Assignment 1

Deep Learning and Applications

The Report

Submitted By

Dipanshu Verma	B18054
Ayushman Dixit	B18164
Harshit Kumar Mittal	B18056

Abstract

The assignment deals with the application of the concepts of the neural networks learnt and build a simple neural network from scratch that could carry out the tasks of classification and regression. The report has been divided into 3 parts dealing with Regression, Classification and finally the Bag of Words implementation.

Regression

As we know regression, is supposed to learn the relation between the input data (d dimensional) and a continuously varying output (K dimensional)

Henceforth, the task at hand can be further subclassified to deal with a single perceptron regression (that will simply learn a hyperplane between the input and the output) and secondly, forming a generalized neural network that can form an arbitrary number of layers with any number of nodes and subsequently even learning the non-linearities in the data distribution.

Single-Perceptron Learning

For the purpose of this report the topic will be considered from 2 aspects of data provided to us:

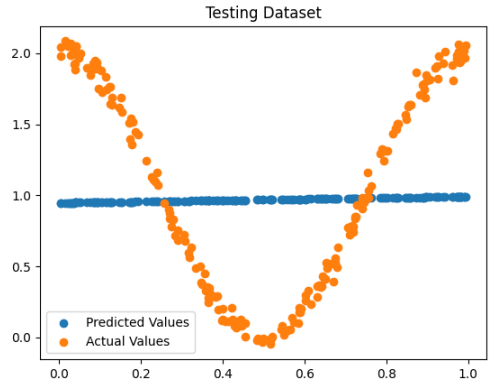
- Univariate data (1 dim input - 1 dim output)
- Bivariate data (2 dim input - 1 dim output)

Univariate Data

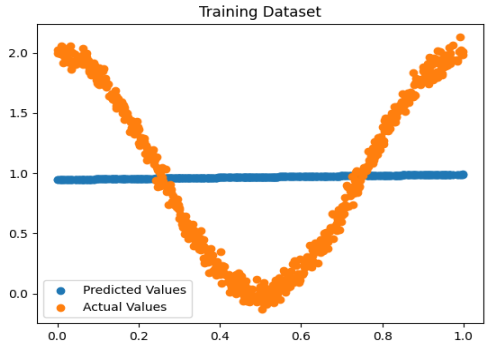
The data consists of real values for both the input and the output layers. The library of pandas is used for basic pre-processing of the data (randomisation splitting normalising etc.) and getting the data into the required format.

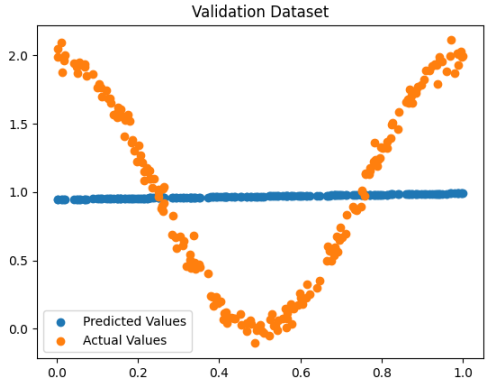
The single perceptron works on the basis of MSE (Mean Squared Error) loss, and learns accordingly by the method of gradient descent. The implementation of these algorithms can be viewed in the code itself.

It is worthwhile here to discuss the results using this very basic form of learning can attain



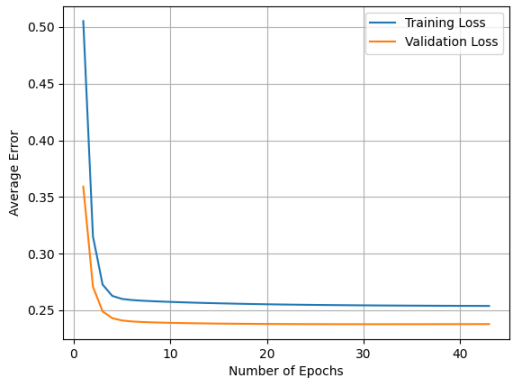
In the above picture we can see the orange distribution as the original dataset (test dataset) and the blue part depicting the learned curve. (In case of 1d input hence this is a line) The curve clearly as one can intuitively guess is the line that would give us the least MSE on the data for this distribution. The graphs obtained for the training and validation datasets can be seen as:





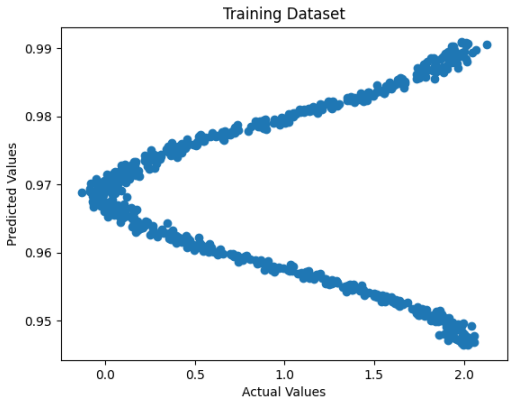
Clearly data learned is quite apt.

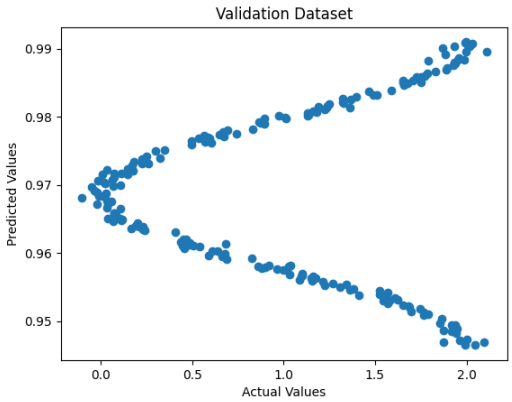
During its learning phase the loss vs epoch graph can be visualized with epochs on x axis and Average loss on y axis



As expected the model learns quite enough by the end of 10 iterations through the dataset. Also there are no signs of overfitting or underfitting as can be seen from comparing the training and validation losses.

To complete the discussion we could visualise the relation between the output and the predicted datasets. Which should be like a line (x=y) in case of a very appreciable amount of regression.





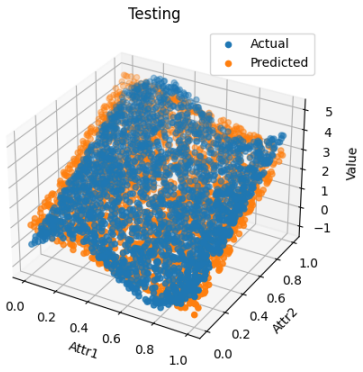
Clearly the linear regression is not still enrapturing all the aspects of the curve and the curve is thus a very little correlation between the two.

