CS671: Deep Learning and Applications
Programming Assignment I

Adarsh Raj
B18100

Shivani Bodhke
B18108

Sanskar Gupta
B18140

Abstract

In this report we have implemented the single perceptron and multilayer perceptron model for classification and regression tasks.We analysed the results for different learning rates , different epoches and differnt number of hidden layers(MLFFNN) .

1 About The Data

We are given various datasets for training and testing our models for the classification and regression task. For all the datasets, we divided 60% of the data from each class as training data, 20% of the data from each class as validation data and remaining 20% of data from each class as test data.

1.1 Classification Task Data

We are given three datasets for classification task.

- Linearly separable data : This is shown in Fig1.
- Non Linearly Separable Data : This is shown in Fig2.
- Image Data

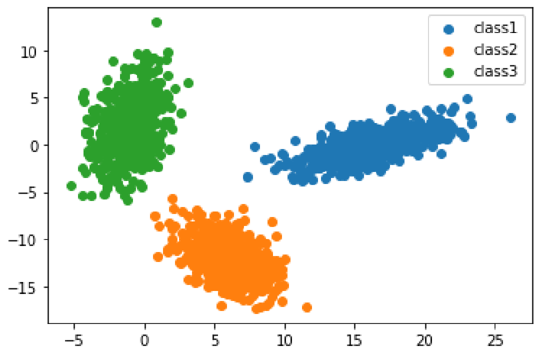


Figure 1: Linearly separable data

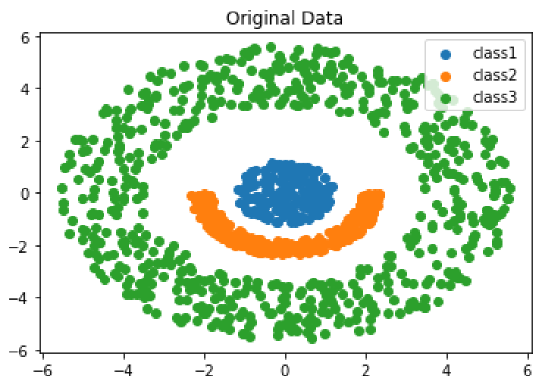


Figure 2: Non linearly separable data

1.2 Regression Task Data

We are given two datasets for regression task.

- 1-dimensional (Univariate) input data : This is shown in Fig3.
- 2-dimensional (Bivariate) input data : This is shown in Fig4.

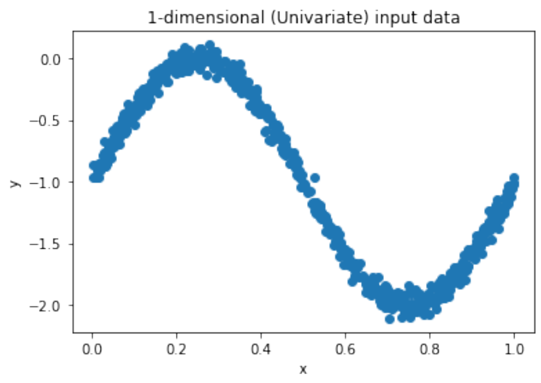


Figure 3: 1-dimensional Univariate data

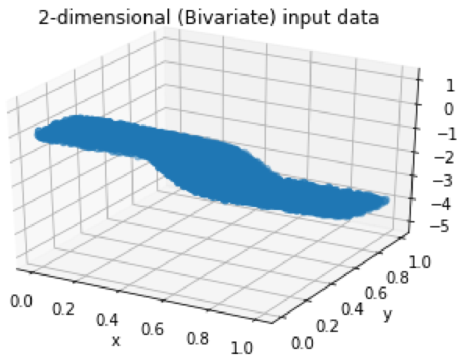


Figure 4: 2-dimensional Bivariate data

2 Introduction

2.1 Perceptron

Perceptron is a single layer neural network. There are four main parts of Perceptron i.e. input value,weights and bias, activation function and net sum.In our experiment we have used sigmoid activation function for classification task and linear activation function for regression task. Learning of bias and weights are done by gradient decent and backpropagation method.

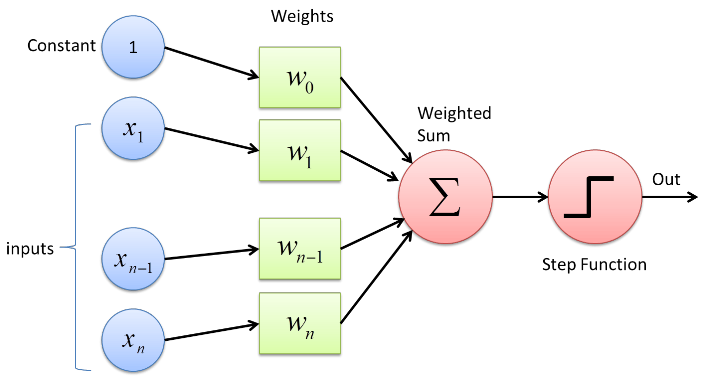


Figure 5: This is a plot of linearly separable data given to us

2.2 Gradient Decent

It is a first order iterative approach for finding local minimum of any differentiable function .It is the optimised way of reaching local minimum . The main idea in this approach is to move in the opposite direction of the gradient at any point because this way we can reach minima fastest possible way.

2.3 Backpropagation

You initialise the random weights and calculate the error in forward pass , so to minimise the error we will have to change weights and bias in backward pass . this is called backpropagation in which error is being backpropagted throughut your model and weights are being updated .

2.4 Multilayer Perceptron

We know single perceptron is binary classifier . so it can classify only linaerly separable boolean function accurately . So we add more layers of perceptron to make it classify accurately for non linear dataset or learn complicated architectures for better accuracy of prediction .Multilayer perceptron can be seen as multiple small linear boundries are self learnt for non linear classification .

