A Report on

CS671: Assignment 2

Classification and Regression using FCNN

Team Members(Group 04):

Ananya Angra(S22025)
Shilpa Chandra(S22004)
Pushkar Kumar(S22038)

Course Instructor

Dr. Dileep A. D.

IIT Mandi Academic Year 2022-2023

Contents

List of Figures

List of Tables

Chapter 1

Fully Connected Neural Network(FCNN)

FCNN stands for Fully Convolutional/Connected Neural Network, which is a type of deep neural network used for image segmentation, object detection, and other computer vision tasks.

1.1 Motovation behind FCNN

- As seen in Fig 1.1 Single neuron based model is not sufficient for classification of non linear separable data
- A network of neurons (neural network) is used that approximates a nonlinear boundary/nonlinear surfaceby stitching the boundary/surface from each neuron

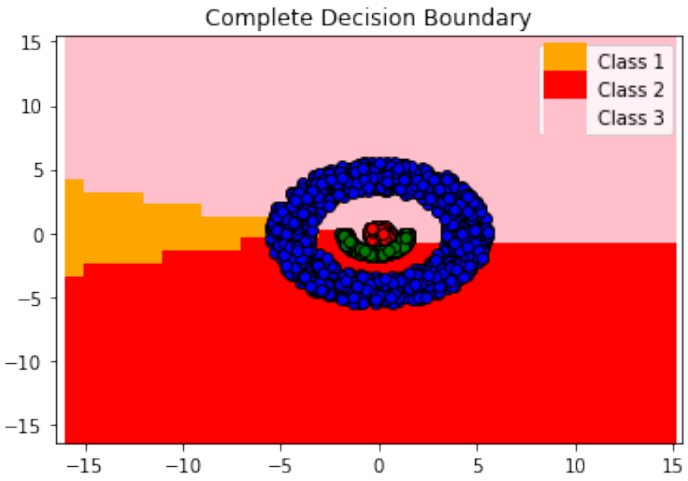


Figure 1.1: Decision boundary of 3 classes using single perceptron

1.2 Architecture of FCNN

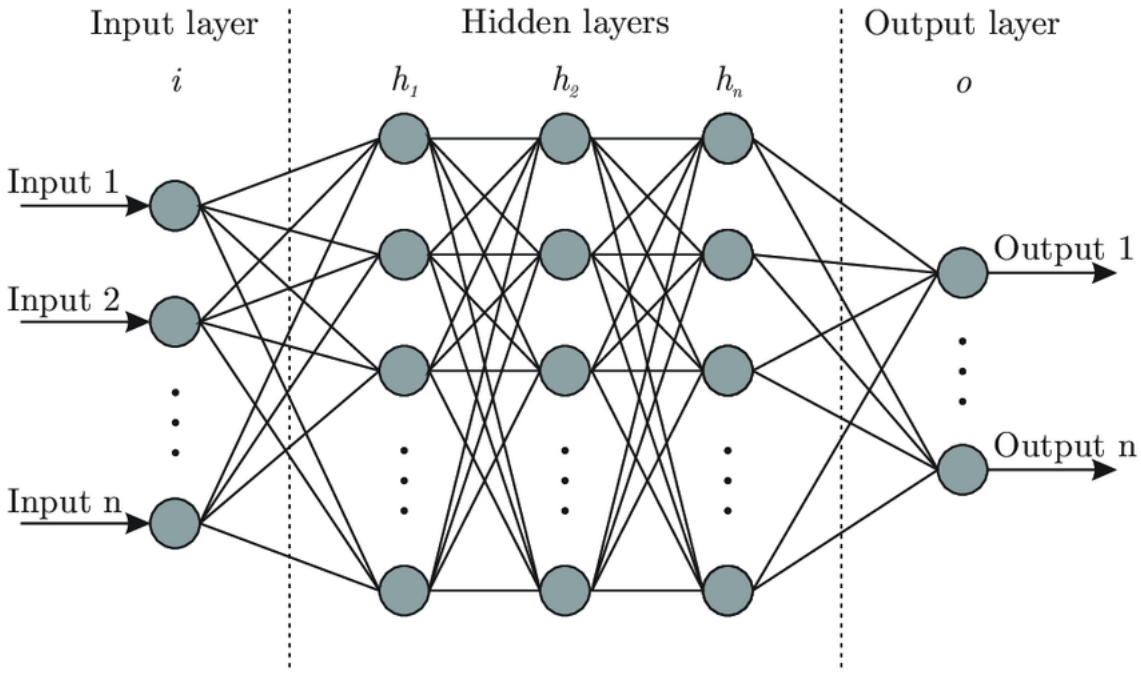


Figure 1.2: Architecture of FCNN [1]

- Input Layer: Linear neurons
  - Linear neuron: When an input is given to a neuron and the sameinput comes out as output
  - Number of layers in input layer = Dimension of the data vector
- Hidden layers: (1 or 2 or more): Sigmoidal neurons
  - Sigmoid neuron: Neuron with sigmoid activation function
  - Number of layers and neurons in each of the hidden layers are decidedexperimentally
- Output layer: Sigmoidal neurons (for pattern classificationtask) OR Linear neurons (for regression task)
  - Number of neurons in the output layer = Number of classes in classification or Number of output variables in regression

Weights associated with all the connection between the neurons indicate the parameter of the complex nonlinear discriminant function/nonlinear surface that the network is trying to approximate

The activation function used in this assignment is Logistic activation function
$f(a) = \frac{1}{1+e^{-(\beta a)}}$
$\frac{df(a)}{f(a)} = \beta f(a)(1 - f(a))$

Learning Menthod: Error correction learning (Backpropagation algorithm)

1.3 Backpropagation learning

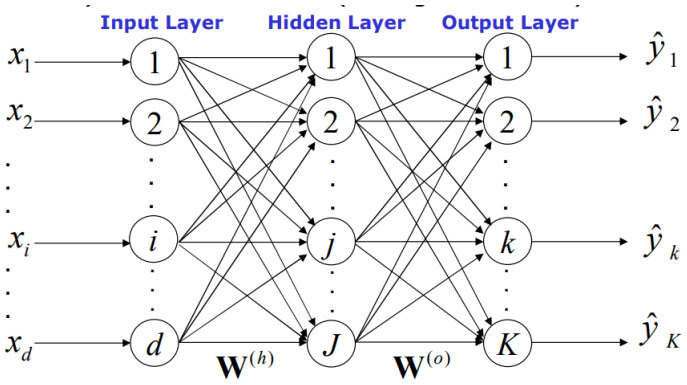


Figure 1.3: Architecture of FCNN [2]

Parameter Learning (Pattern Mode) – Stochastic Gradient Descent
STEPS:

1. Initialize the $\mathbf{W^{(h)}}$ and $\mathbf{W^{(o)}}$ with random values
2. Randomly choose a training example $\hat{x_{\mathrm{i}}}$
3. Forward computation:
   Compute output of all output neurons $(k = 1, 2, \ldots K) : \hat{y_{\mathrm{nk}}}$
4. Backward operation:
   Compute instantaneous error:

$$E_{\mathrm{n}} = \frac{1}{2}(y_{\mathrm{nk}} - \hat{y_{\mathrm{nk}}})^2$$

   Update weights between hidden and output layer (j=1, 2, ... J and k=1, 2, ... K)
5. Repeat steps 2 to 4 till all the training examples are presented once

6. Compute the average error:

$$E_{\text{av}} = \frac{1}{N} \sum_{i=1}^{N} E_{\text{n}}(\mathbf{w})$$

Convergence Criteria:

– A fixed number of epoch is reached

– Absolute difference between average error of successive epochs fall below a threshold ($E.g. threshold = 10^{-3}$)

Chapter 2

Classification Task

2.1 Linearly Separable Classes Results

The algorithm is tested for linearly separable data shown below:

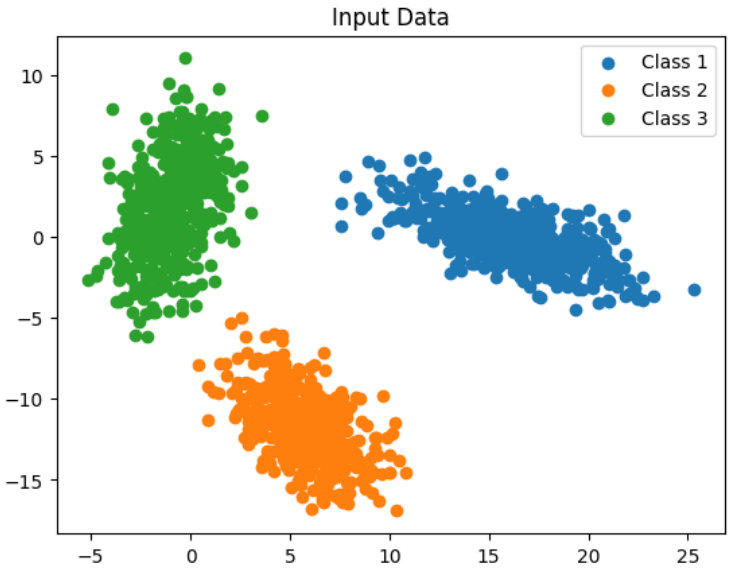


Figure 2.1: Input data for Linearly separable data

- $\eta = 0.01$ The weights in the hidden layer are initialised randomly using np.random.randn()

The weights in the output layer are initialised using np.random.randn()

Number of epochs 100 required to achieve convergence - this is the stopping criteria used here.

Different Model Architectures

The dataset was tested on differnet architectures and the results are shown below based on the validation dataset provided.

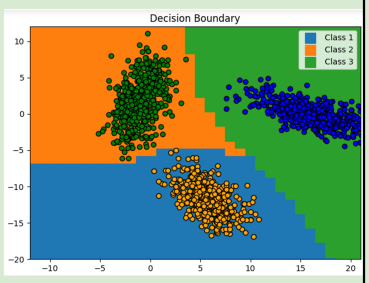
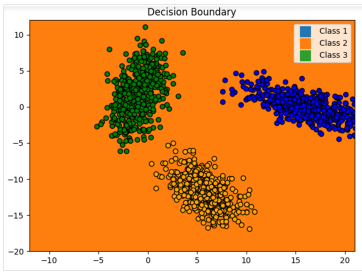
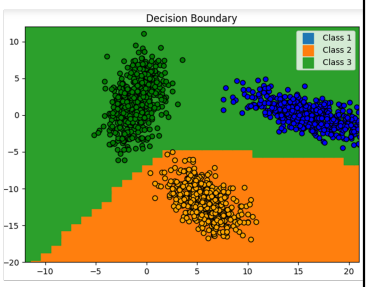
No of neurons	Decision Boundary	Confusion Matrix	Accuracy	Precision	Recall	F-Measure
4		[[100 0 0] [0 100 0] [0 0 100]]	100%	1.0	1.0	1.0
1		[[0 100 0] [0 100 0] [0 100 0]]	33%	0.33	0.06	0.11
2		[[0 0 100] [0 100 0] [0 2 98]]	66%	0.65	0.43	0.49

Table 2.1: Variation of architecture for linear data

BEST ARCHITECTURE

As seen in Table 2.1 the best architecture for linear data is when we have 4 neurons in the hidden layer. The confusion matrix and precision, recall and f1-score is also presented below

Confusion Matrix

Predicted Values				
Actual Values		Class 1	Class 2	Class 3
	Class 1	100	0	0
	Class 2	0	100	0
	Class 3	0	0	100

Table 2.2: Confusion Matrix of Linearly separable data

	Precision	Recall	F1 - score
Class 1	100%	100%	100%
Class 2	100%	100%	100%
Class 3	100%	100%	100%
Average	100%	100%	100%

Table 2.3: Accuracy, Precision, Recall and F-1 score of Non Linearly separable data for the three classes

The following plots from Fig 2.2 till Fig 2.8 is based on the best architecture

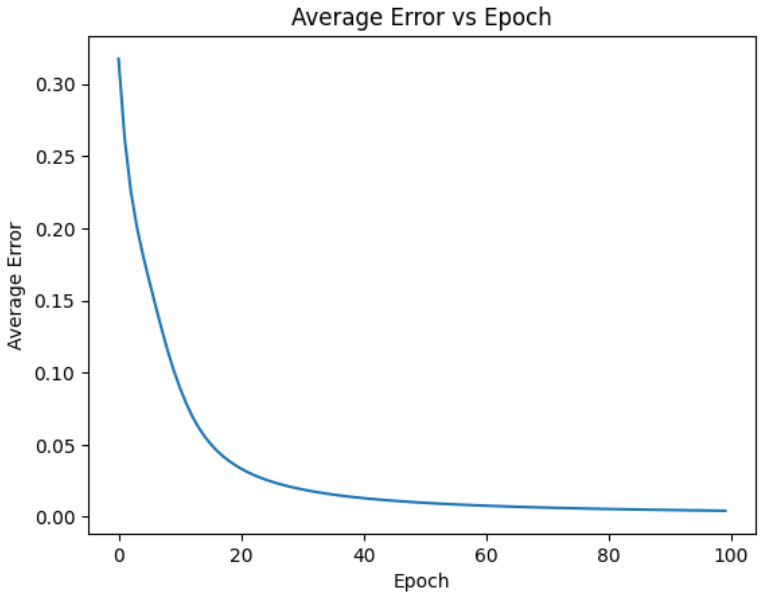
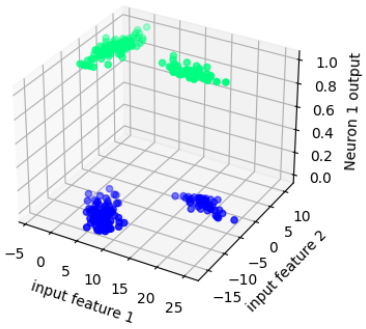
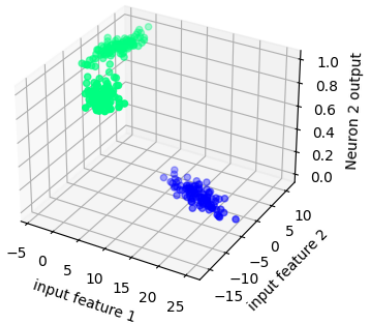


Figure 2.2: Plot of average error (y-axis) vs epochs (x-axis)

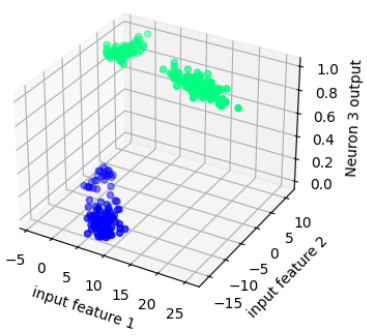
From Fig 2.2 it can be inferred that 100 epochs are enough to train the model as the error starts to converge after approximately 100 epochs.



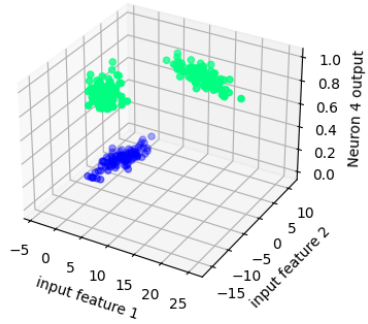
(a) Fig: neuron 1



(b) Fig: neuron 2

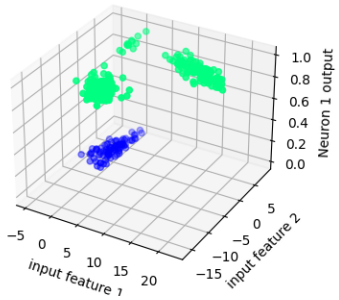


(c) Fig: neuron 3

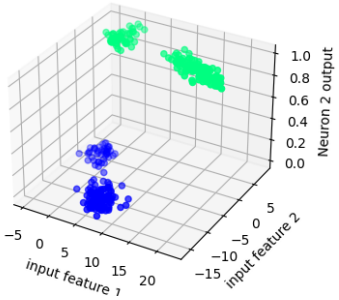


(d) Fig: neuron 4

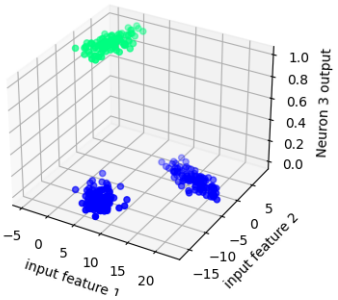
Figure 2.3: Output of neurons in single hidden layer for test data