Bivariate Data

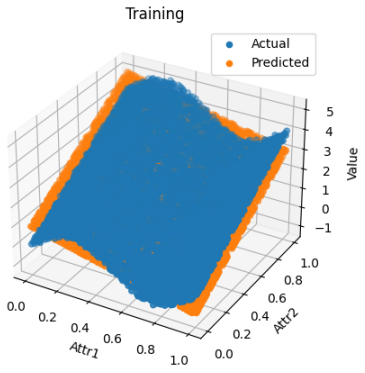
The data consists of real values for both the input and the output layers. The library of pandas is used for basic pre-processing of the data (randomisation splitting normalising etc.) and getting the data into the required format.

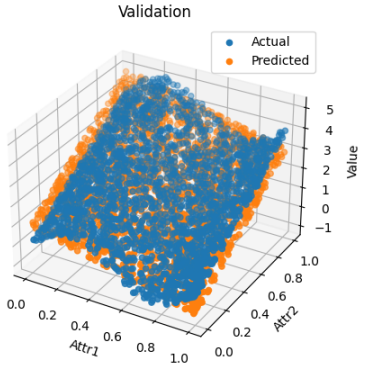
The single perceptron works on the basis of MSE (Mean Squared Error) loss, and learns accordingly by the method of gradient descent. The implementation of these algorithms can be viewed in the code itself.

It is worthwhile here to discuss the results using this very basic form of learning can attain



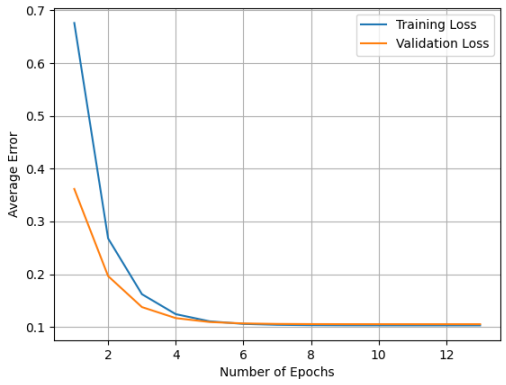
Since the data is 2D thus we obtain a 3D graph for the distributions. In the above picture we can see the blue distribution as the original dataset (test dataset) and the blue part depicting the learned curve. (In case of 2d input hence this is a Plane) The curve clearly as one can intuitively guess is the plane that would give us the least MSE on the actual data for this distribution. The graphs obtained for the training and validation datasets can be seen as:





Clearly data learned is quite apt.

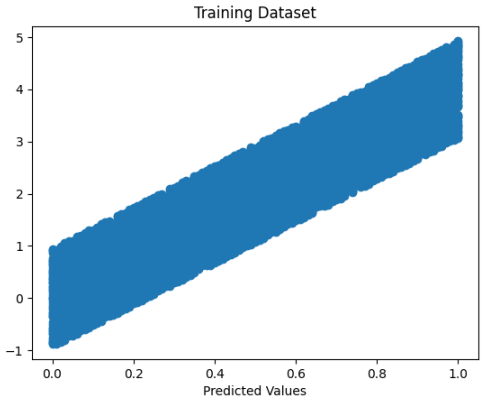
During its learning phase the loss vs epoch graph can be visualized with epochs on x axis and Average loss on y axis

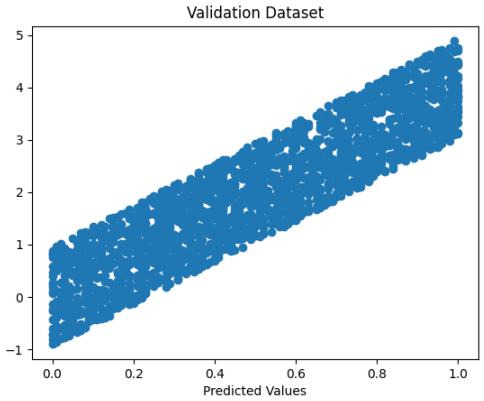


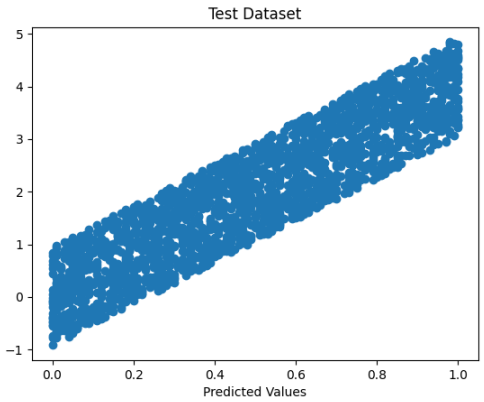
As expected the model learns quite enough by the end of 6 iterations through the

dataset. Also there are no signs of overfitting or underfitting as can be seen from comparing the training and validation losses.

To complete the discussion we could visualise the relation between the output and the predicted datasets. Which should be like a line (x=y) in case of a very appreciable amount of regression.







Clearly the linear regression is not still enrapturing all the aspects of the curve However the correlation between the distributions in this case is quite high as can be seen from the graphs.

MLFFNN based regression

For the purpose of this report the topic will be considered from 2 aspects of data provided to us:

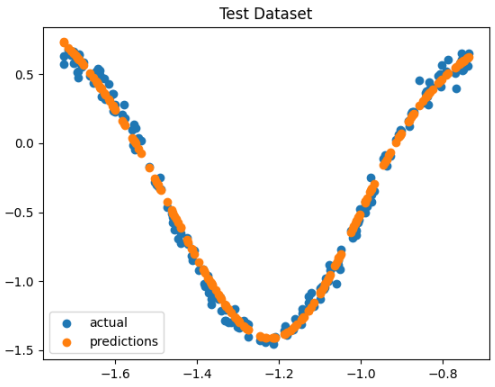
- Univariate data (1 dim input - 1 dim output)
- Bivariate data (2 dim input - 1 dim output)

Univariate Data

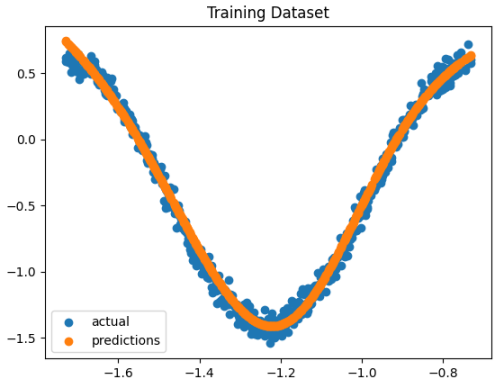
The data consists of real values for both the input and the output layers. The library of pandas is used for basic pre-processing of the data (randomisation splitting normalising etc.) and getting the data into the required format.