3 Presentation of Results

3.1 Classification Task Results

3.1.1 Linearly separable data using perceptron with sigmoid activation function

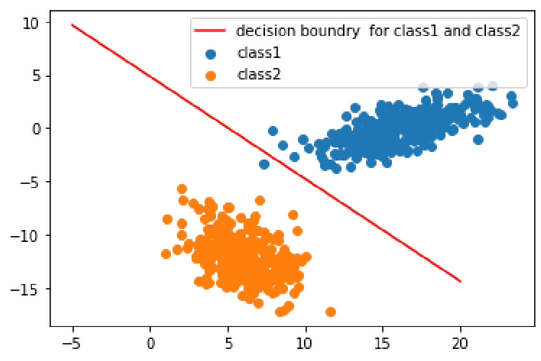


Figure 6: lr = 0.1 and epoches = 50

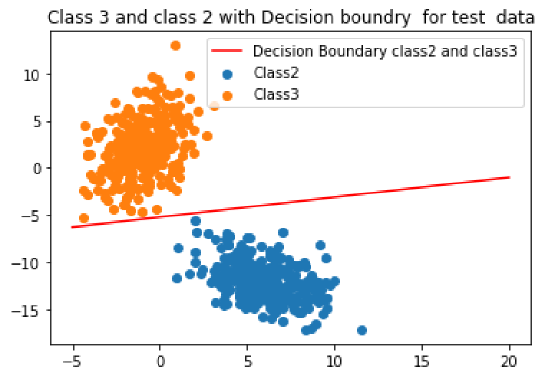


Figure 7: lr = 0.1 and epoches = 50

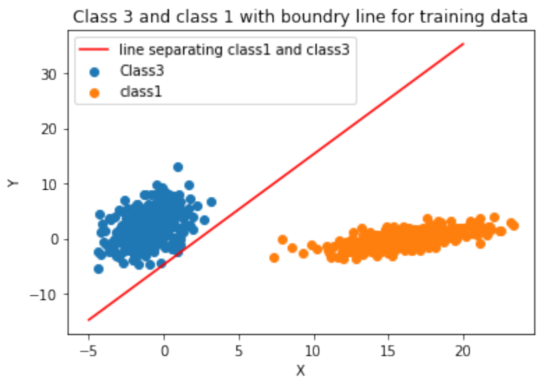


Figure 8: lr = 0.1 and epoches = 50

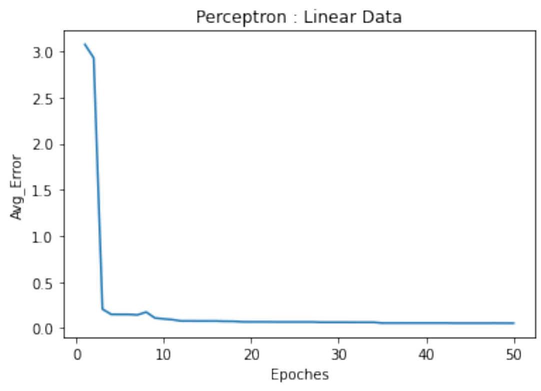


Figure 9: lr = 0.1 and epoches = 50

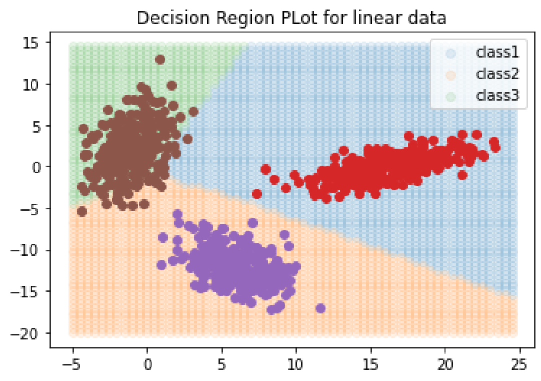


Figure 10: lr = 0.1 and epoches = 50

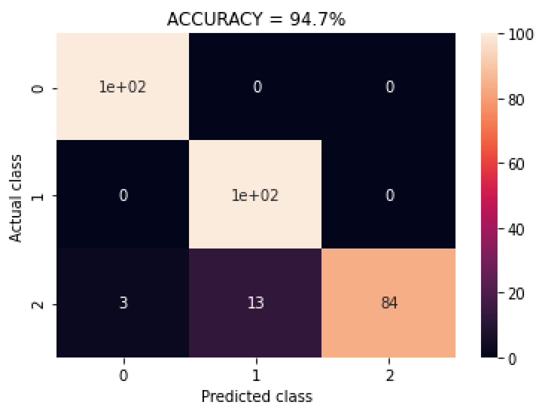


Figure 11: lr = 0.1 and epoches = 50

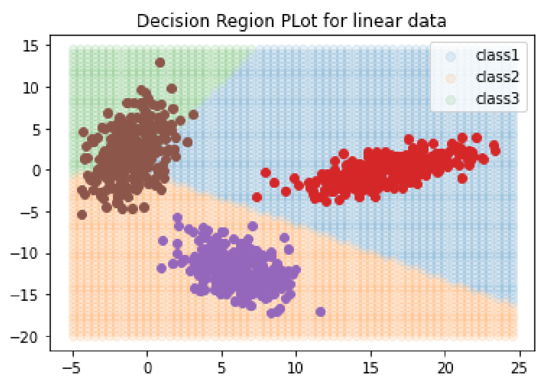


Figure 12: lr = 0.2 and epoches = 10

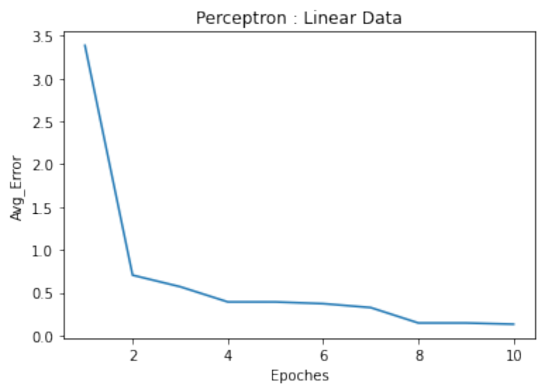


Figure 13: lr = 0.2 and epoches = 10

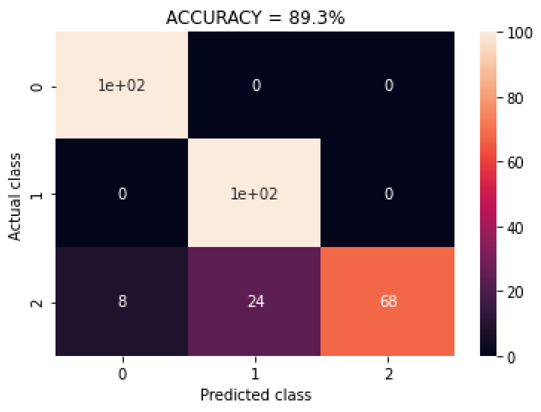


Figure 14: lr = 0.2 and epochs = 10

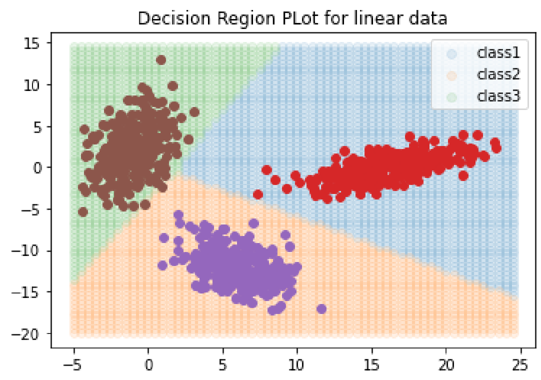


Figure 15: lr = 0.5 and epochs = 1000

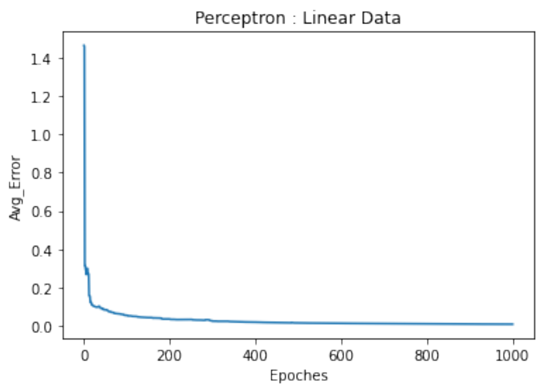


Figure 16: lr = 0.5 and epochs = 1000

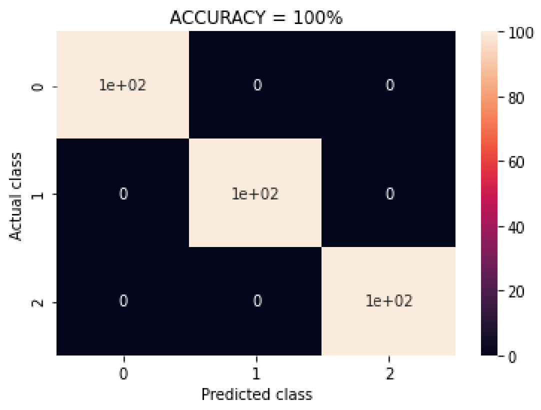


Figure 17: lr = 0.5 and epochs = 1000

Inferences from the plots:

- No matter how you start with weights and bias and learning rate, perceptron will always converge in case of linearly separable data this can be seen from all error plots .
- Perceptron easily classifies linearly separable data with good accuracy.
- If we increase the lr then , it will take more time/epoches to converge .
- In our case perceptron converges in not more than 10 epoches .

3.1.2 Non-Linearly separable data using perceptron with sigmoid activation function

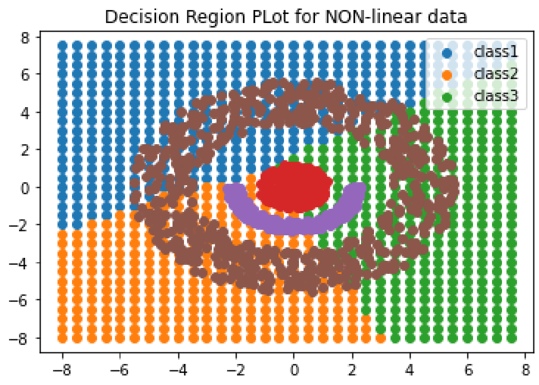


Figure 18: lr = 0.1 and epochs = 10

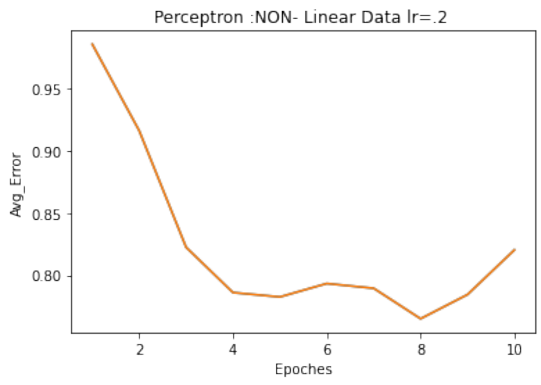
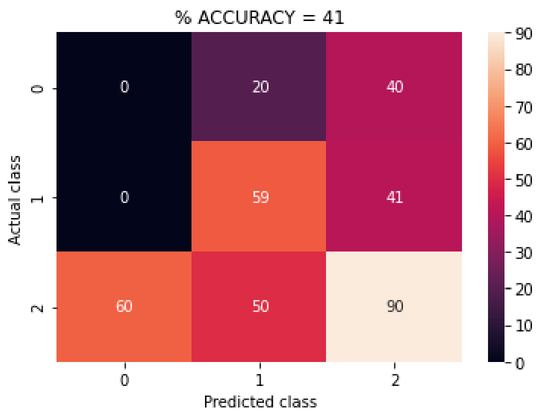