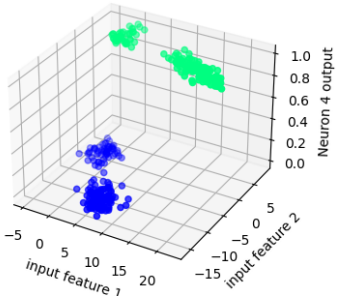
(a) Fig: neuron 1



(b) Fig: neuron 2



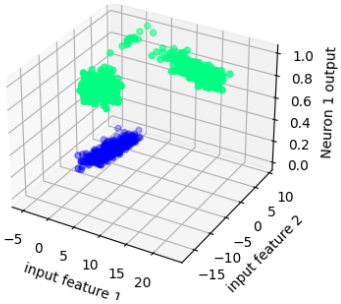
(c) Fig: neuron 3



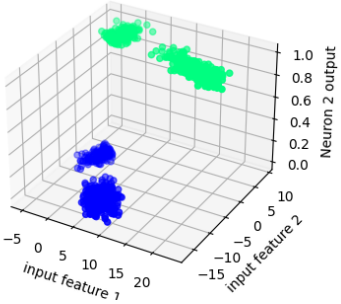
(d) Fig: neuron 4

Figure 2.4: Output of neurons in single hidden layer for validation data

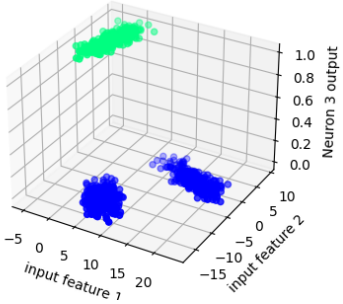
---



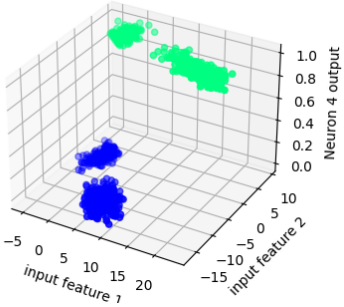
(a) Fig: neuron 1



(b) Fig: neuron 2



(c) Fig: neuron 3



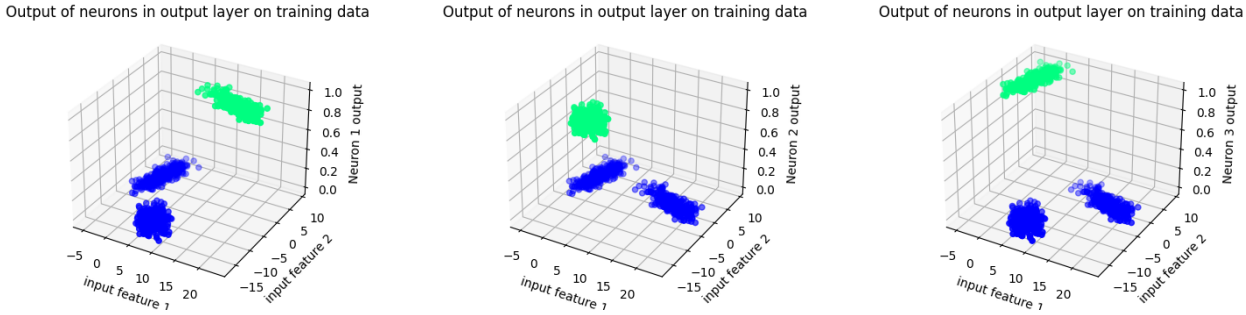
(d) Fig: neuron 4

Figure 2.5: Output of neurons in single hidden layer for training data

---

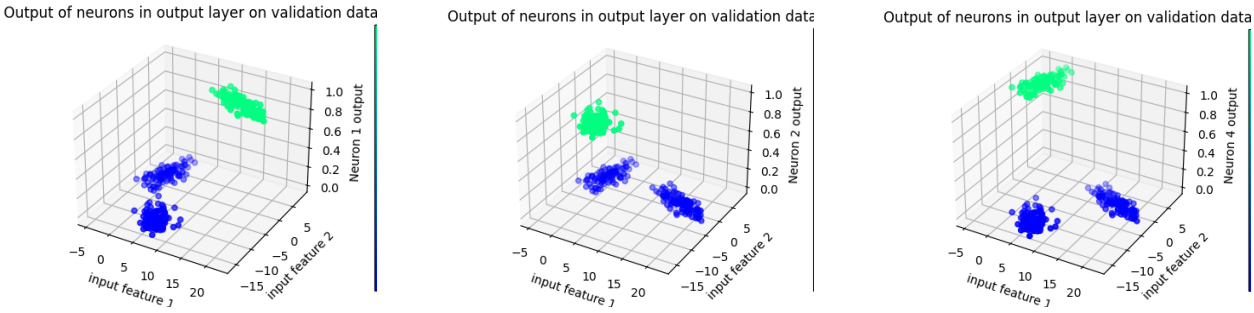
Figure 2.3,2.4,2.5 show the outputs for each of the hidden nodes in FCNN for each of the datasets after the model is trained. x and y axis are input variables of each example, z axis is output of hidden node.The plots are

obtained for training , validation and test data.It shows how each neuron learns the shape of the data.



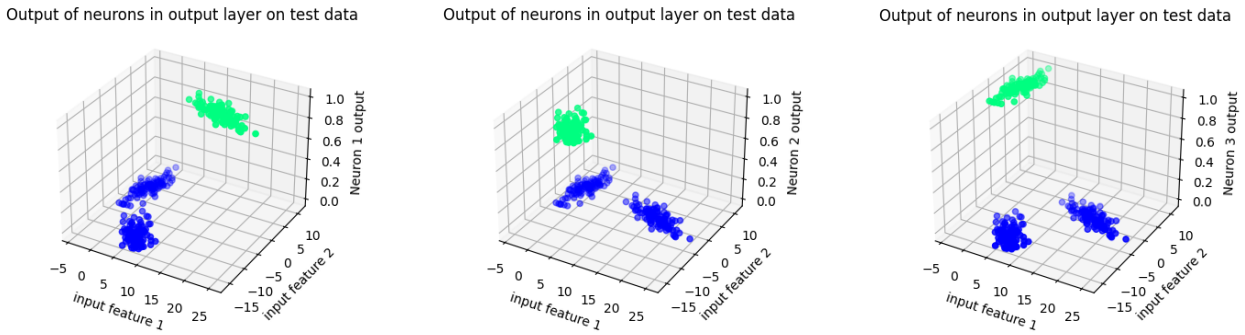
(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

Figure 2.6: Output of neurons in output layer for training data



(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

Figure 2.7: Output of neurons in output layer for validation data



(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

Figure 2.8: Output of neurons in output layer for testing data

Fig 2.6,2.7,2.8 show the scatter plot of each of the output neurons for training, validation, and test data.

2.2 Non Linearly Separable Classes Results

The algorithm is tested for non linearly separable data shown below:

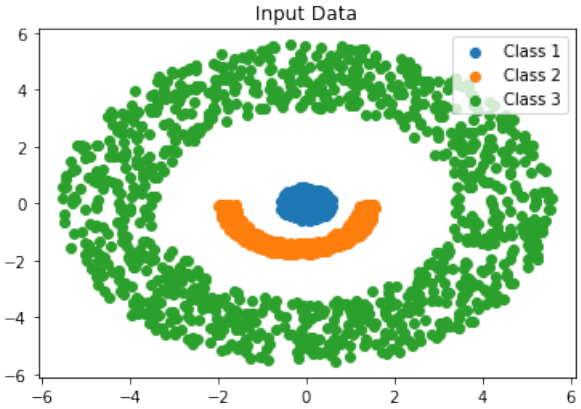


Figure 2.9: Input data for Non Linearly separable data

The parameters used to train univariate data:
- $\eta = 0.01$
- epochs = As the dataset for non linear data required more training to form a proper boundary and also the model converges after only close to 850 epochs as seen in Fig 2.10 (on the best architecture). This is the stopping criteria used here.