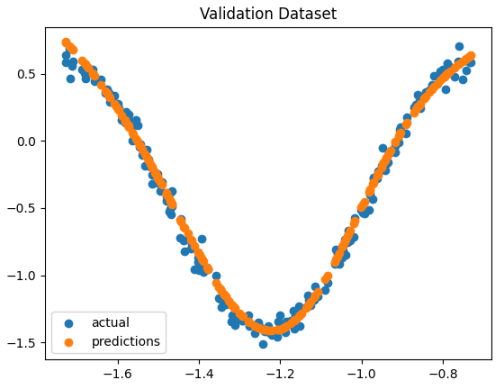
The MLFFNN (multi layer feed forward neural network) works on the basis of MSE (Mean Squared Error) loss, and learns accordingly by the method of gradient descent. The model consists of 1 hidden layer with 3 nodes and an output layer with 1 node.

It is worthwhile here to discuss the results using this advanced form of learning (expected to learn the non-linearities skipped by the single perceptron regressions)



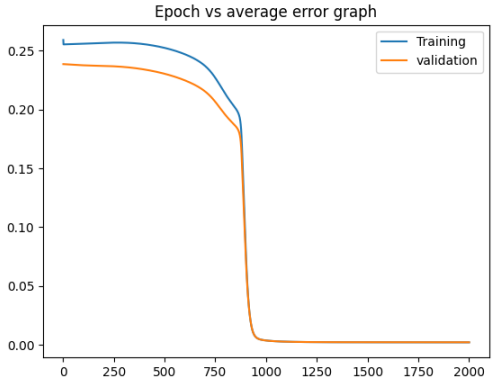
In the above picture we can see the blue distribution as the original dataset (test dataset) and the orange part depicting the learned curve. The curve learned is almost the same distribution as the training data and gives great results on the test data. The graphs obtained for the training and validation datasets can be seen as:





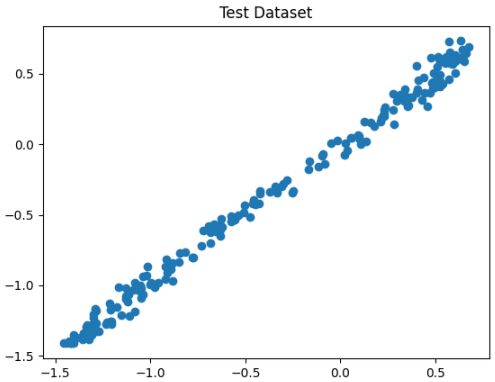
Clearly data learned is quite apt.

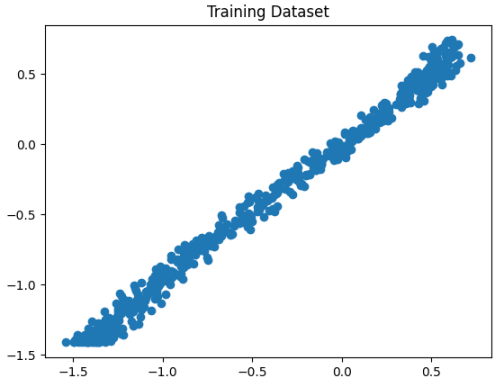
During its learning phase the loss vs epoch graph can be visualized with epochs on x axis and Average loss on y axis

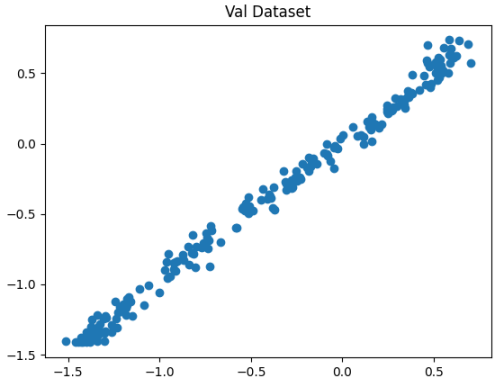


the model learns quite enough by the end of 1000 iterations through the dataset. Also there are no signs of overfitting or underfitting as can be seen from comparing the training and validation losses.

To complete the discussion we could visualise the relation between the output and the predicted datasets. Which should be like a line (x=y) in case of a very appreciable amount of regression.



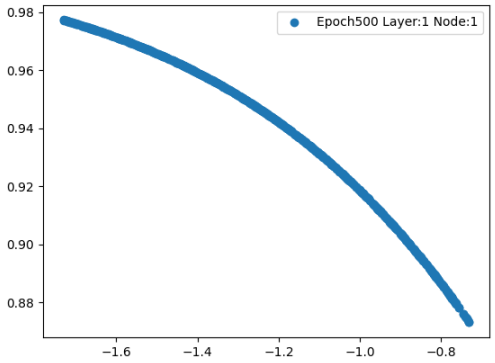


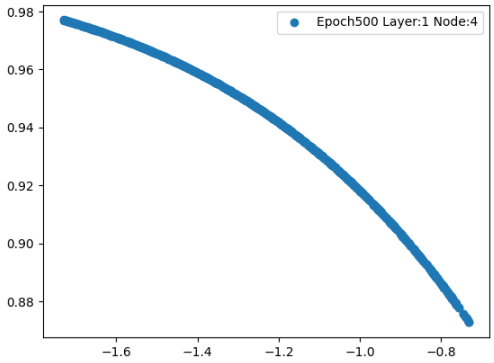


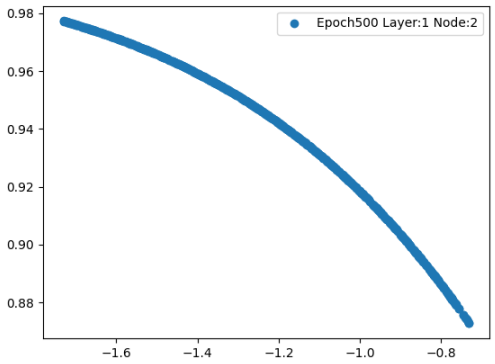
The data is having a correlation coefficient between the axis highly close to one. This depicts the outputs predicted are very close to the actual ground truth values of the dataset for all train, test and validation.