Figure 19: lr = 0.1 and epochs = 10



	0	1	2
0	0	20	40
1	0	59	41
2	60	50	90

Figure 20: lr = 0.1 and epochs = 10

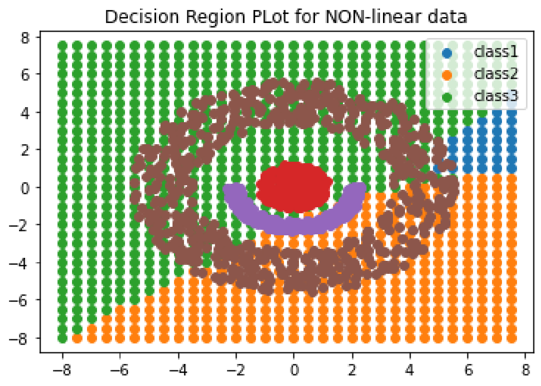


Figure 21: lr = .2 and epochs = 1000

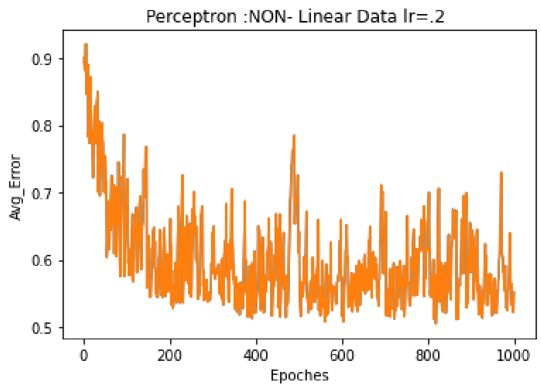
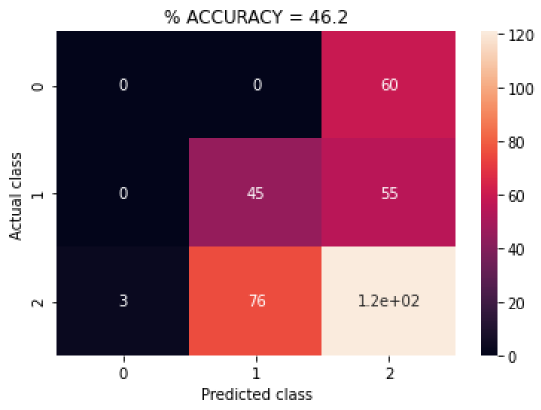


Figure 22: lr = .2 and epochs = 1000



	0	1	2
0	0	0	60
1	0	45	55
2	3	76	1.2e+02

Figure 23: lr = .2 and epochs = 1000

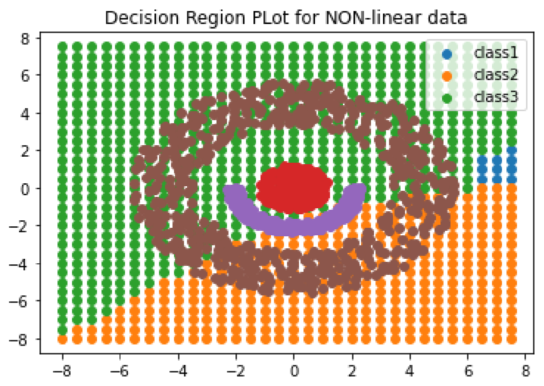


Figure 24: lr = .1 and epochs = 1000

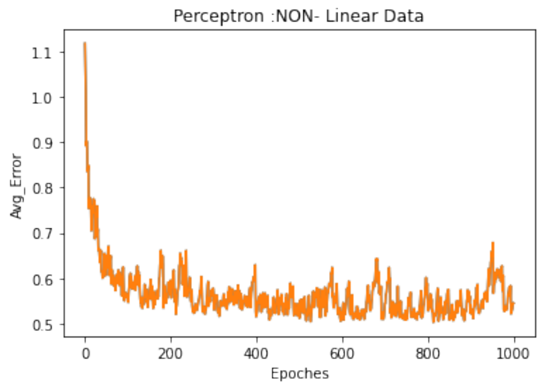


Figure 25: lr = .1 and epochs = 1000

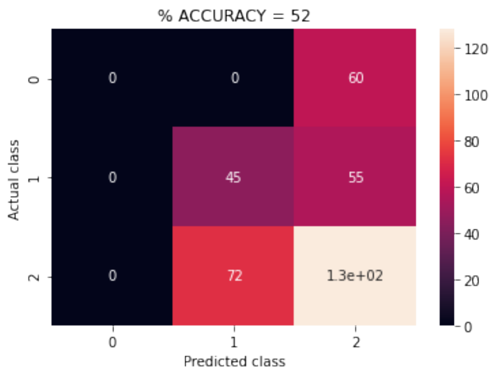


Figure 26: lr = .1 and epochs = 1000

Inferences from the above plots for non-linear data:

- No matter how you start with weights, bias and learning rate, perceptron will never converge in case of non-linearly separable data ,this can be seen from all error plots.
- If we decrease the lr then for same number of epochs , accuracy increases .
- In our case, perceptron will never converge and maximum accuracy found on test sample is 52% .

3.1.3 Linearly separable data using multi layer feed-forward neural network

For all the experiment results shown here learning rate = 0.1 and number of epochs = 50 . I have used a different convention for hidden layer representation [x,y] means 2 hidden layers are there with x and y number of neurons respectively .

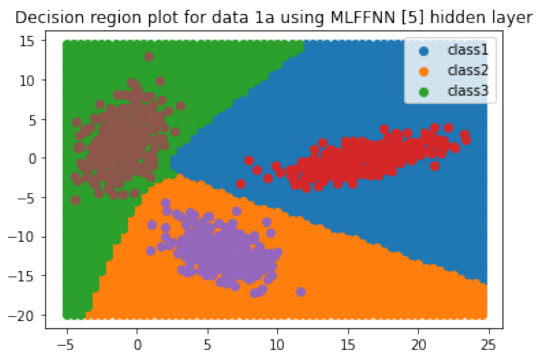


Figure 27: hidden-layers = [5]

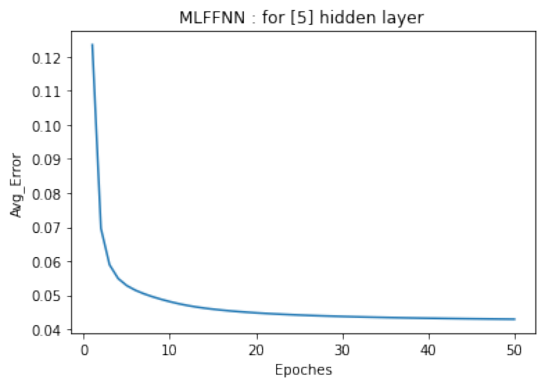


Figure 28: hidden-layers = [5]

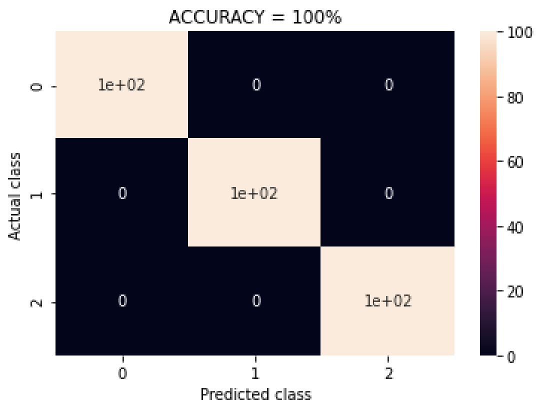


Figure 29: hidden-layers = [5]

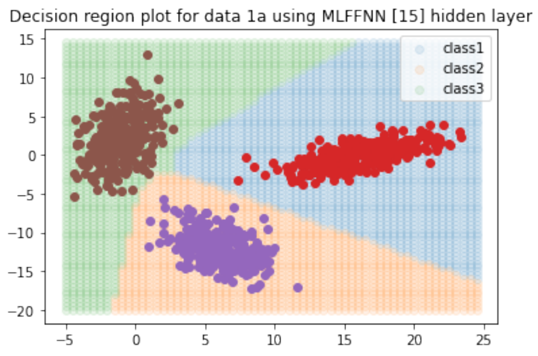


Figure 30: hidden-layers = [15]

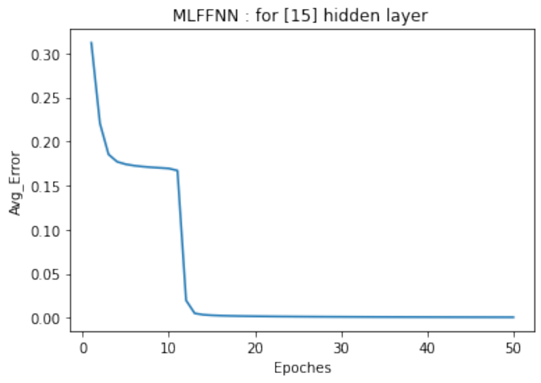


Figure 31: hidden-layers = [15]