The dataset was tested on different architectures and the results are shown below based on the validation dataset provided.

Different Model Architectures

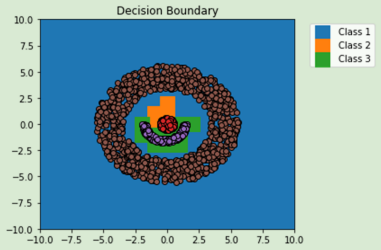
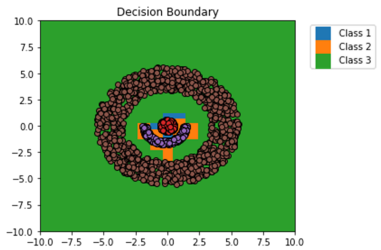
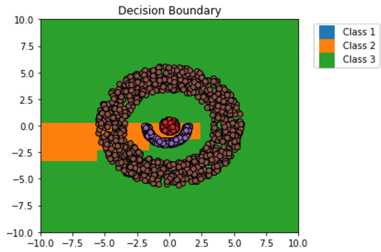
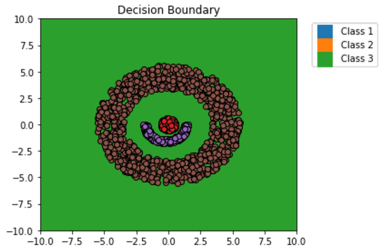
No of neurons	Decision Boundary	Confusion Matrix	Accuracy	Precision	Recall	F-Measure
8 in 1st layer and 3 in 2nd layer		[[60 0 0] [0 100 0] [0 0 200]]	100%	1.0	1.0	1.0
7 in 1st layer and 3 in 2nd layer		[[0 13 47] [0 100 0] [0 0 200]]	83.33%	0.66	0.49	0.56
2 in 1st layer and 2 in 2nd layer		[[0 45 15] [0 99 1] [0 21 179]]	77.27%	0.59	0.42	0.47
1 in 1st layer and 1 in 2nd layer		[[0 0 60] [0 0 100] [0 0 200]]	55.55%	0.33	0.12	0.18

Table 2.4: Variation of architecture for nonlinear data

BEST ARCHITECTURE

As seen in table 2.2 the best architecture for non linear data is when we have 8 neuron in hidden layer 1 and 3 neurons in hidden layer 2. The confusion matrix and precision, recall and f1-score is also presented below

	Precision	Recall	F1 - score
Class 1	100%	100%	100%
Class 2	100%	100%	100%
Class 3	100%	100%	100%
Average	100%	100%	100%

Table 2.5: Accuracy, Precision, Recall and F-1 score of Non Linearly separable data for the three classes

Below are the plots for output of neurons for first, second and output layer for the best architecture on training, testing and validation dataset.

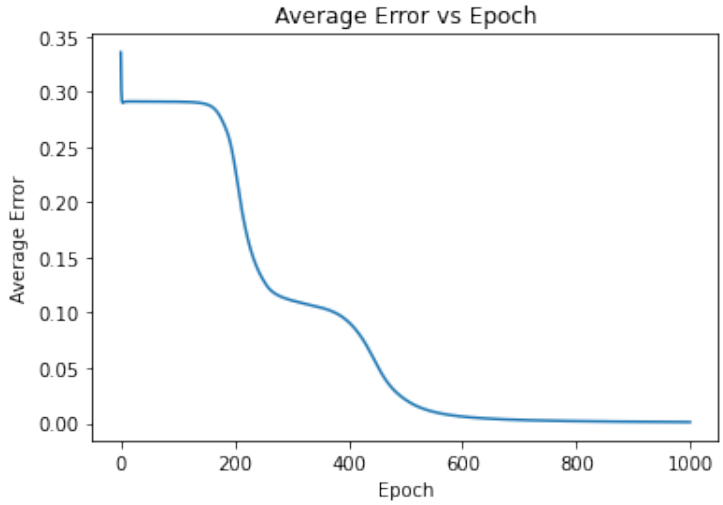
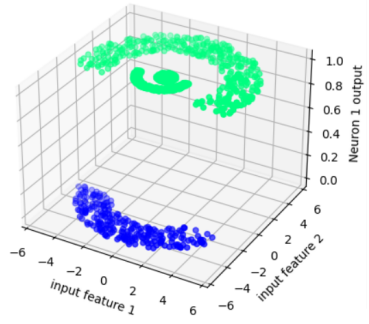
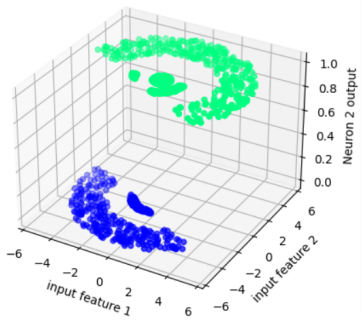


Figure 2.10: Plot of average error (y-axis) vs epochs (x-axis)

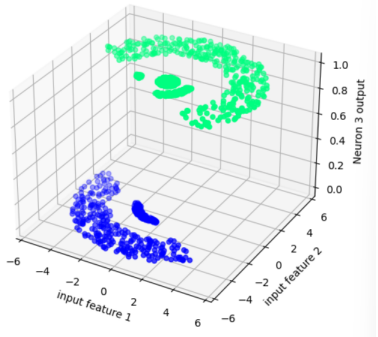
Output of neurons for first, second and output layer for the best architecture on training dataset



