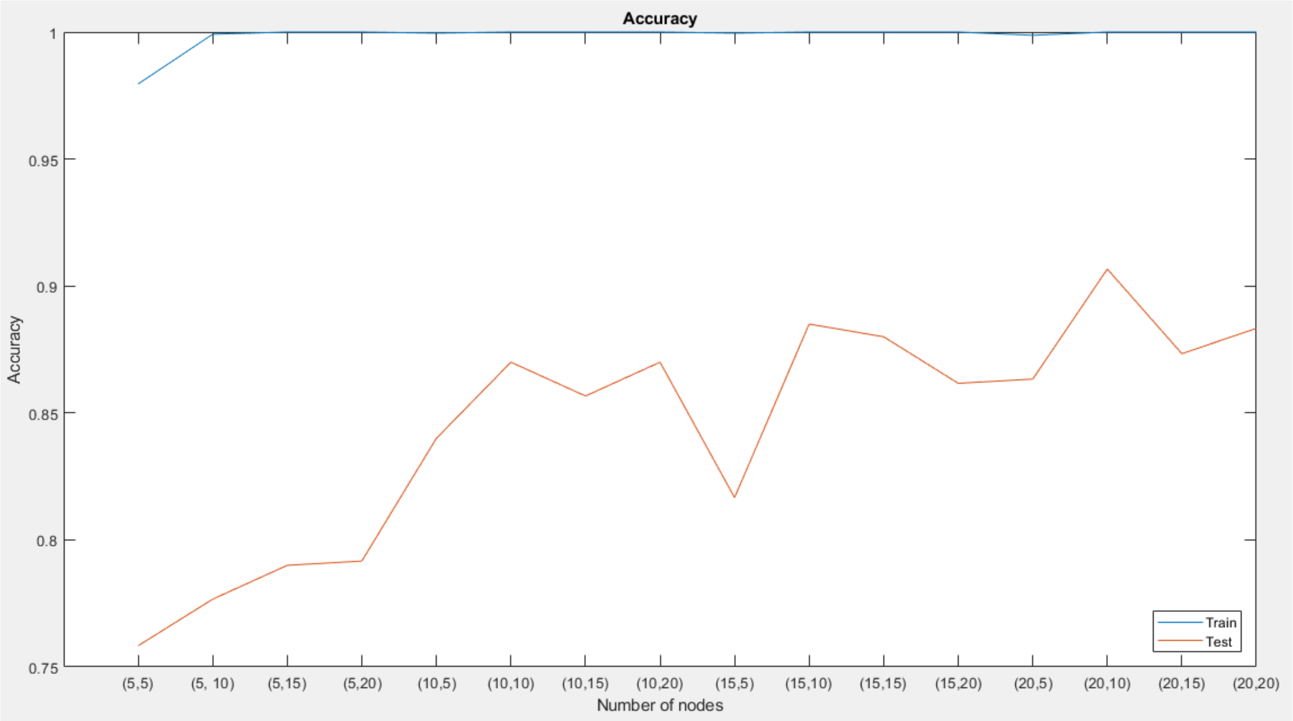


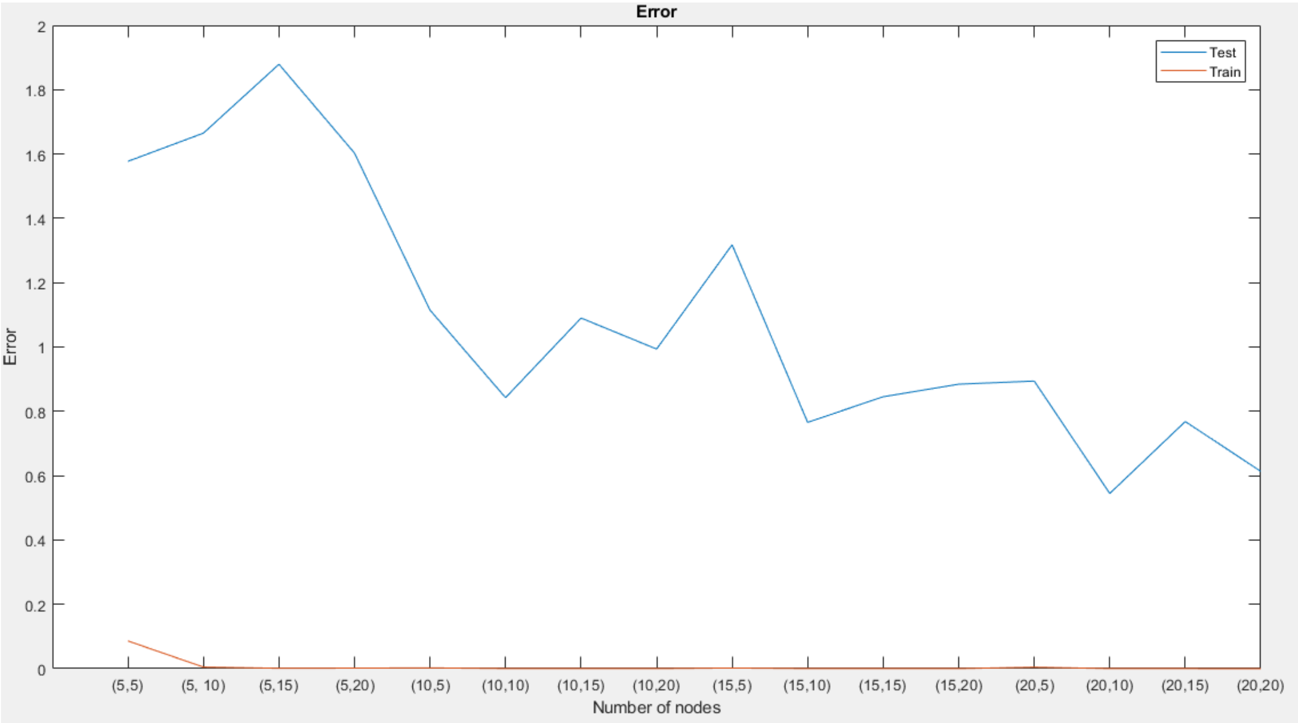
Accuracy
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Accuracy
(5,5)
(5, 10)
(5,15)
(5,20)
(10,5)
(10,10)
(10,15)
(10,20)
(15,5)
(15,10)
(15,15)
(15,20)
(20,5)
(20,10)
(20,15)