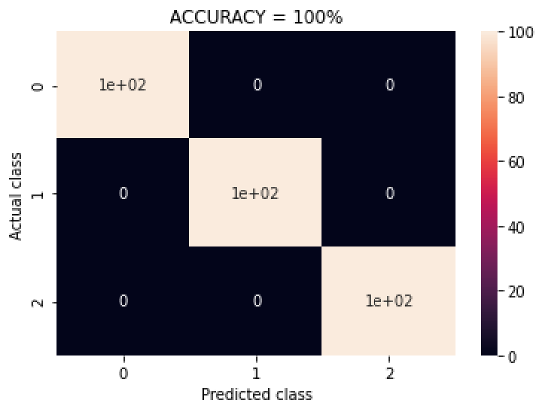


Figure 32: hidden-layers = [15]

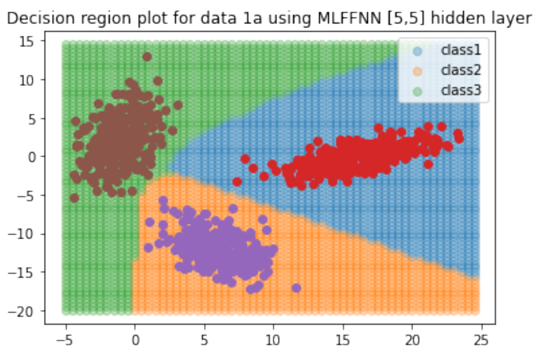


Figure 33: hidden-layers = [5,5]

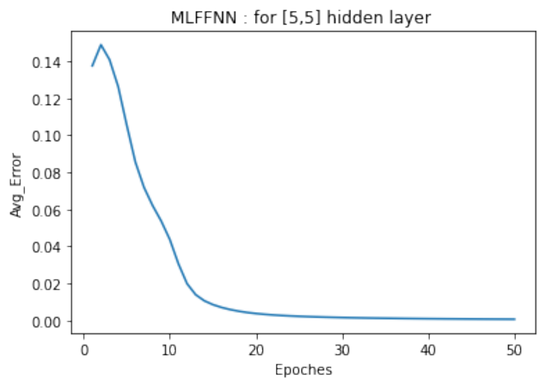


Figure 34: hidden-layers = [5,5]

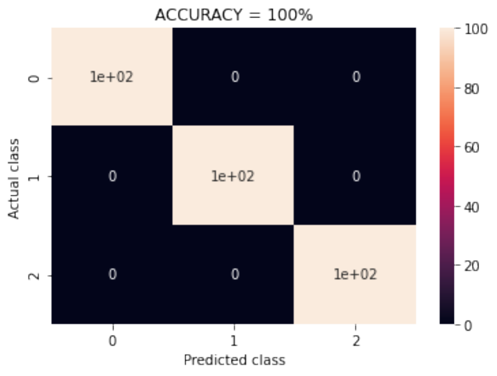


Figure 35: hidden-layers = [5,5]

Inferences from the above plots of Linearly separable data using MLFFNN:

- MLFFNN very easily classifies linearly separable data , in not more than 12 epochs , with 100% accuracy .All the error graphs converges at 10-15 epochs .
- Even after changing the lr by small amount , the accuracy of the convergence of graphs does not change .
- The data is very easily separable as all the models are classifying with 100% accuracy .

3.1.4 Non-Linearly separable data using multi layer feed forward neural network

For all the experiment results shown here learning rate = 0.1 . I have used a different convention for hidden layer representation [x,y] means 2 hidden layers are there with x and y number of neurons respectively .

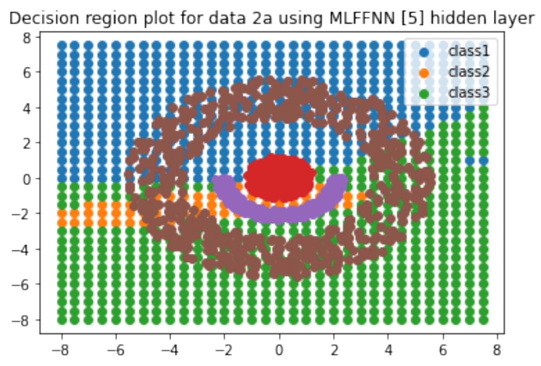


Figure 36: hidden-layers = [5], epochs = 50

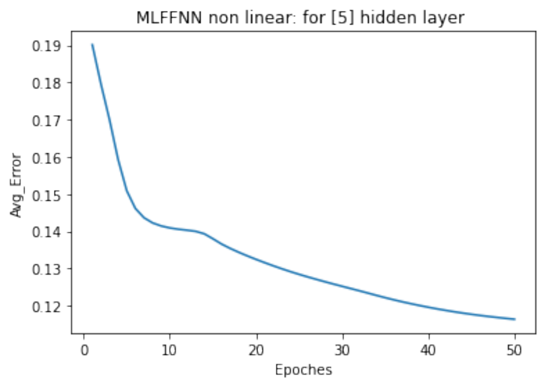
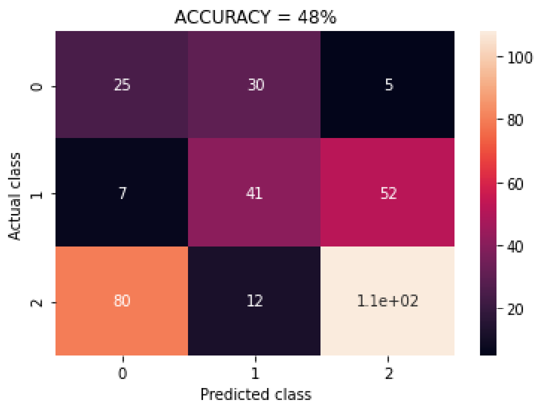


Figure 37: hidden-layers = [5],epochs = 50



	0	1	2
0	25	30	5
1	7	41	52
2	80	12	1.1e+02

Figure 38: hidden-layers = [5],epochs = 50

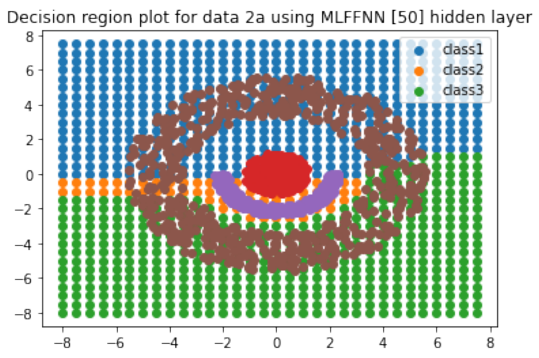


Figure 39: hidden-layers = [50],epochs = 500

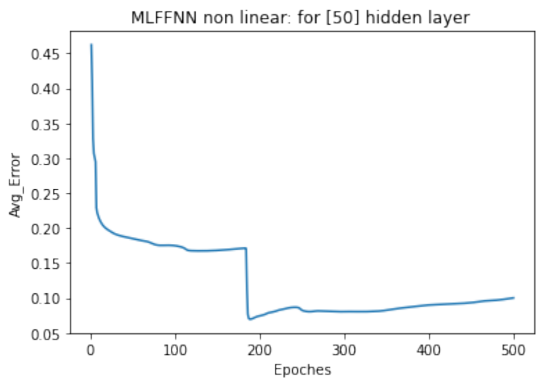
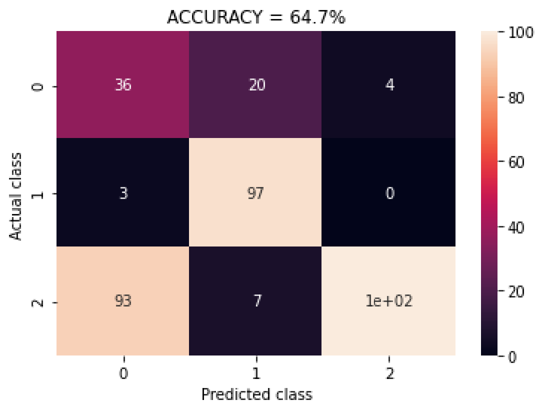