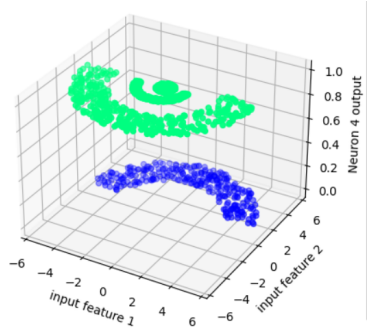
(a) Fig: neuron 1



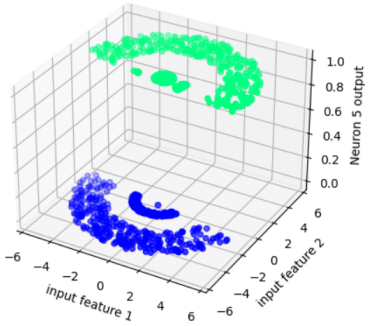
(b) Fig: neuron 2



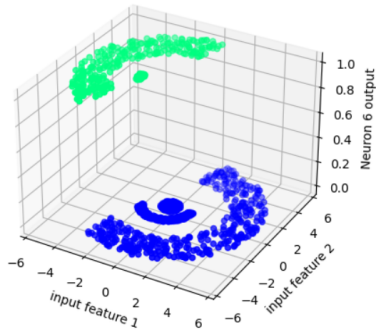
(c) Fig: neuron 3



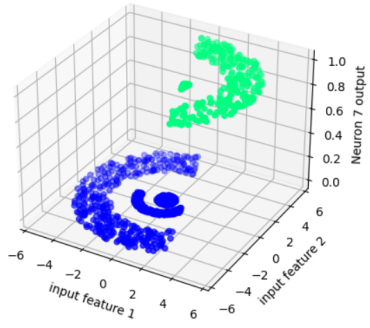
(d) Fig: neuron 4



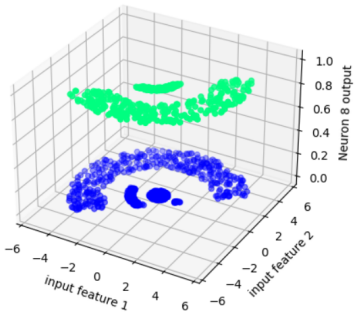
(e) Fig: neuron 5



(f) Fig: neuron 6

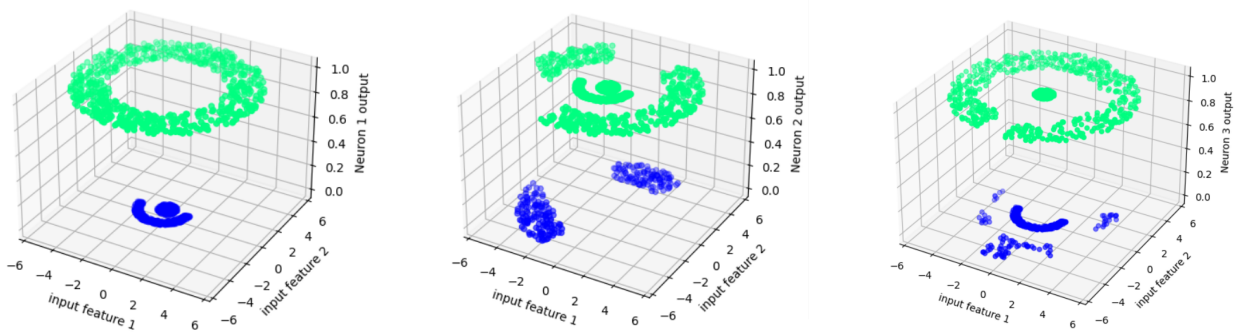


(g) Fig: neuron 7



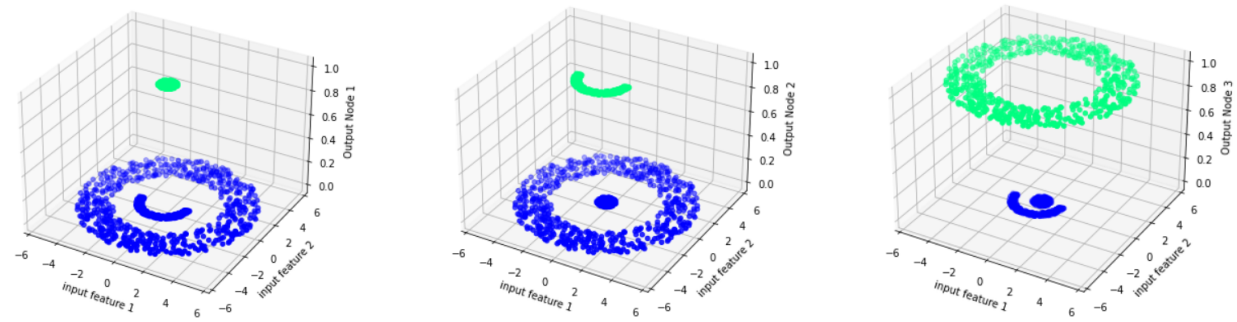
(h) Fig: neuron 8

Figure 2.11: Output of neurons in first hidden layer for training data



(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

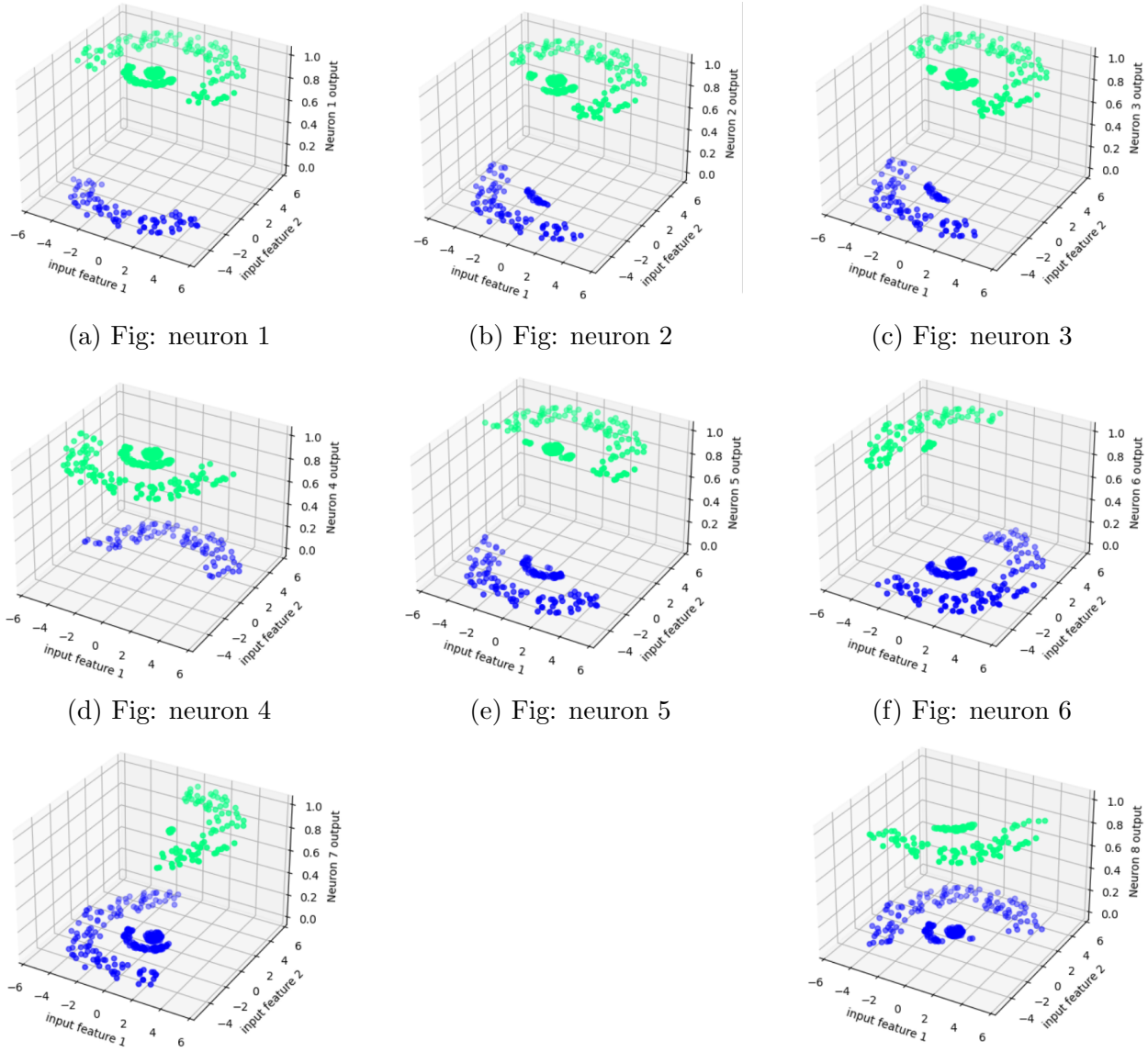
Figure 2.12: Output of neurons in second hidden layer for training data



(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

Figure 2.13: Output of neurons in output layer for training data

Output of neurons for first, second and output layer for the best architecture on validation dataset