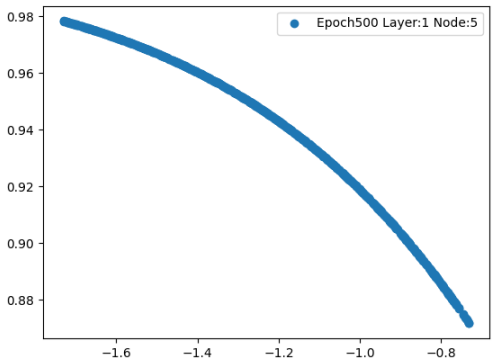
Another important aspect to look into in this case is what the neural systems have learnt at each of their nodes. In this case let us see the state of the learnt individual values at each node (at epoch 500)

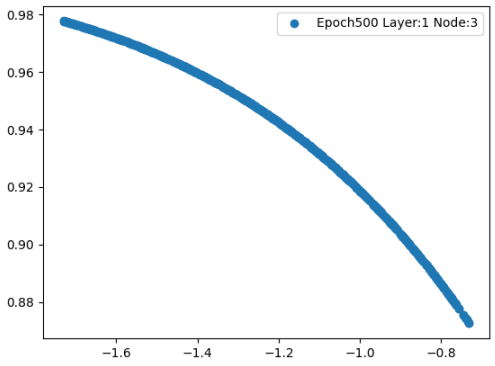
Epoch:500

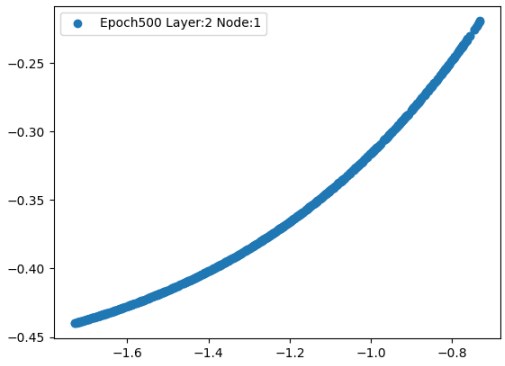








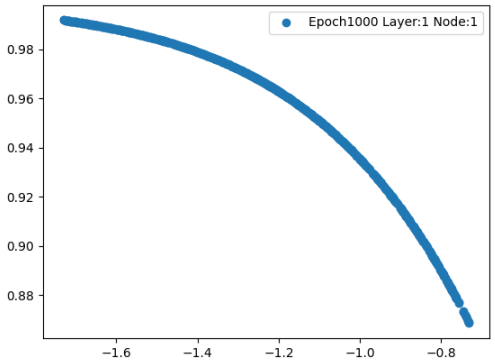


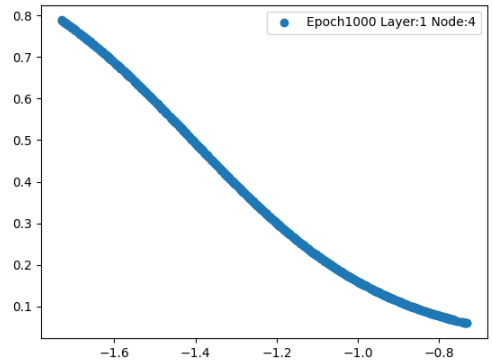


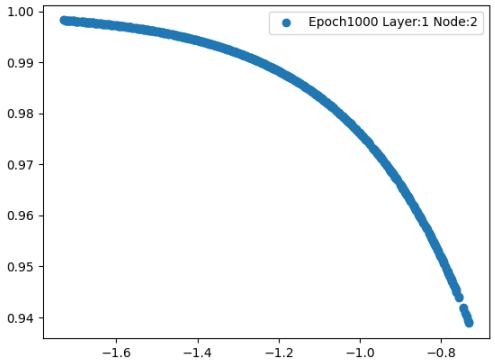
The nodes have started to begin to learn the non-linearities. Also the weighted sum gives a curve totally in the opposite direction to what hidden layer nodes learn. This suggests how important the relative size of weights can be.

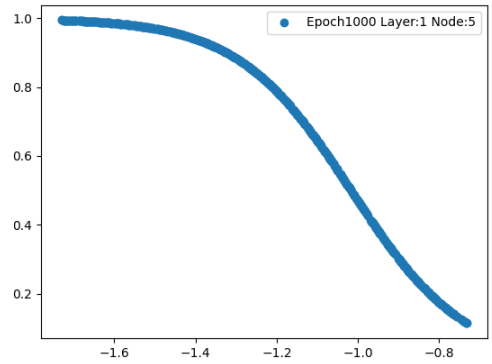
By the end of epoch 500, the model has started to learn various orientations of but of the same curve these when combined give the curve obtained for layer 2 node 1 (output layer).

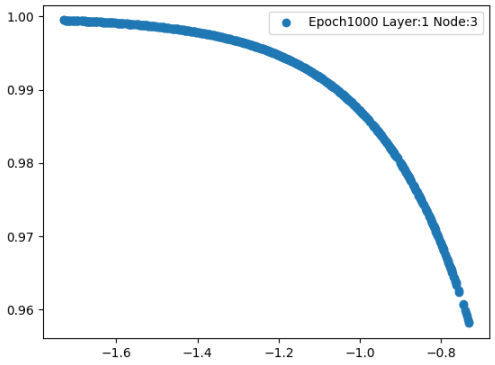
Epoch: 1000

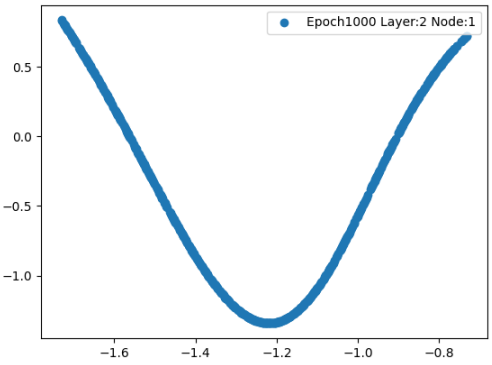












It is thus by the end of epoch 2 that we see the final output node delivering a quite good enough output (inline with that of actual distribution).

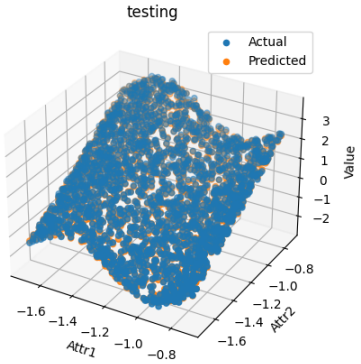
Also by the end of epoch 1000 we see the hidden layer nodes learn various features such as curves of different shapes that make up the final curve.