Figure 40: hidden-layers = [50],epochs = 500



	0	1	2
0	36	20	4
1	3	97	0
2	93	7	1e+02

Figure 41: hidden-layers = [50],epochs = 50

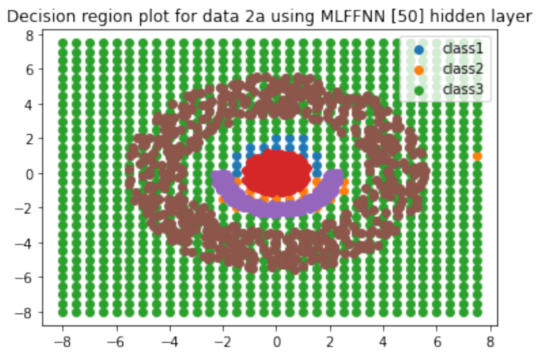


Figure 42: hidden-layers = [15,15],epochs = 500

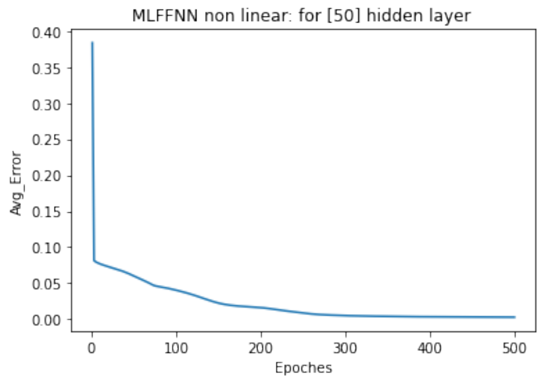


Figure 43: hidden-layers = [15,15],epochs = 500

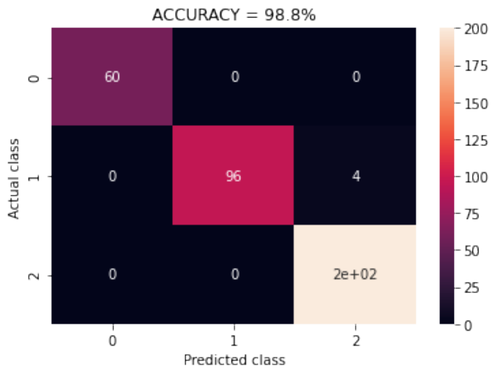


Figure 44: hidden-layers = [15,15],epochs = 500

Inferences from the above plots for non-linearly separable data:

- As the number of hidden layer increases , the accuracy of model increases .
- In our case, perceptron will never converge for large number of epochs and maximum accuracy found on test sample is 98.8% .

3.2 Regression Task Results

3.2.1 Univariate data using perceptron with linear activation function

Plots of epochs (x-axis) vs average error (y-axis) for different learning rates

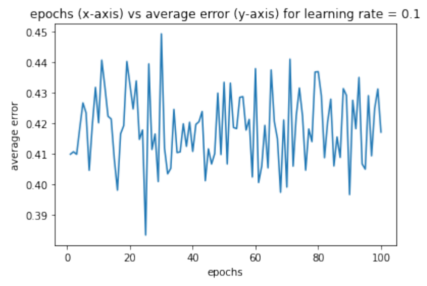


Figure 45: Learning rate = 0.1

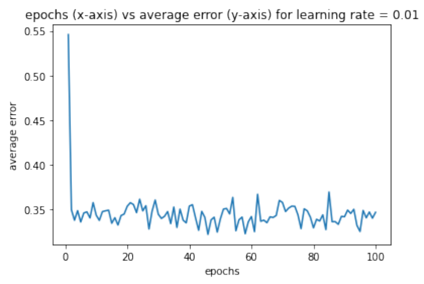


Figure 46: Learning rate = 0.01

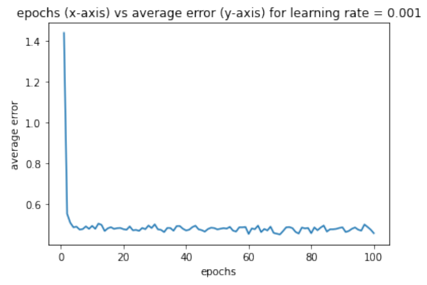


Figure 47: Learning rate = 0.001

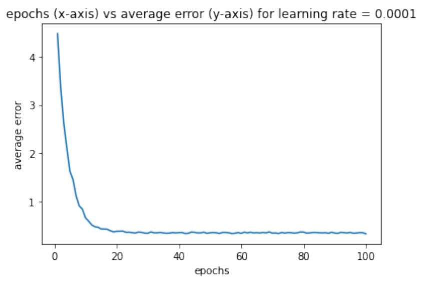


Figure 48: Learning rate = 0.0001

Plots of model output vs target output for training data, validation data and test data learning rate = 0.0001

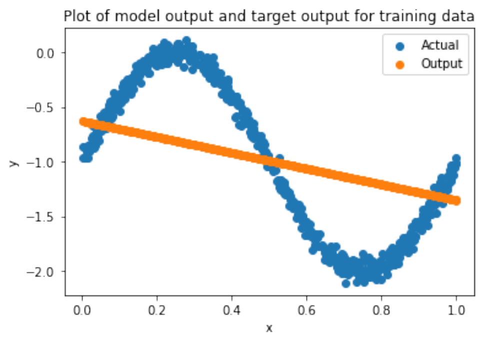


Figure 49: Training data

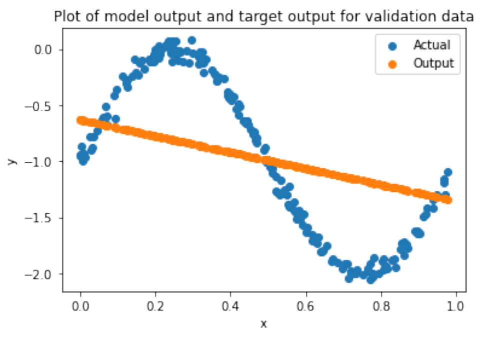


Figure 50: Validation data

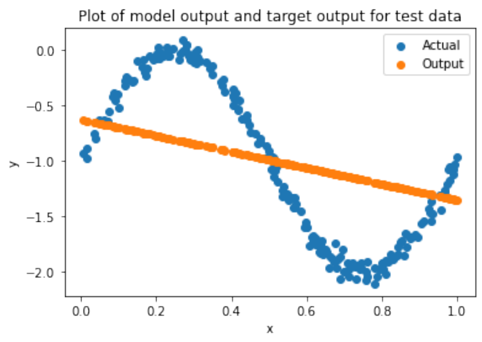


Figure 51: Test data

Scatter plots with target output on x-axis and model output on y-axis, for training data, validation data and test data for learning rate = 0.0001

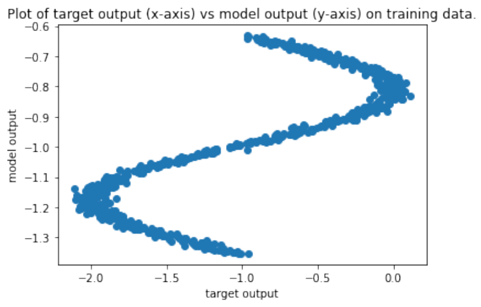


Figure 52: Training data

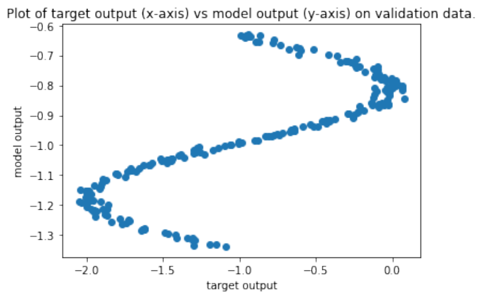


Figure 53: Validation data

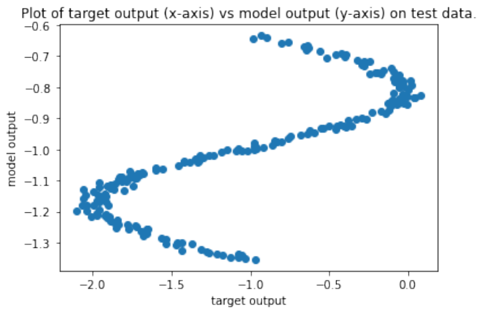


Figure 54: Test data

Inferences from the plots:

- A large learning rate can cause the model to converge to a suboptimal solution as seen in the case of learning rate = 0.1.
- A small learning rate allows the model to learn a more optimal solution. But it takes more number of epochs to converge i.e. 3 epochs for learning rate = 0.001 and 22 epochs for learning rate = 0.0001.
- Hence the optimal learning rate for our model should be 0.0001.
- Straight line fitting in case of a single input.