(20,20)
Number of nodes
Train
Test



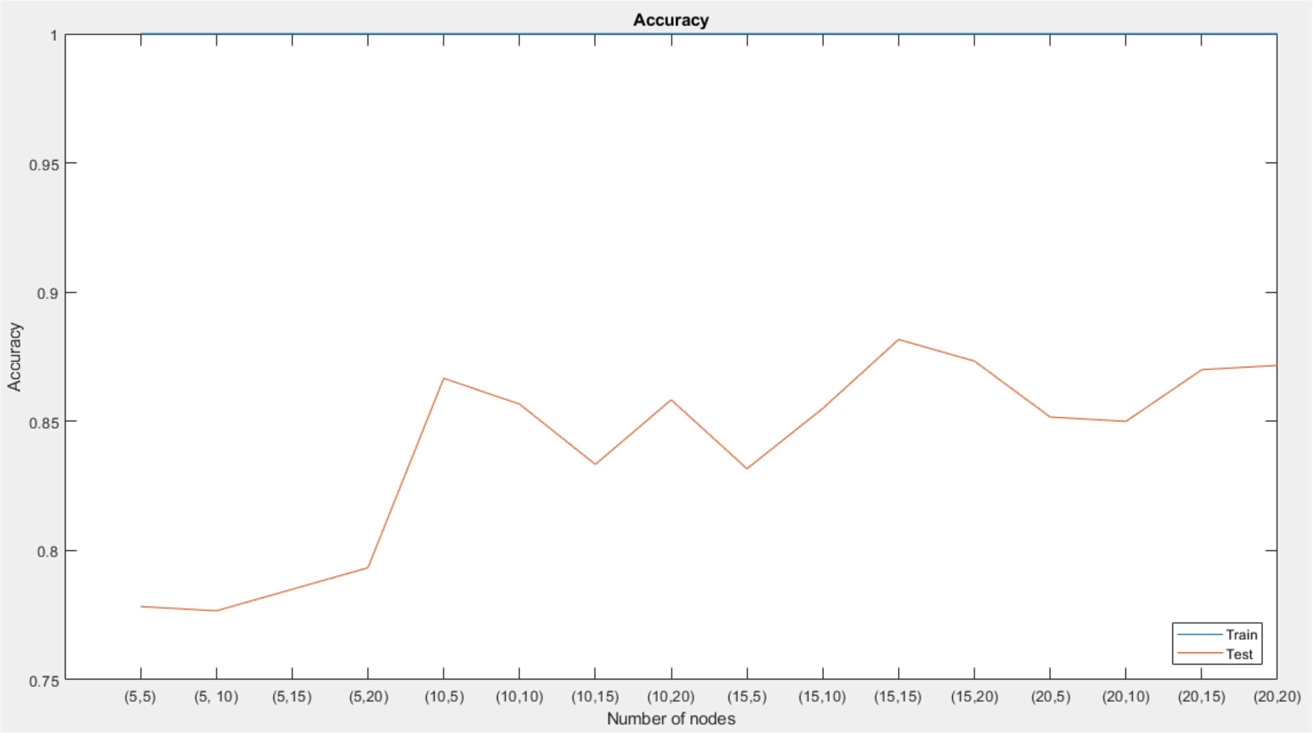
Error
3.5
3
2.5
2
1.5
1
0.5
0
Error
(5,5)
(5, 10)
(5,15)
(5,20)
(10,5)
(10,10)
(10,15)
(10,20)
(15,5)
(15,10)
(15,15)
(15,20)
(20,5)
(20,10)
(20,15)
(20,20)
Number of nodes
Test
Train