Bivariate Data

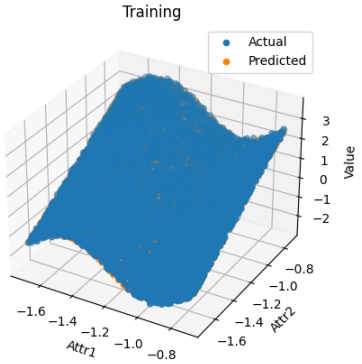
The data consists of 2D input real values for both the input and the output layers. The library of pandas is used for basic pre-processing of the data (randomisation splitting normalising etc.) and getting the data into the required format.

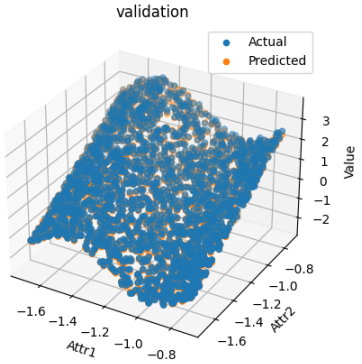
The single perceptron works on the basis of MSE (Mean Squared Error) loss, and learns accordingly by the method of gradient descent. The model consists of 1 hidden layer with 3 nodes and an output layer with 1 node.

It is worthwhile here to discuss the results using this advanced form of learning (expected to learn the non-linearities skipped by the single perceptron regressions)



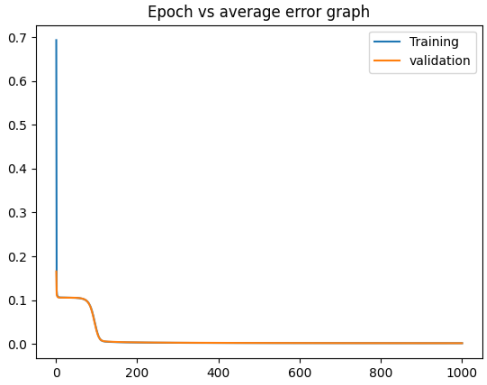
Since the data is 2D thus we obtain a 3D graph for the distributions. In the above picture we can see the blue distribution as the original dataset (test dataset) and the orange part depicting the learned curve. (In case of 2d input hence this is a Plane) The obtained curve is almost overlapping with the actual curve depicting the model has learned the relation quite well. The graphs obtained for the training and validation datasets can be seen as:





Clearly data learned is highly apt.

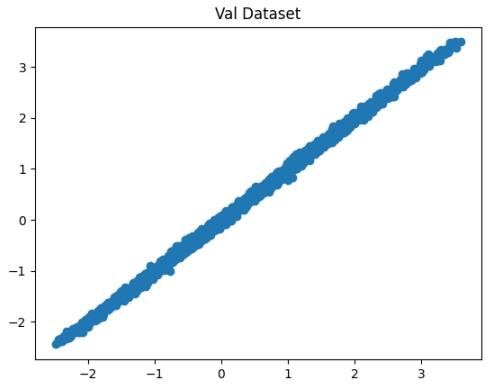
During its learning phase the loss vs epoch graph can be visualized with epochs on x axis and Average loss on y axis



As expected the model learns quite enough by the end of 2000 iterations through the dataset. Also there are no signs of overfitting or underfitting as can be seen

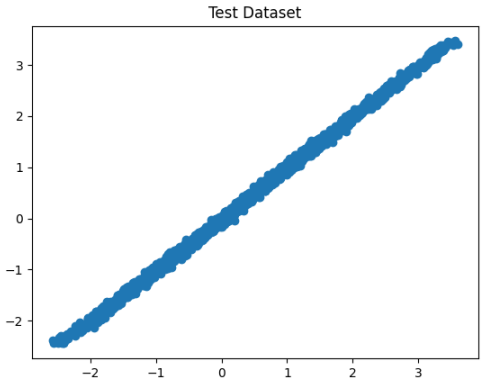
from comparing the training and validation losses. Also while learning there are instances of plateaus which had to be overcome for a better accuracy

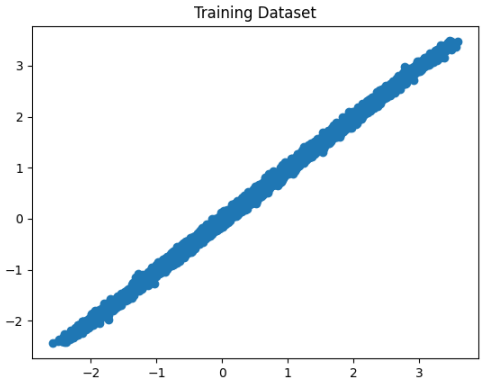
To complete the discussion we could visualise the relation between the output and the predicted datasets. Which should be like a line (x=y) in case of a very appreciable amount of regression.



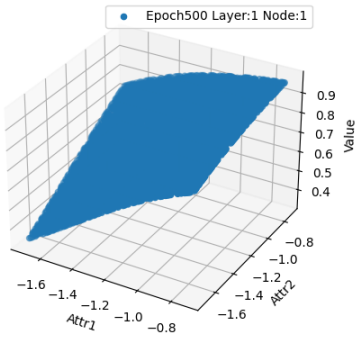
The data is having a correlation coefficient between the axis highly close to one. This depicts the outputs predicted are very close to the actual ground truth values of the dataset for all train, test and validation.

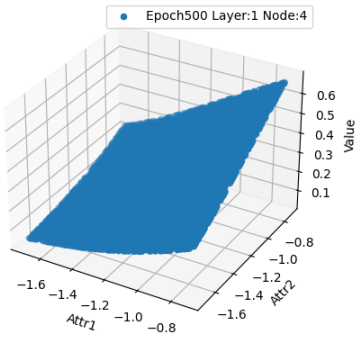
Another important aspect to look into in this case is what the neural systems have learnt at each of their nodes. In this case let us see the state of the learnt individual values at each node (at epoch 500)

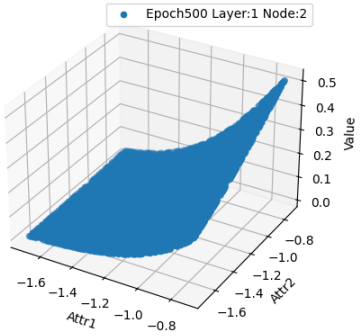


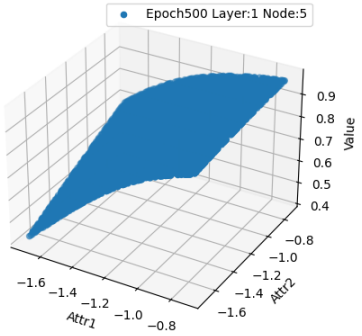


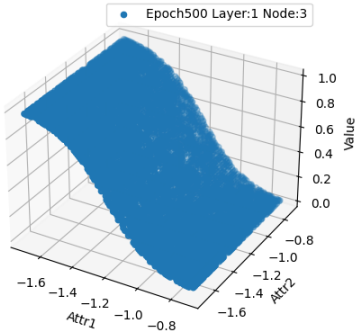
Epoch:500

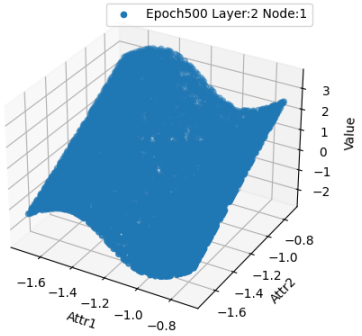












By the end of epoch 500 the model has already learned various curves that when combined gives a representation close enough to the actual representation.