3.2.2 Bivariate data using perceptron with linear activation function

Plots of epochs (x-axis) vs average error (y-axis) for different learning rates

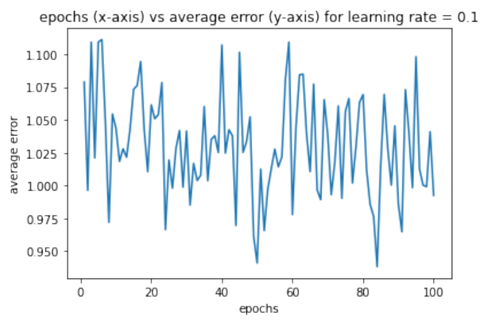


Figure 55: Learning rate = 0.1

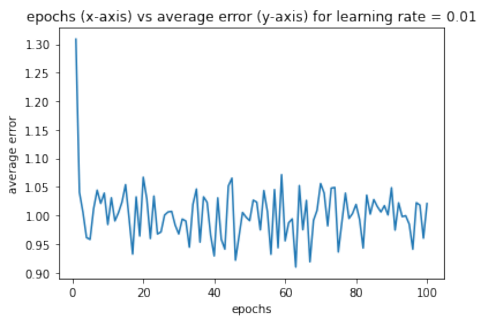


Figure 56: Learning rate = 0.01

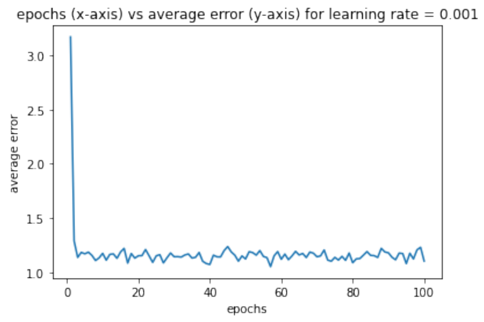


Figure 57: Learning rate = 0.001

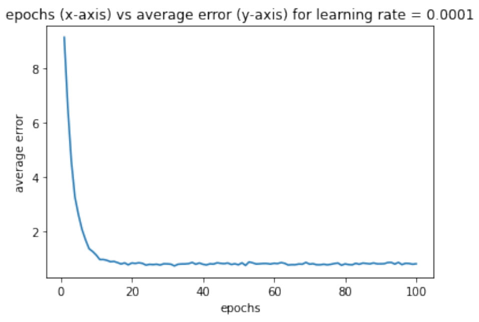


Figure 58: Learning rate = 0.0001

Plots of model output vs target output for training data, validation data and test data learning rate = 0.0001

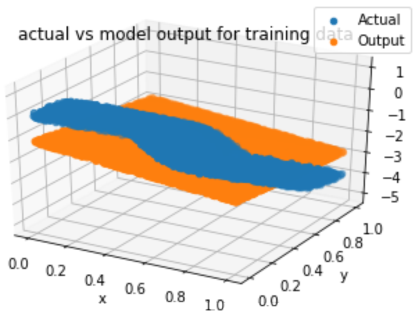


Figure 59: Training data

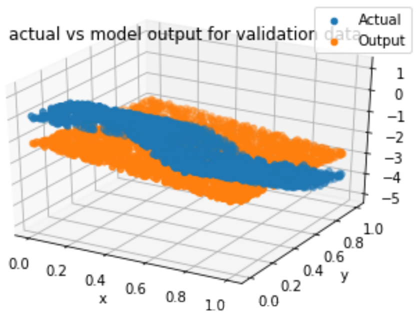


Figure 60: Validation data

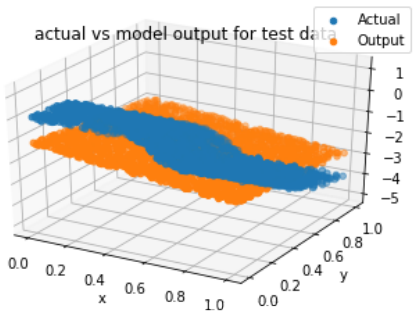


Figure 61: Test data

Scatter plots with target output on x-axis and model output on y-axis, for training data, validation data and test data for learning rate = 0.0001

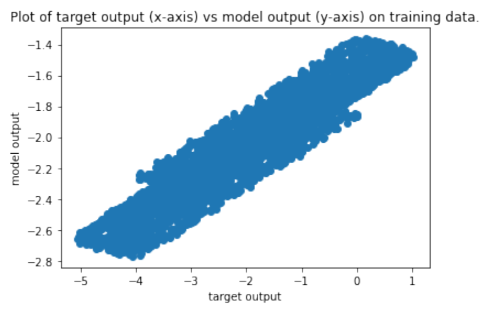


Figure 62: Training data

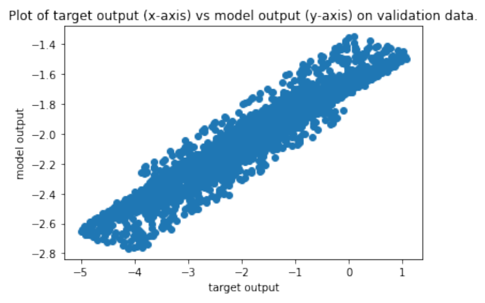


Figure 63: Validation data

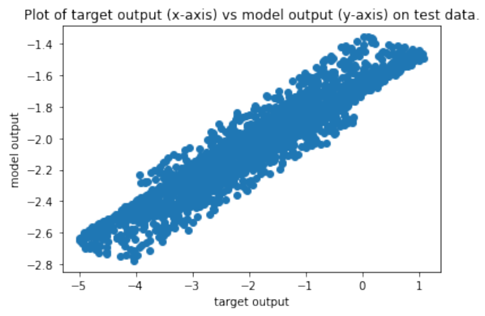


Figure 64: Test data

Inferences from the plots:

- A large learning rate can cause the model to converge to a suboptimal solution as seen in the case of learning rate = 0.1.
- A small learning rate allows the model to learn a more optimal solution. But it takes more number of epochs to converge i.e. 3 epochs for learning rate = 0.001 and 18 epochs for learning rate = 0.0001.
- Hence the optimal learning rate for our model should be 0.0001.
- Linear surface (hyperplane) fitting in case of a single input.

3.2.3 Univariate data using MLFFNN

Plots of epochs (x-axis) vs average error (y-axis) for different model complexities

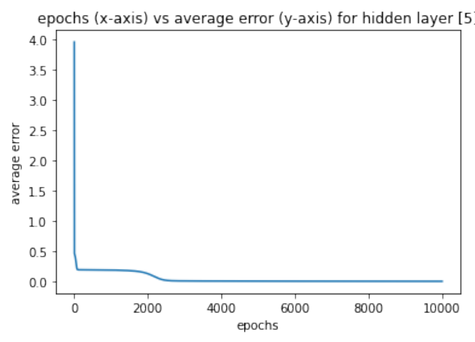


Figure 65: One hidden layer with 5 perceptrons

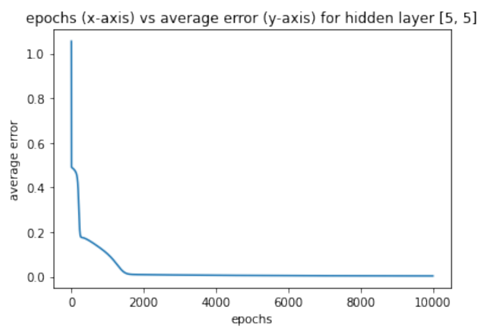


Figure 66: Two hidden layers with 5 perceptrons each

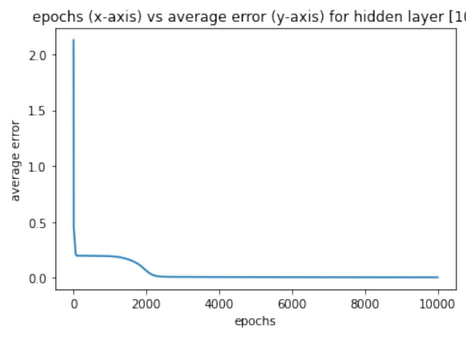


Figure 67: One hidden layer with 10 perceptrons

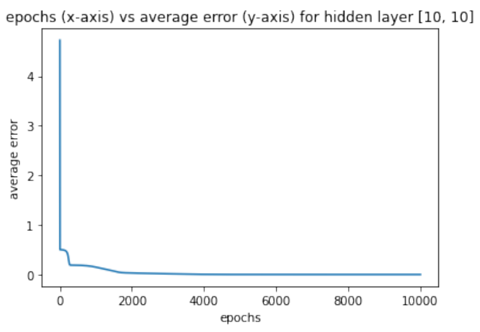


Figure 68: Two hidden layers with 10 perceptrons each

Plots of the values of mean squared error (MSE) on training data, validation data and test data for different model complexities