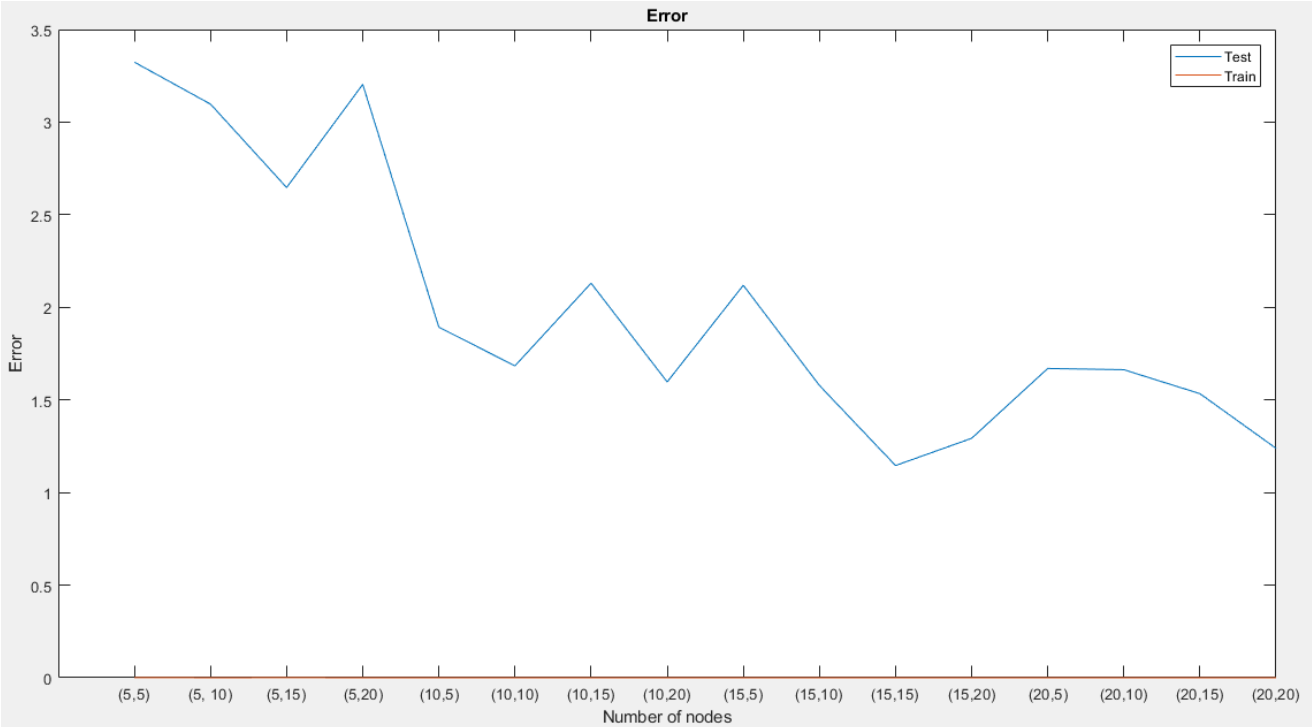
Tanh


Accuracy
1
0.95
0.9
0.85
0.8
0.75
Accuracy
(5,5)
(5, 10)
(5,15)
(5,20)
(10,5)
(10,10)
(10,15)
(10,20)
(15,5)
(15,10)
(15,15)
(15,20)
(20,5)
(20,10)
(20,15)
(20,20)
Number of nodes
Train
Test