This could also be verified by seeing the loss curve per epoch that reduces low enough at epoch=200. Therefore displaying the results of epoch 1000 in this case would be more or less redundant information.

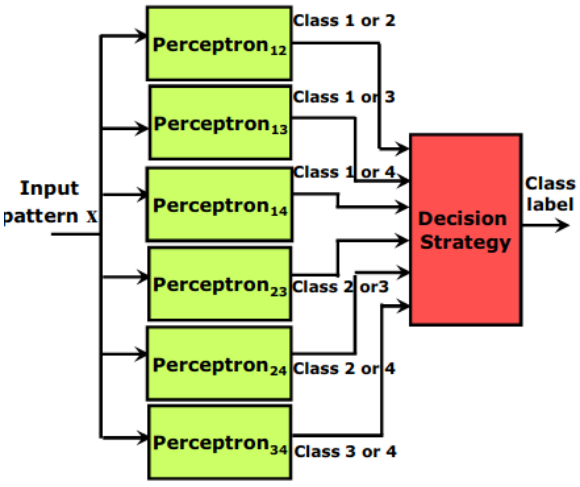
Classification

The report articulates the findings of doing the classification task using a single neuron or perceptron with sigmoidal activation function and through a multi-layered fully connected neural network. For perceptron, the dataset provided is linearly separable and non-linearly separable 2-dimensional data while classification using neural network is done on above mentioned and 3-class image data.

Classification through perceptron model

The model:

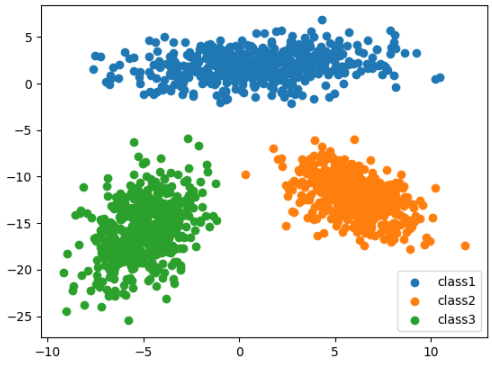
A single perceptron model can only do binary classification. However here we want to do multi-classification. Here we have adopted one-vs-one model strategy. This works by training models to classify every pair of classes and then doing a voting scheme to classify a sample. A schematic model is shown below(*Ref-Class Slides*).



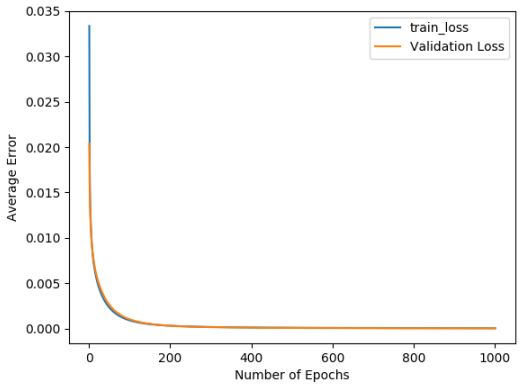
As we know perceptrons have the representational power to classify linearly separable training dataset with 100% accuracy. We have used a sigmoidal activation function on top of the weighted sum.

Dataset 1(Linearly separable data)

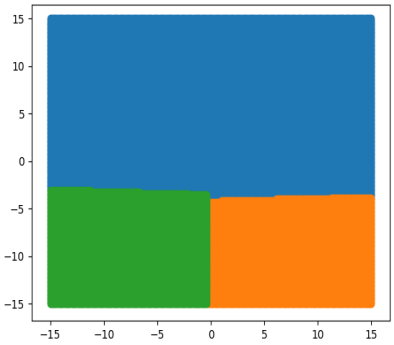
- A bivariate 3 classed linearly separable dataset. It is visualized below:



- As we can see from the below epoch vs average error plot our model is learning and generalizing quite well which is evident from the fall of average error across epochs and both training and validation sets.



- Decision boundary learned through Linearly separable data. Blue region represents class 1, orange region class2 , green region class3.

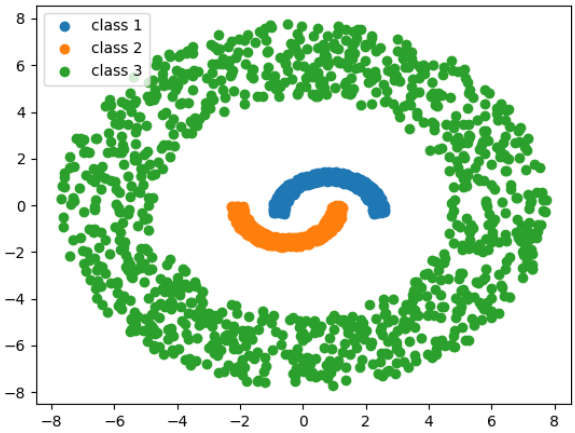


- The accuracy recorded on both train and test was 100%, as the data is linearly separable and perceptron has learned the decision boundary quite well. The confusion matrix on the test.