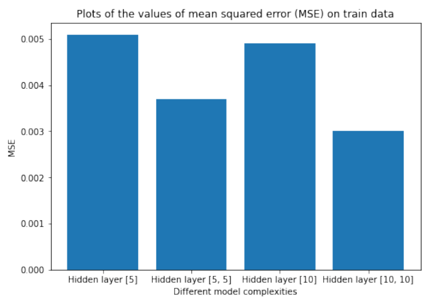


Figure 69: Training data

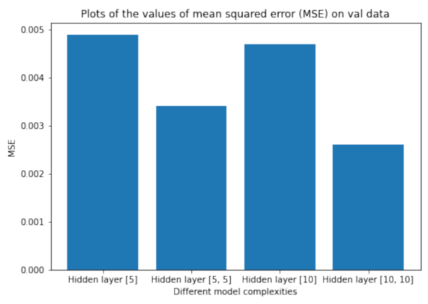


Figure 70: Validation data

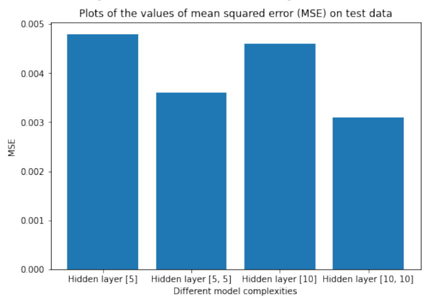


Figure 71: Test data

Plots of model output vs target output for training data, validation data and test data with two hidden layers of 10 perceptrons each

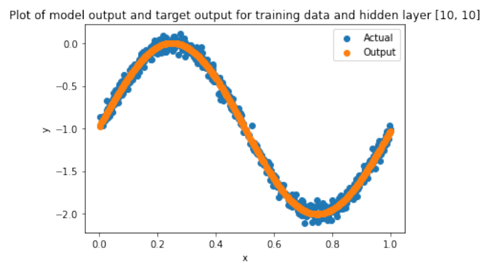


Figure 72: Training data

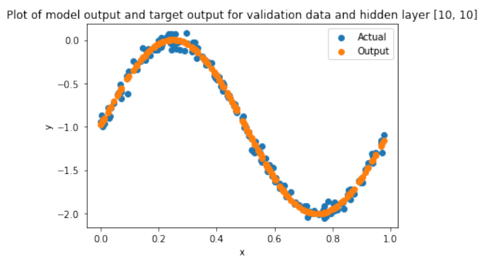


Figure 73: Validation data

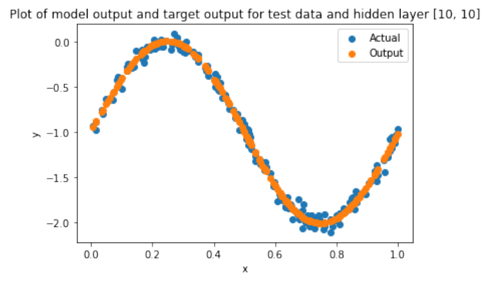


Figure 74: Test data

Scatter plots with target output on x-axis and model output on y-axis, for training data, validation data and test data with two hidden layers of 10 perceptrons each

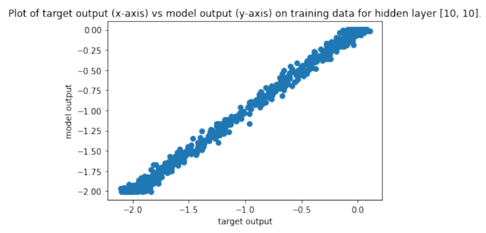


Figure 75: Training data

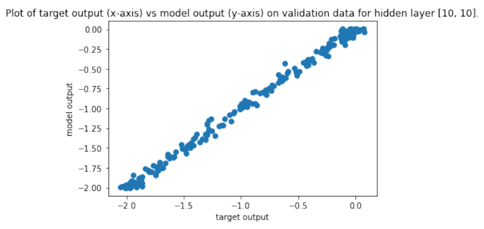


Figure 76: Validation data

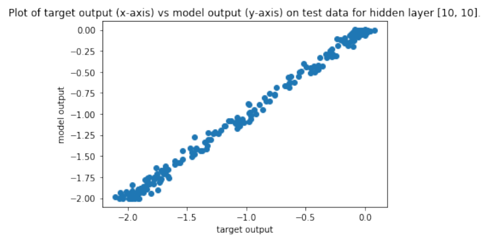


Figure 77: Test data

Inferences from the plots:

- Each of the tested models with the hidden layers given as [5], [5, 5], [10] and [10, 10] converges within 3000 epochs.
- Average error is almost zero in each case.
- The model with the minimum MSE for training data, validation data and test data is the one with two hidden layers of 10 perceptrons each.
- Model output almost fits the actual output.
- Plot of model output vs actual output is almost a straight line y = x.

3.2.4 Bivariate data using MLFFNN

Plots of epochs (x-axis) vs average error (y-axis) for different model complexities

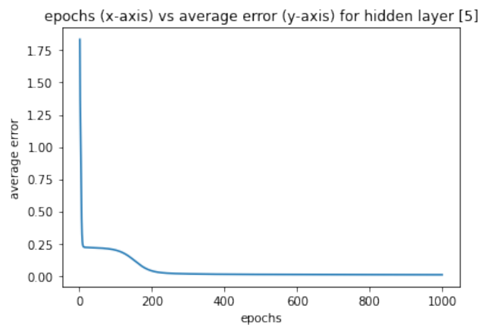


Figure 78: One hidden layer with 5 perceptrons

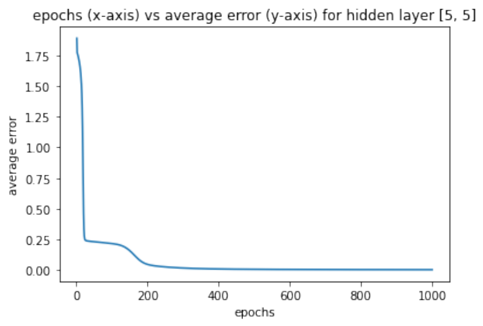


Figure 79: Two hidden layers with 5 perceptrons each

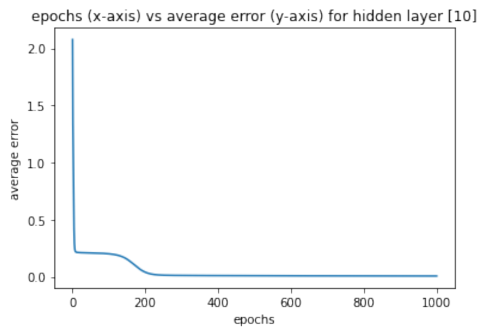


Figure 80: One hidden layer with 10 perceptrons

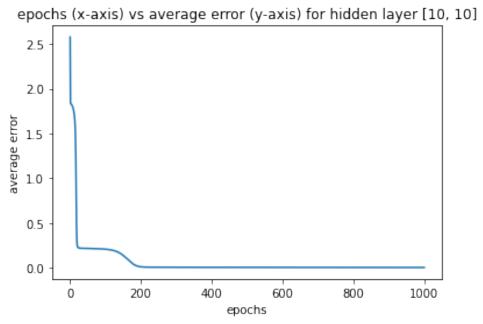


Figure 81: Two hidden layers with 10 perceptrons each

Plots of the values of mean squared error (MSE) on training data, validation data and test data for different model complexities

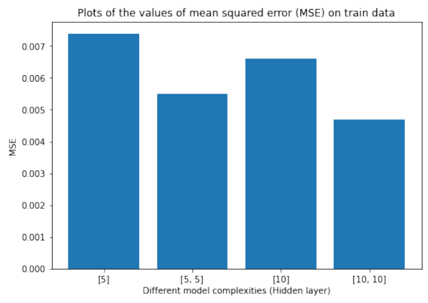


Figure 82: Training data

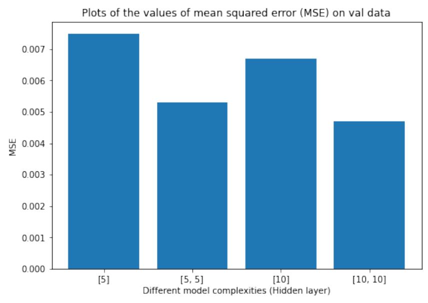


Figure 83: Validation data

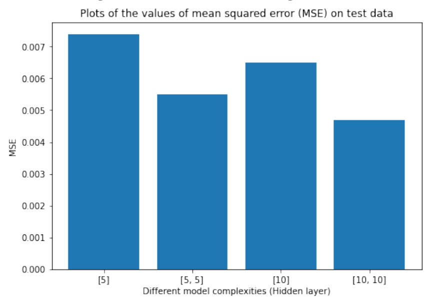


Figure 84: Test data

Plots of model output vs target output for training data, validation data and test data with two hidden layers of 10 perceptrons each