Error
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Error
(5,5)
(5, 10)
(5,15)
(5,20)
(10,5)
(10,10)
(10,15)
(10,20)
(15,5)
(15,10)
(15,15)
(15,20)
(20,5)
(20,10)
(20,15)
(20,20)
Number of nodes
Test
Train

Leaky Relu


Accuracy
Accuracy
1
0.95
0.9
0.85
0.8
0.75
(5,5)
(5, 10)
(5,15)
(5,20)
(10,5)
(10,10)
(10,15)
(10,20)
(15,5)
(15,10)
(15,15)
(15,20)
(20,5)
(20,10)
(20,15)
(20,20)
Number of nodes
Train
Test


Error
Error
3.5
3
2.5
2
1.5
1
0.5
0
(5,5)
(5, 10)
(5,15)
(5,20)
(10,5)
(10,10)
(10,15)
(10,20)
(15,5)
(15,10)
(15,15)
(15,20)
(20,5)
(20,10)
(20,15)
(20,20)
Number of nodes
Test
Train

To summarise, we see that highest validation accuracy is attained by **sigmoid** activation function with (**15, 10)** nodes, the value of which is **0.913333** and the corresponding error is **0.481964**. The time taken to run each hyperparameter setting was **155** sec on average.

Hence, we see that the best setting is **sigmoid** activation function with **one** hidden layer, with the layer consisting **15** nodes.

- **Regularisation**

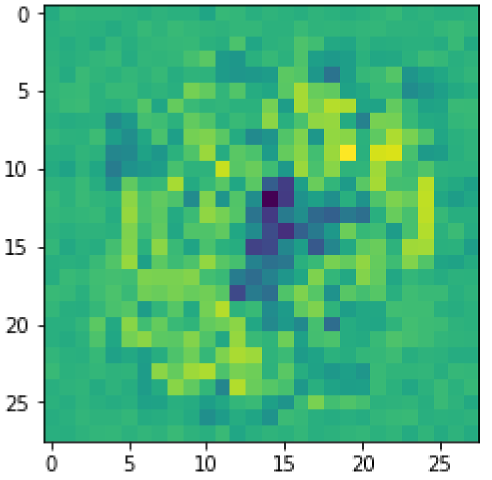
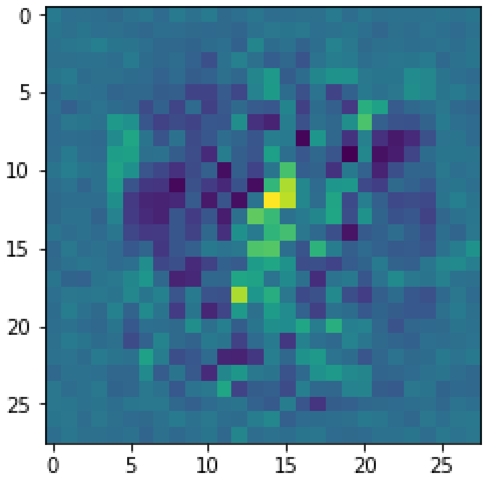
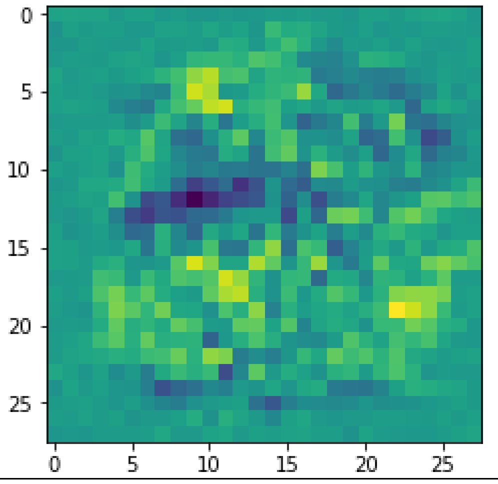
  I carried out L2 regularisation and the value of lambda was chosen to be **0.01** after a few runs with different values. The best result we got above was tested with regularisation. The results:
  Train accuracy: **New – 1.0**; **Old – 1.0**;
  Test accuracy: **New – 0.8923**; **Old – 0.9183**
  Train loss: **New – 8.111e-06**; **Old – 0.000337**
  Test loss: **New – 0.6381**; **Old – 0.3754**