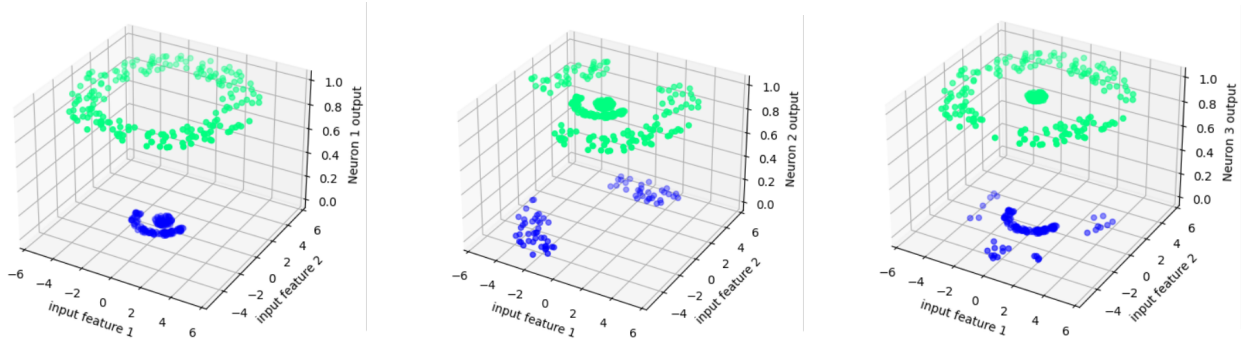


(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

(d) Fig: neuron 4 (e) Fig: neuron 5 (f) Fig: neuron 6

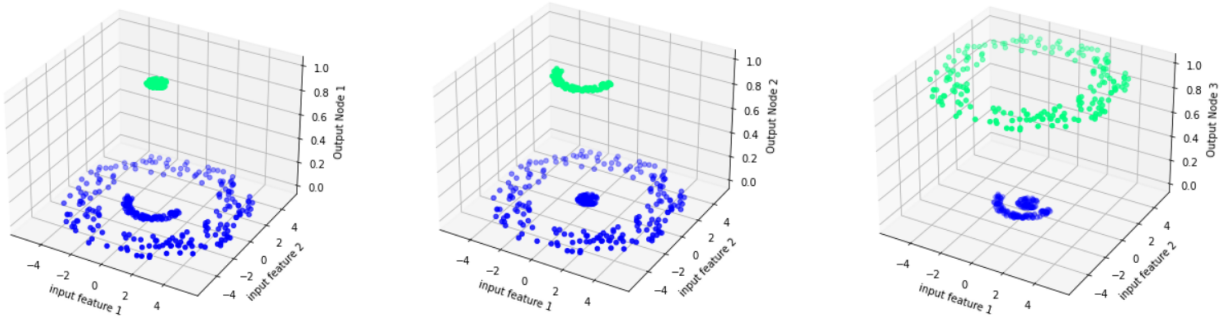
(g) Fig: neuron 7 (h) Fig: neuron 8

Figure 2.14: Output of neurons in first hidden layer for validation data



(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

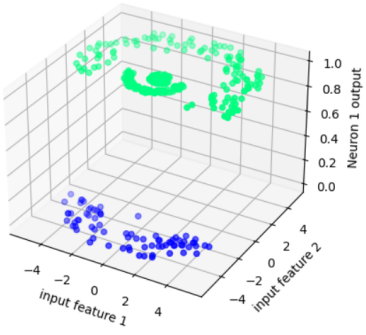
Figure 2.15: Output of neurons in second hidden layer for validation data



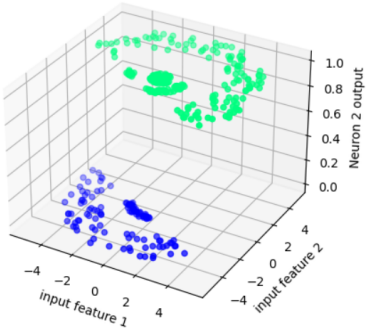
(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

Figure 2.16: Output of neurons in output layer for validation data

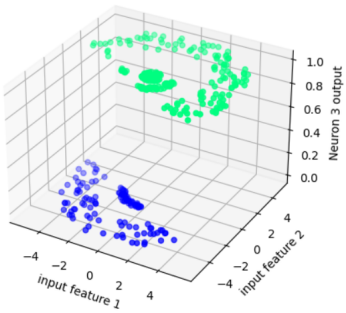
Output of neurons for first, second and output layer for the best architecture on testing dataset



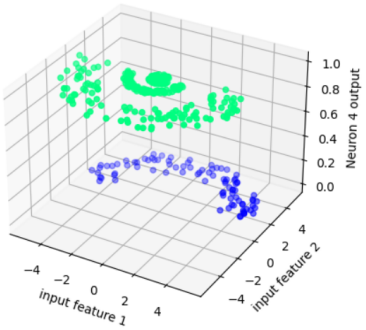
(a) Fig: neuron 1



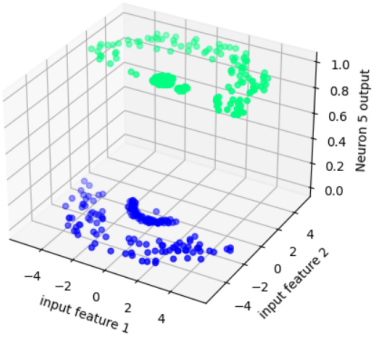
(b) Fig: neuron 2



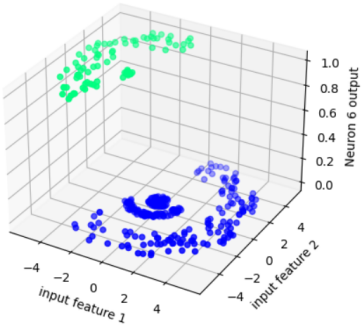
(c) Fig: neuron 3



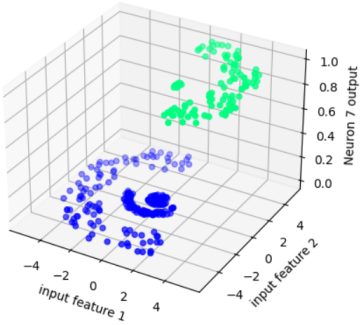
(d) Fig: neuron 4



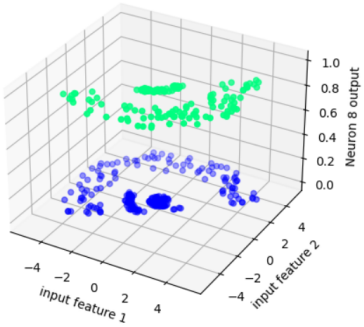
(e) Fig: neuron 5



(f) Fig: neuron 6

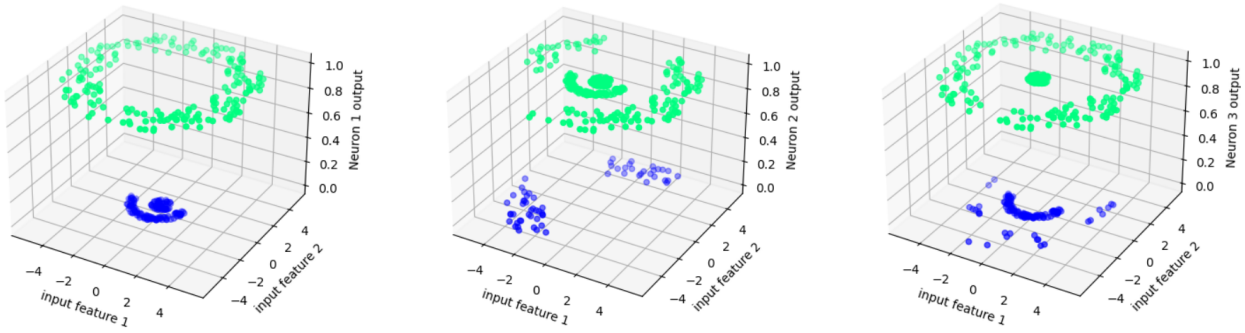


(g) Fig: neuron 7



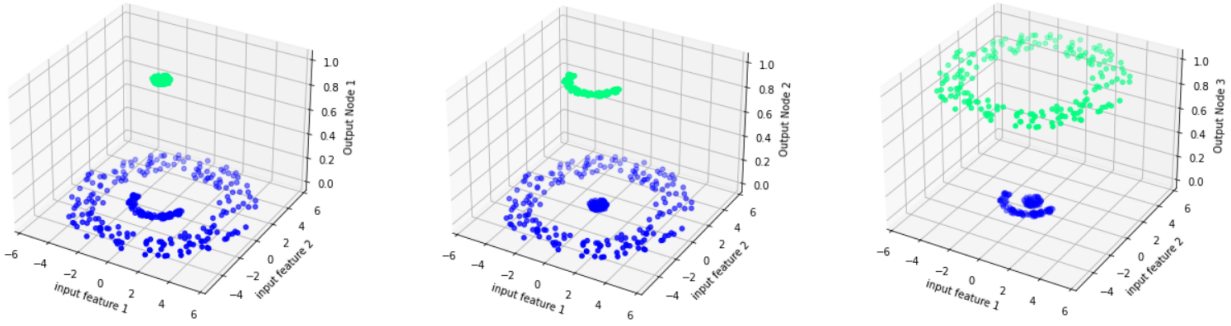
(h) Fig: neuron 8

Figure 2.17: Output of neurons in first hidden layer for testing data



(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

Figure 2.18: Output of neurons in second hidden layer for testing data



(a) Fig: neuron 1 (b) Fig: neuron 2 (c) Fig: neuron 3

Figure 2.19: Output of neurons in output layer for testing data

Fig 2.11,2.14,2.17 show the scatter plot of outputs for each of the hidden nodes in first layer for training, validation, and test data. It shows how each neuron uses the sigmoidal actiovation function to learn the shape of the data in small chucks

Fig 2.12,2.15,2.18 show the scatter plot of outputs for each of the hidden nodes in second layer for training, validation, and test data. It shows how each neuron uses the sigmoidal actiovation function to learn the shape of the data learned in 1st hidden layers and now tries to stitch it up together to learn it better.