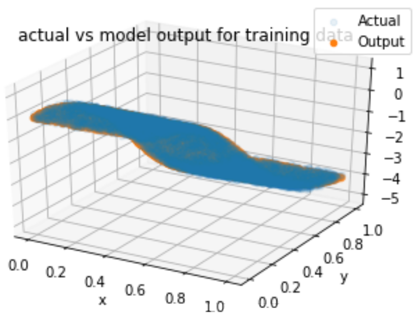


Figure 85: Training data

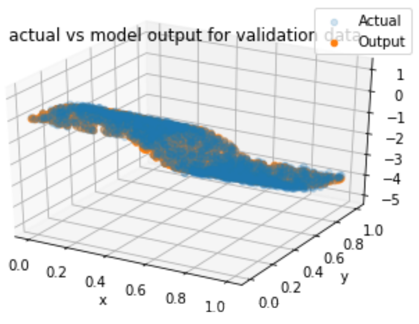


Figure 86: Validation data

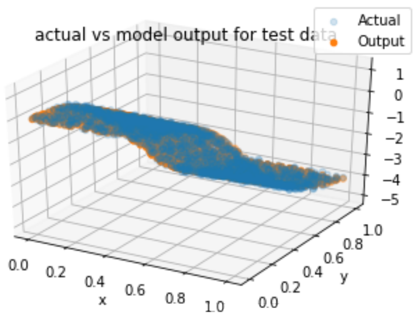


Figure 87: Test data

Scatter plots with target output on x-axis and model output on y-axis, for training data, validation data and test data with two hidden layers of 10 perceptrons each

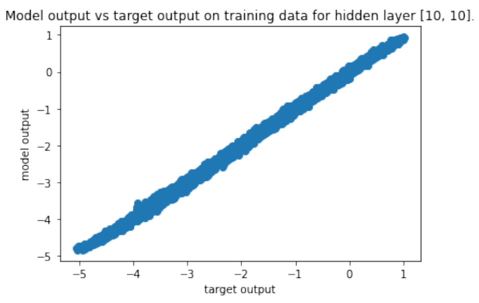


Figure 88: Training data

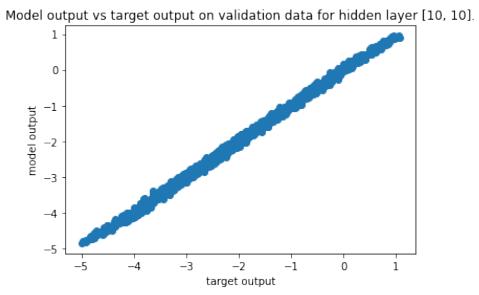


Figure 89: Validation data

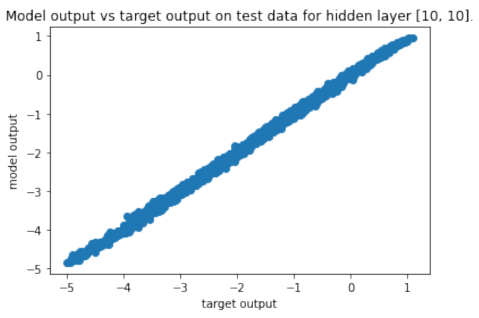


Figure 90: Test data

Inferences from the plots:

- Each of the tested models with the hidden layers given as [5], [5, 5], [10] and [10, 10] converges within 300 epochs.
- Average error is almost zero in each case.
- The model with the minimum MSE for training data, validation data and test data is the one with two hidden layers of 10 perceptrons each.
- Model output almost covers the actual output.
- Plot of model output vs actual output is almost a straight line y = x.

4 Image Classification

4.1 About the data

We are given images of 40 images of elevator shaft , 40 images of bus-interior and 40 images of botenical-garden for training our model and 10 images from each data set for validation data and 50 images as testing data .

we calculated Bag of visual words representaion for the images given . and trained out model for 2 different MLFFNN-architectures .

- One with [24] hidden layers and 100 epochs and learning rate = 0.01
- other with [50,50] hidden layers and 100 epochs and learning rate = 0.002.

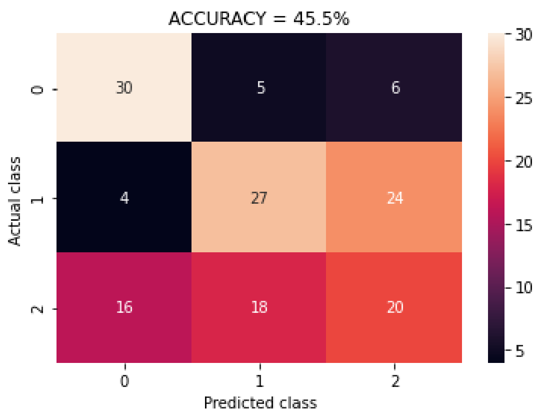


Figure 91: hidden-layers = [24]

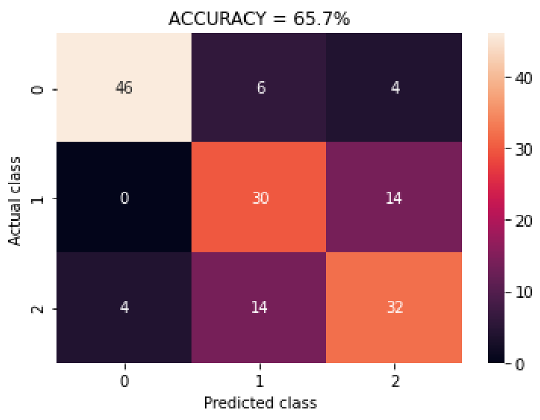


Figure 92: hidden-layers = [50,50]

Inferences from the above plots :

- The training size of image is very small , the model is not able to learn all the properties of images .
- What I saw from image , Botanical garden and bus interior images have very similar colour shades , we also know BOVW is colour representation of image . so the confusion between these images are very high.
- Most of the images from class 1 (Elevator Shaft) are colourless(Black-white-brown) , the confusion is less for classification of this type .

5 Model Comparison

5.1 Classification

Perceptron Model Details(lr=0.1)	Accuracy(LS data)	Accuracy(NLS data)
lr=0.1 and epochs = 50	94.7%	39.2%
lr=0.2 and epochs = 10	89.3%	40.2%
lr=0.5 and epochs = 1000	100%	45.1%
lr=0.2 and epochs = 1000	100%	46.2%
lr=0.1 and epochs = 1000	100%	52%

MLFFNN Model Details	Accuracy(LS data)	Accuracy(NLS data)
epochs = 50 hidden = [5]	100%	48%
epochs = 50 hidden = [15]	100%	50.1%
epochs = 50 hidden = [5,5]	100%	54%
epochs = 500 hidden = [50]	100%	64%
epochs = 500 hidden = [15,15]	100%	98.8%

5.2 Regression

MSE of the training data, validation data and test data for different model complexities is given in the following tables.
1-dimensional (Univariate) input data

Training data

Perceptron with lr = 0.0001	0.4369
MLFFNN with hidden layer [5]	0.0051
MLFFNN with hidden layer [5, 5]	0.0037
MLFFNN with hidden layer [10]	0.0049
MLFFNN with hidden layer [10, 10]	0.0030

Validation data

Perceptron with lr = 0.0001	0.5004
MLFFNN with hidden layer [5]	0.0049
MLFFNN with hidden layer [5, 5]	0.0034
MLFFNN with hidden layer [10]	0.0047
MLFFNN with hidden layer [10, 10]	0.0026

Test data

Perceptron with lr = 0.0001	0.5215
MLFFNN with hidden layer [5]	0.0048
MLFFNN with hidden layer [10]	0.0036
MLFFNN with hidden layer [5, 5]	0.0046
MLFFNN with hidden layer [10, 10]	0.0031

2-dimensional (Bivariate) input data

Training data

Perceptron with lr = 0.0001	1.9682
MLFFNN with hidden layer [5]	0.0074
MLFFNN with hidden layer [5, 5]	0.0055
MLFFNN with hidden layer [10]	0.0066
MLFFNN with hidden layer [10, 10]	0.0047

Validation data

Perceptron with lr = 0.0001	1.9380
MLFFNN with hidden layer [5]	0.0075
MLFFNN with hidden layer [5, 5]	0.0053
MLFFNN with hidden layer [10]	0.0067
MLFFNN with hidden layer [10, 10]	0.0047

Test data

Perceptron with lr = 0.0001	1.3293
MLFFNN with hidden layer [5]	0.0074
MLFFNN with hidden layer [5, 5]	0.0055
MLFFNN with hidden layer [10]	0.0065
MLFFNN with hidden layer [10, 10]	0.0047

References

[1] https://machinelearningmastery.com/implement-perceptron-algorithm-scratch-python/

[2] https://towardsdatascience.com/building-neural-network-from-scratch-9c88535bf8e9

[3] https://medium.com/@aybukeyalcinerr/bag-of-visual-words-bovw-db9500331b2f