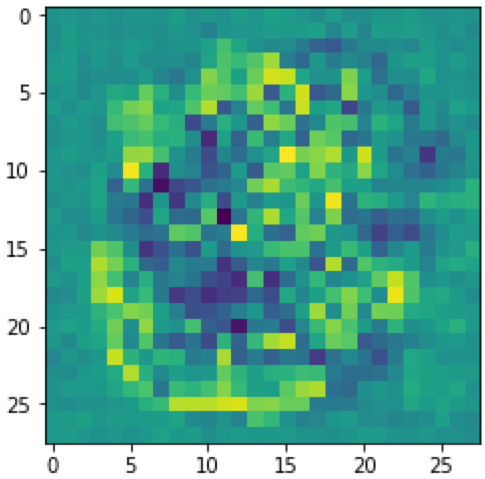
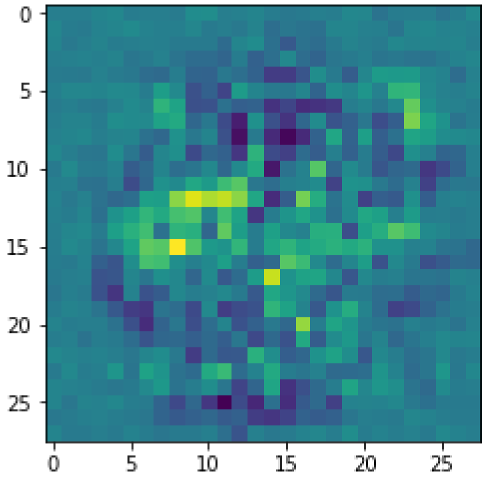
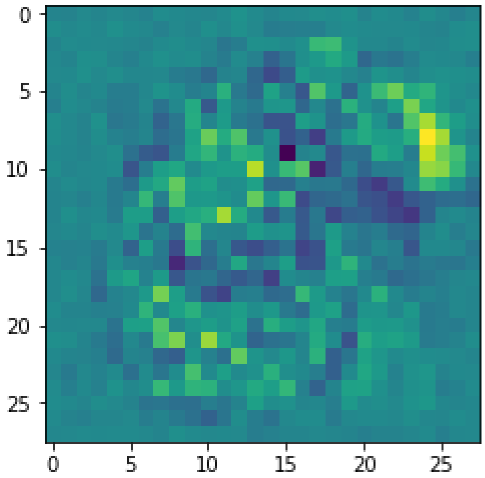
  We see that the although the train loss considerably decreases, the test values don't show improvement. This can be due to poorly optimised hyperparameters, or because the data is randomly broken into training and validation sets, there is possibility of variance of results on different runs.