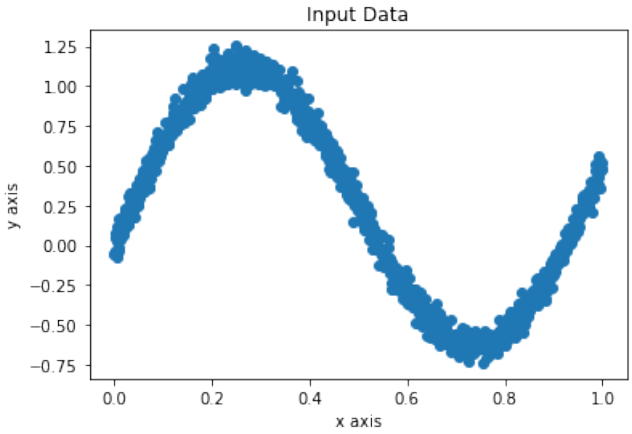
Fig 2.13,2.16,2.19 show the scatter plot of output neurons for the output layer for training, validation, and test data. It shows how each neuron learns the shape of the data correctly for each class.

Chapter 3

Regression Task

3.1 Univariate Data Results

The below plot show the input dataset for univariate case



The parameters used to train univariate data:
- $\eta = 0.05$
- epochs = As the dataset for univariate data is small(around 1000) lesser number of epochs are required to converge as seen in Fig 3.2. This is the stopping criteria used here(with reference to best architecture)

Different Model Architectures

The dataset was tested on different architectures and the results are shown below based on the validation dataset provided.

No. of Nodes	Training	Validation	Testing
1	0.08889933	0.08889933	0.08889933
2	0.08529873	0.08663012	0.08529873
4	0.01171854	0.01173387	0.01059213

Table 3.1: Mean squared error (MSE) on training data, validation data and test data for different number of nodes

From Table 3.1 if we take more no. of hidden layers then model will give less mean square error **BEST ARCHITECTURE**

The nodes in hidden layer have been experimented and the best result was observed for 4 nodes in hidden layer.
The following plots are for the best architecture.

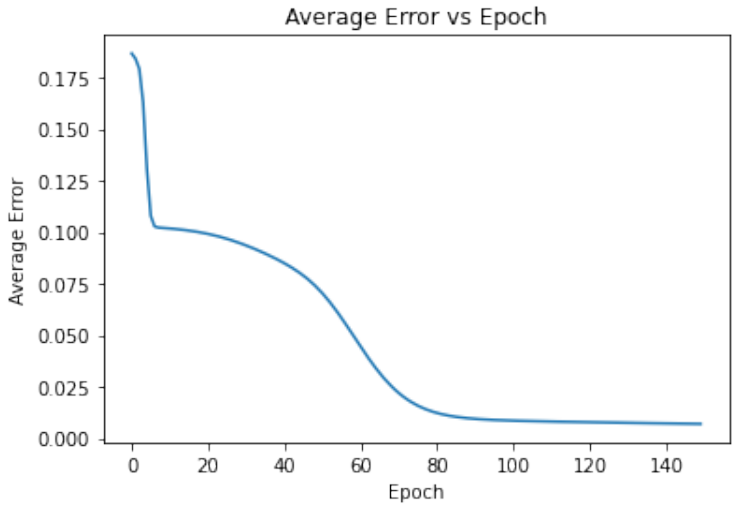
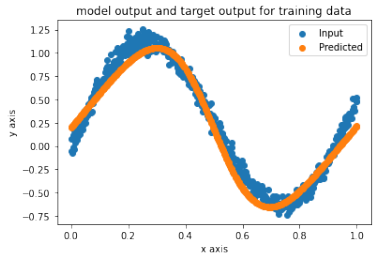
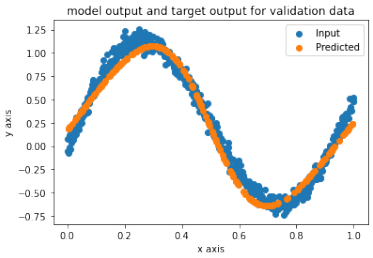


Figure 3.1: Plot of average error (y-axis) vs epochs (x-axis)



(a) Fig: model output and target output for training data

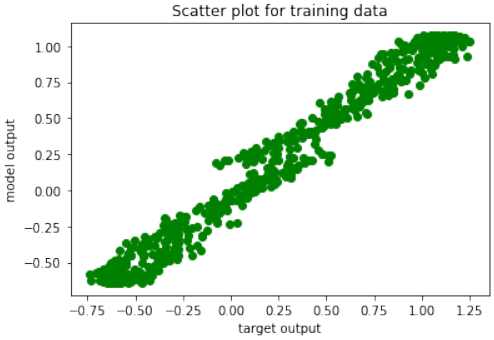
(b) Fig: model output and target output for testing data



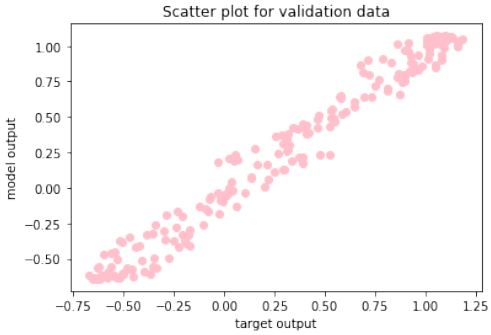
(c) Fig: model output and target output for validation data

Figure 3.2: Plots of model output and target output for training data, validation data and test data

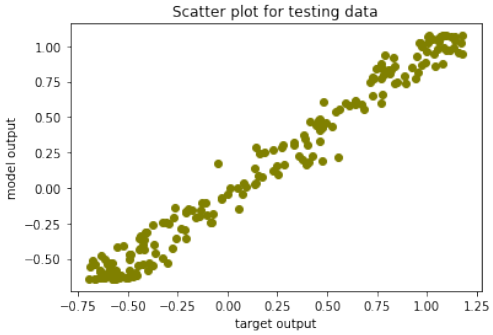
In Fig 3.2 it shows that dues to FCNN the model is easily able to approximate a nonlinear boundary for the nonlinear dataset



(a) Fig: Scatter plot for training data



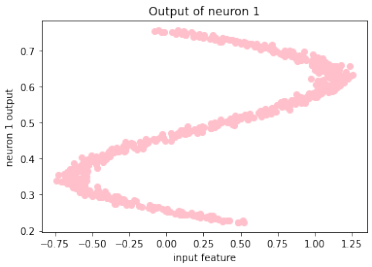
(b) Fig:Scatter plot for validation data



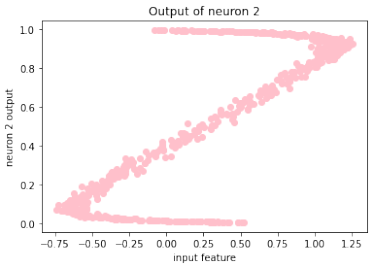
(c) Fig: Scatter plot for testing data

Figure 3.3: Scatter plot with target output on x-axis and model output on y-axis, for training data, validation data and test data