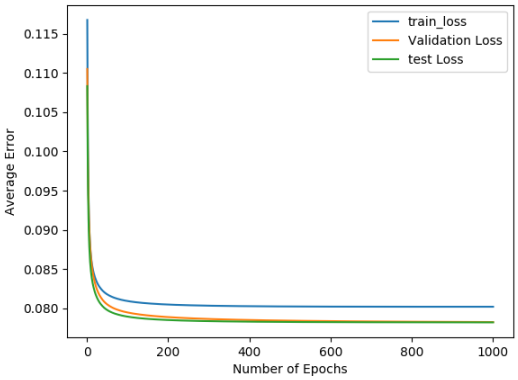
  [[200, 0 , 0],
  [0 , 200, 0],
  [0 , 0 , 200]]

Dataset 2(non-linearly separable data)

- A bivariate 3 classed non-linearly separable dataset. It is visualized below:

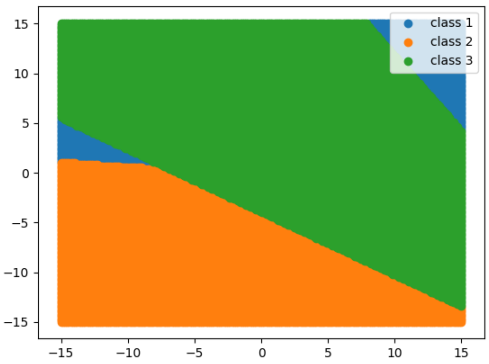


- As we can see from the below epoch vs average error plot our model is learning and generalizing quite well which is evident from the fall of average error across epochs and both training and validation sets however the error seems to saturate and no significant reduction is observed after 400 epochs. This may be because the data is very complex and the perceptron representation power is not sufficient.

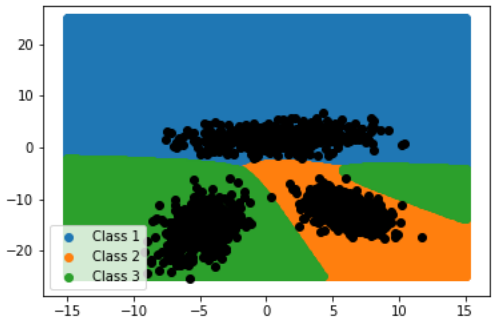


The accuracy of the model on test data is 35.1% which is not good.Hence we need a better classifier. The confusion matrix is:
[[0, 0, 216],
[0, 0, 189],
[1, 113, 281]])

- The decision surface learned by model



decision boundaries are plotted below.

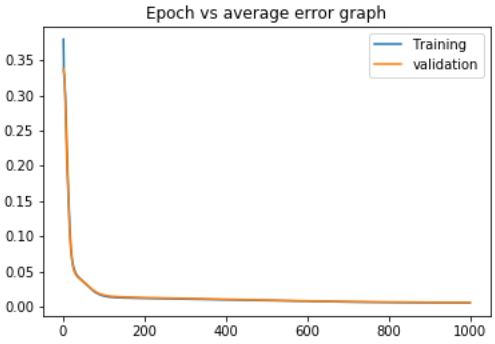


Classification through Multi-layered fully connected neural network model

Here we have used an artificial neural network or multilayer perceptron to make a classification model. Our model contains 1 hidden layer with 3 nodes and a 3 dimensional output layer.

Dataset 1 - Linearly separable

- Following curve was observed of epoch vs average error



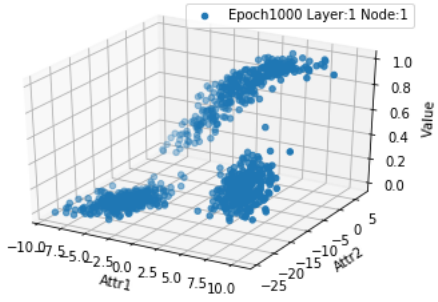
The learning seems to be very smooth as evidenced by the graph.

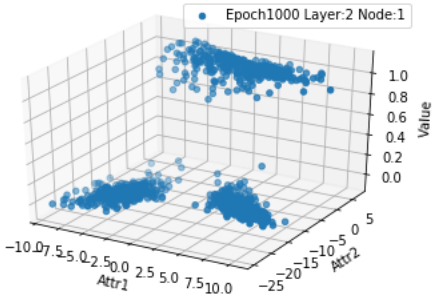
- The accuracy was found to be 100% for both test and train dataset. The

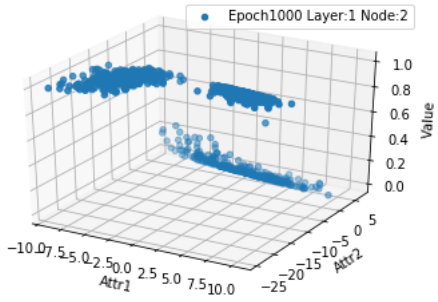
Confusion matrix was calculated

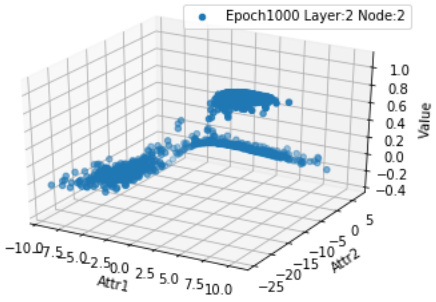
[[200, 0 , 0],
[0 , 200, 0],
[0 , 0 , 200]]

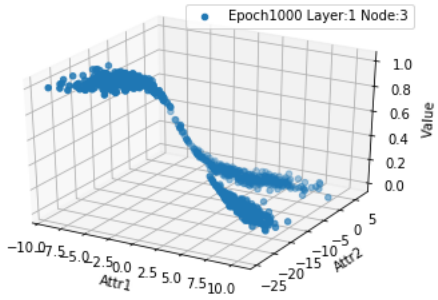
Epoch:1000

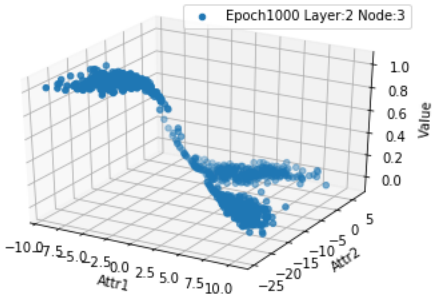










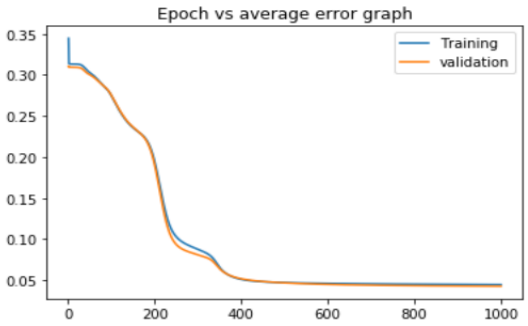


The Nodes in the first layer as we can see are learning various abstract S shaped features from the input data. These features further when combined as a weighted sum would help the output layer to classify better.