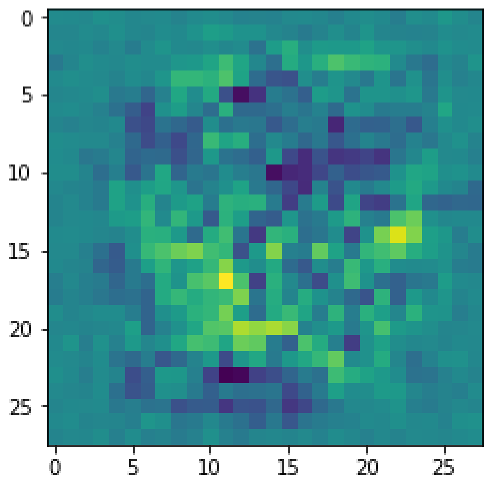
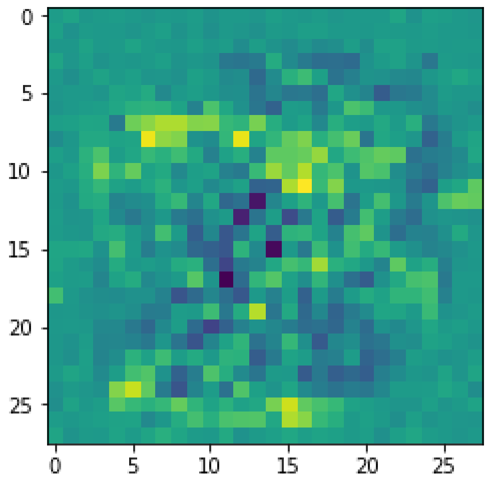
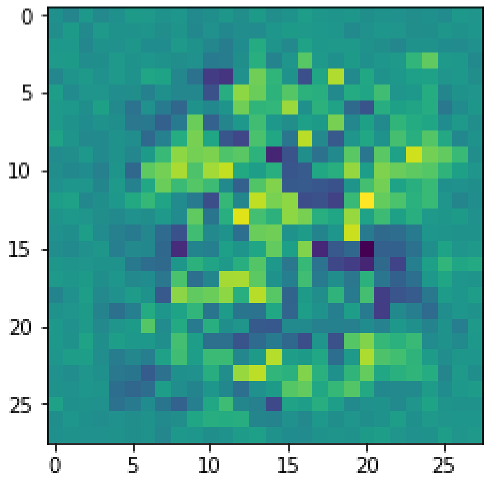
- **Comparison with inbuilt library**

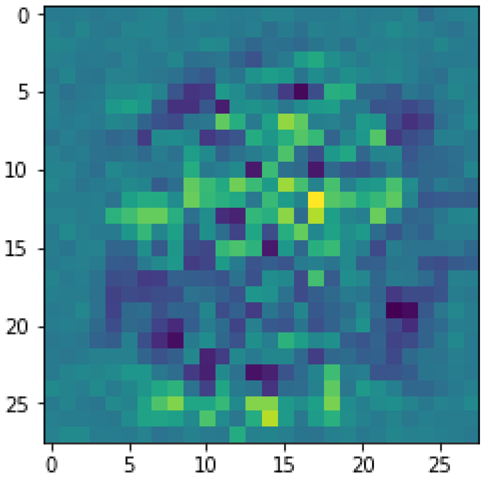
  I used scikit.neural_network.MLPclassifier library for this exercise. I used sigmoid activation function with one hidden layer, which had 15 nodes, because this was the best model found above. The library gave validation accuracy as **0.86333**, which is comparable to, although less than the value our built model gave. The reasons can be data shuffling or that my model gave me complete control over all steps like weight initialisation, which can play a vital role in determining accuracy while the inbuilt model had no such provisions and as a result the difference in accuracies.

- **Visualising the representations learnt by hidden layers**

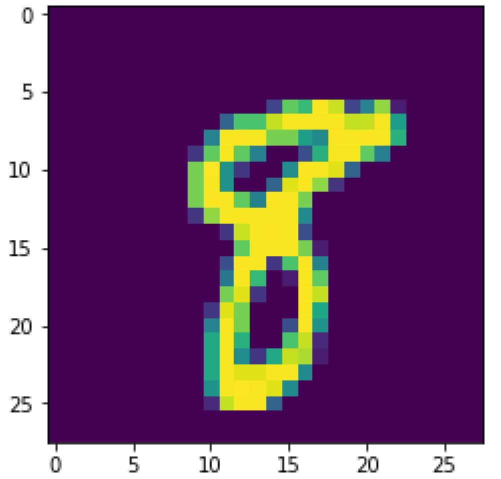
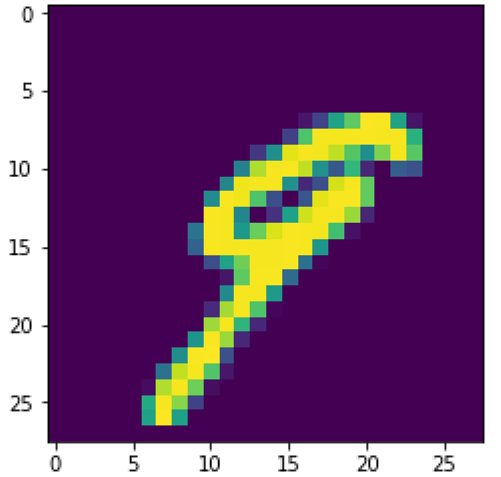
0	
1	
2	

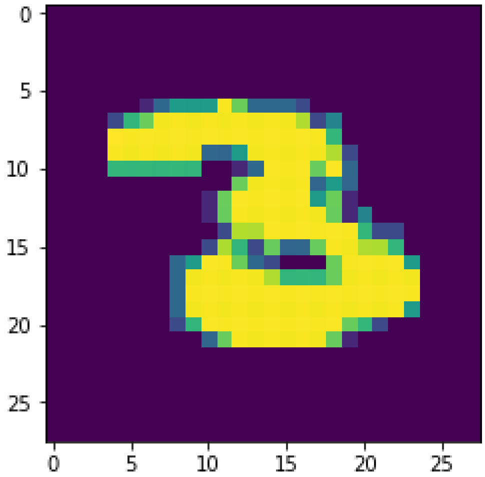
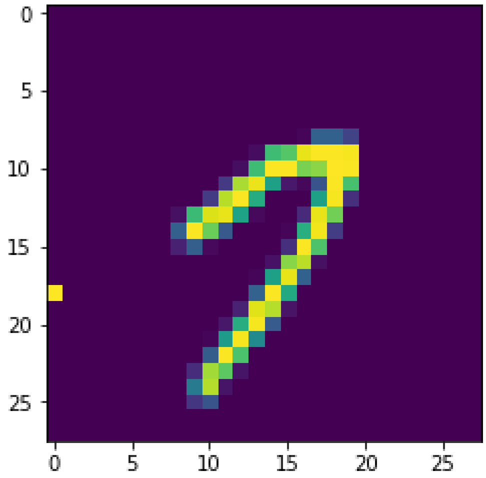
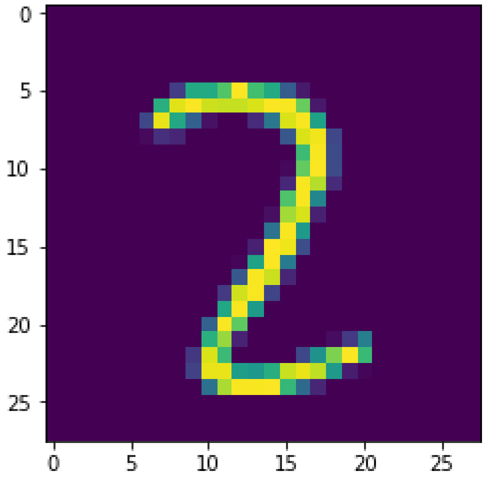
3	0 5 10 15 20 25 0 5 10 15 20 25
4	0 5 10 15 20 25 0 5 10 15 20 25
5	0 5 10 15 20 25 0 5 10 15 20 25

6	0 5 10 15 20 25 0 5 10 15 20 25
7	0 5 10 15 20 25 0 5 10 15 20 25
8	0 5 10 15 20 25 0 5 10 15 20 25

9	

- **Incorrectly classified examples**

Figure	What my model predicted
	5
	8

	7
	4
	3

- **Comparison with PCA features**

Now, I used the PCA features from the previous data-set and fed them to my neural network model, which performed best previously and analysed the results.

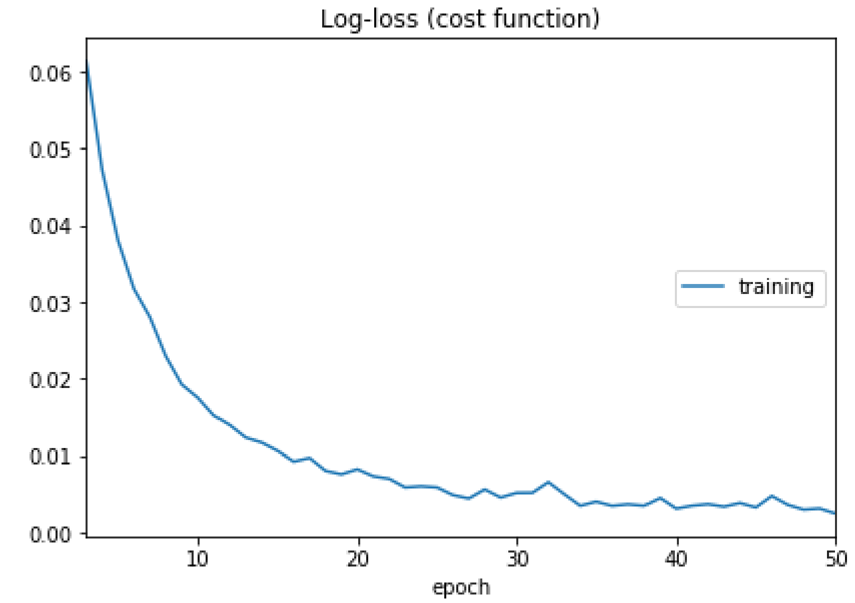
Validation accuracy – **0.8783333**
Loss – **0.4286755**