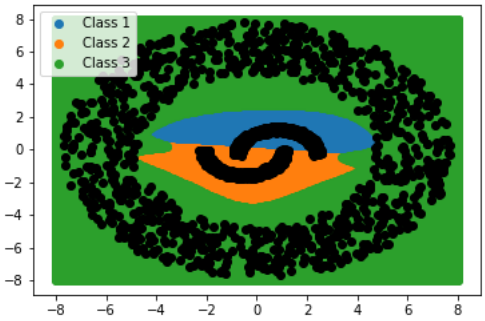
The nodes of the output layers as we can see are each learning the aspects to classify one class vs all the others. Clearly hence one the classes seems to be lifted (near to 1) while the other classes are placed below (near to 0)

Dataset 2 - Non linearly separable

- Following curve was observed of epoch vs average error



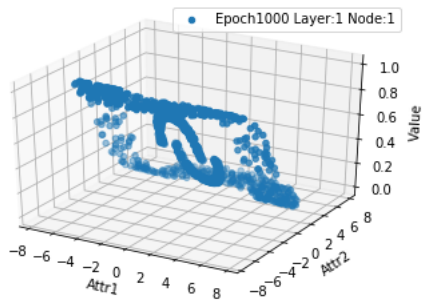
- The accuracy was found to be 97% on test data which is far better than the earlier model of perceptron. The decision boundaries are plotted below.

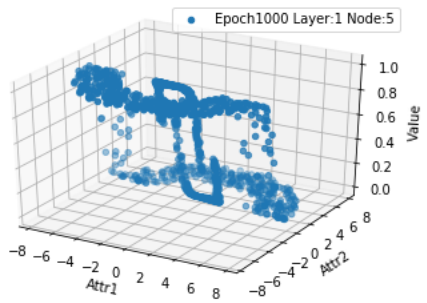


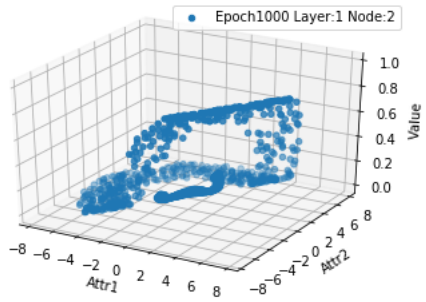
The confusion matrix on test data is

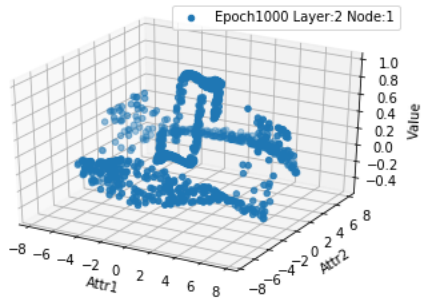
[[95, 6, 1,
[4, 86 , 1]
[0, 0, 207]]

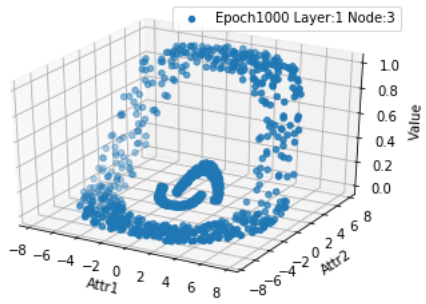
Epoch:1000

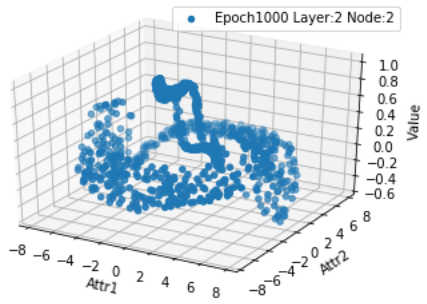

Epoch1000 Layer:1 Node:1
Attr1
Attr2
Value

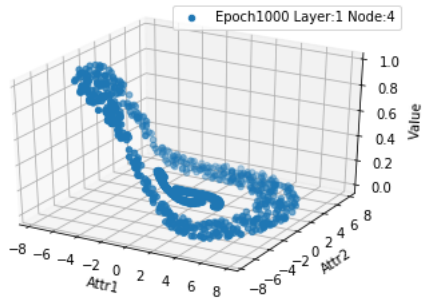

Epoch1000 Layer:1 Node:5
Attr1
Attr2
Value

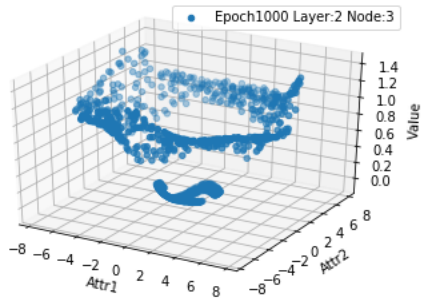

Epoch1000 Layer:1 Node:2
Attr1
Attr2
Value


Epoch1000 Layer:2 Node:1
Attr1
Attr2
Value


Epoch1000 Layer:1 Node:3
Attr1
Attr2
Value


Epoch1000 Layer:2 Node:2
Attr1
Attr2
Value


Epoch1000 Layer:1 Node:4
Attr1
Attr2
Value


Epoch1000 Layer:2 Node:3
Attr1
Attr2
Value

Dataset 3 - 3 classed image dataset

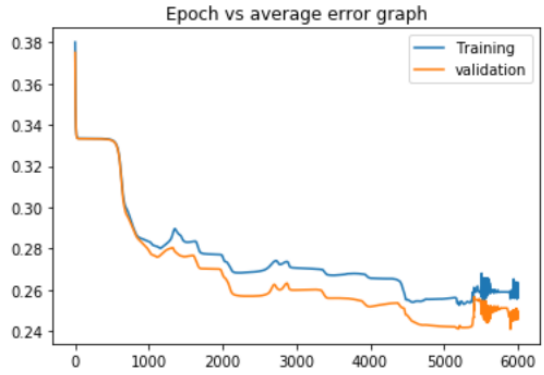
First we tried to extract features of an image using a bag of visual words. For feature extraction we split the image into chunks of small size(32X32) and used a 24 dimensional vector to represent that small part of the image. Now using hard clustering we find cluster labels for each 24 dimensional vector and an image is 32 dimensional representation derived from cluster labels of all small chunks.

After feature extraction, we have a 32 dimensional vector representation of an image that works as input for learning models.

The model:

After trying with various hyperparameters such as activation functions, number of layers, etc. Best results were reported on a model with two 16 and 8 nodes hidden layers , 32 dimensional input layer and 3 dimensional output layer.

The epoch vs average error curve:



Confusion matrix on test

[[1, 45, 4], [0, 46, 4], [1, 11, 38]]