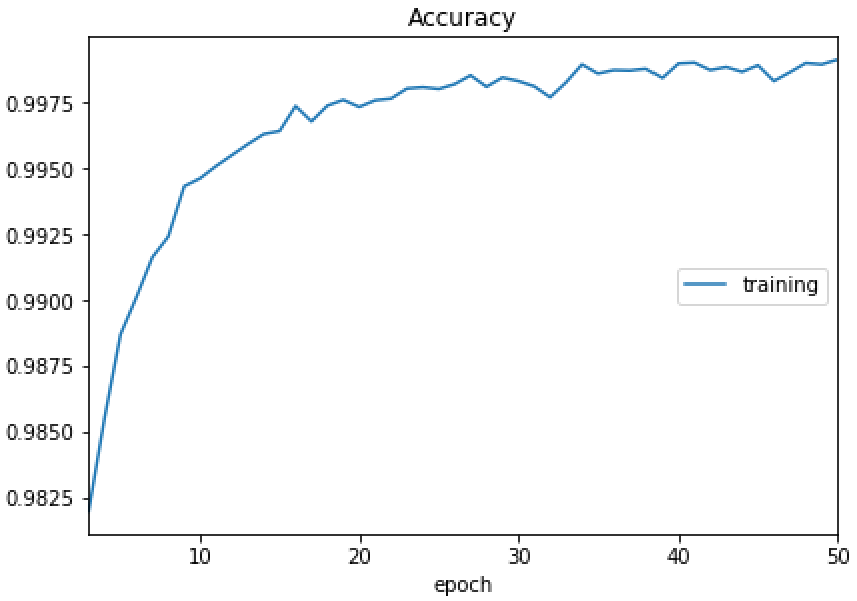
As a reminder, the previous results obtained were a validation accuracy of 0.91833 and a loss of 0.3754. This is pretty impressive considering that we have come down from 784 features to 25 features. The time taken went down to an astonishing **22 sec**, which is one-fifth of the time before.

- **Advanced neural networks**

This is the bonus part of the assignment. Keras was used as frontend with theano as backend. In this, the MNIST data set with 60000 train samples and 10000 test samples was used. Firstly, data was pre-processed and labels were one-hot encoded.
On first try, one layer was used with 300 nodes with dropout regularization. A single convolutional layer with filter number = 32, kernel size = 3 and relu activation function was used. A 2D-pooling layer with (2,2) size was used. 10 epochs and batch-size as 500 was set. The log-loss at the end of 10 epochs was **0.035** and train-accuracy was **0.989**. The test log-loss came out to be **0.0337** and test-accuracy was **0.9887**.
On second run, all parameters were same as above, only the number of epochs were changed to 20. The log-loss at the end of 20 epochs was **0.0318** and train-accuracy was **0.994**. The test log-loss came out to be **0.0337** and test-accuracy was **0.9902**.
Next, I added a second layer with 150 nodes with dropout regularization. The log-loss at the end of 10 epochs was **0.047** and train-accuracy was **0.985**. The test log-loss came out to be **0.0337** and test-accuracy was **0.9888**.
As we can see, the accuracy decreased. Hence, we go back to our previous configuration and increase the number of hidden units to 800. The log-loss at the end of 10 epochs was **0.022** and train-accuracy was **0.993**. The test log-loss came out to be **0.0372** and test-accuracy was **0.9888**.
As a final test, the above configuration was run to 50 epochs and the filter number was changed to 64. The log-loss at the end of 50 epochs

was **0.002** and train-accuracy was **0.999**. The test log-loss came out to be **0.042** and test-accuracy was **0.9905**. The graphs for the final run are shown below.


Log-loss (cost function)
training
epoch


Accuracy
training
epoch