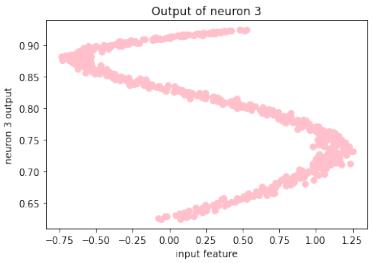
In Fig 3.3 above as target output and model output almost from a line replication y = x. It shows that y_pred and y_out give the same output on the input features which proves that our model is trained well



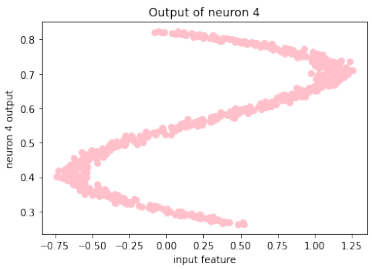
(a) Fig: neuron 1



(b) Fig: neuron 2

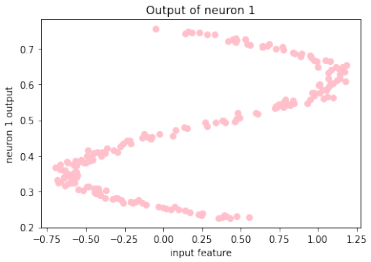


(c) Fig: neuron 3

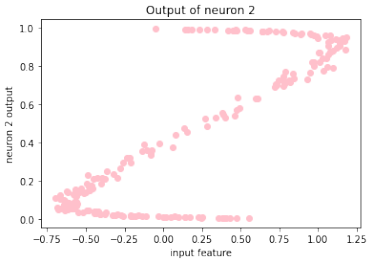


(d) Fig: neuron 4

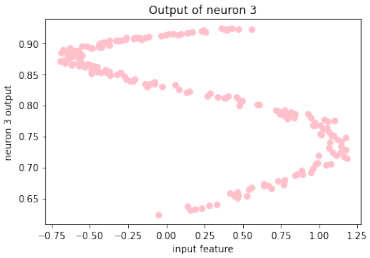
Figure 3.4: Output of neurons in first hidden layer for training data



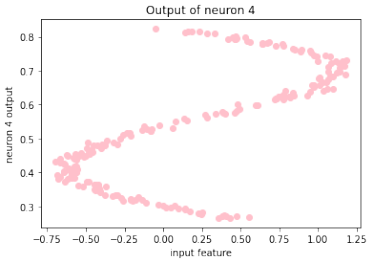
(a) Fig: neuron 1



(b) Fig: neuron 2

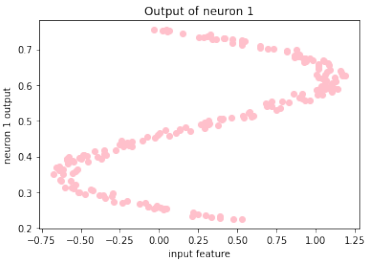


(c) Fig: neuron 3

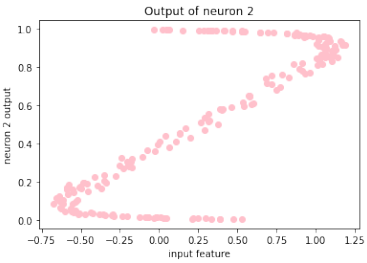


(d) Fig: neuron 4

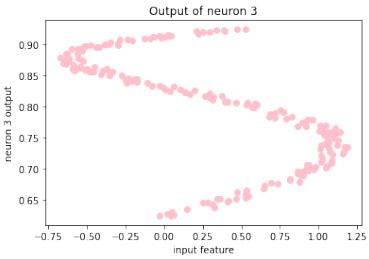
Figure 3.5: Output of neurons in first hidden layer for test data



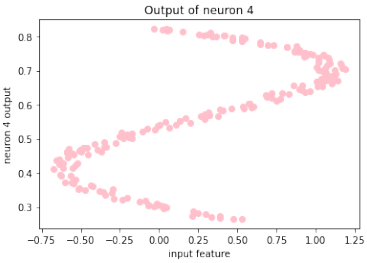
(a) Fig: neuron 1



(b) Fig: neuron 2



(c) Fig: neuron 3



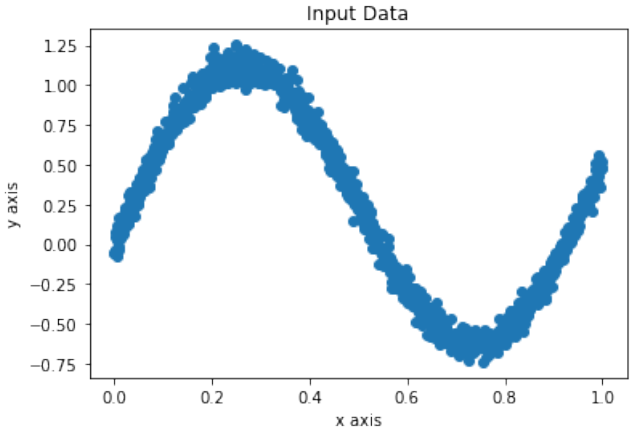
(d) Fig: neuron 4

Figure 3.6: Output of neurons in first hidden layer for validation data

Fig 3.4,3.5,3.6 show the scatter plot Plots of outputs for each of the hidden nodes in hidden layer for training, validation, and test data. It shows how each neuron learns the shape of the data

3.2 Bivariate Data Results

The below plot show the input dataset for univariate case



The parameters used to train Bivariate data:
- $\eta = 0.01$
- epochs = 100 As the dataset for non linear data required training to form a proper boundary and also the model converges after only close to 90 epochs. This is the stopping criteria used here as seen in Fig 3.7(based on the best architecture)

Different Model Architectures

The dataset was tested on different architectures and the results are shown below based on the validation dataset provided.

No. of Nodes in h1	No. of Nodes in h2	Training	Validation	Testing
2	2	0.53198565	0.51247319	0.52537669
3	3	0.52088967	0.50636466	0.51637709
5	5	0.01158498	0.01091931	0.01102444

Table 3.2: Mean squared error (MSE) on training data, validation data and test data for different number of nodes in hidden layer 1 and 2

From Table 3.2 if we take more no. of inner hidden layers then the outer hidden layers will give less mean square error

BEST ARCHITECTURE

The nodes in hidden layer 1 and 2 have been experimented and the best result was observed for 5 nodes in hidden layer 1 and 5 nodes in hidden layer 2
Below all the figures are given in reference to the best architecture

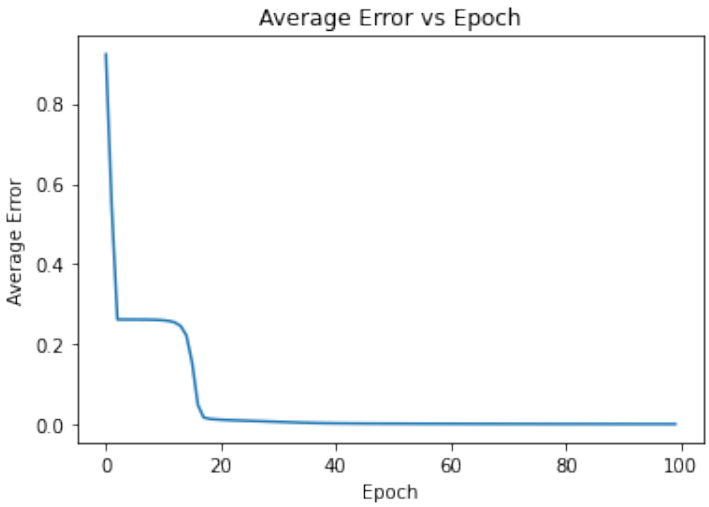
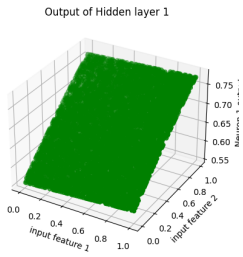
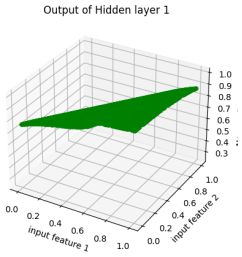


Figure 3.7: Plot of average error (y-axis) vs epochs (x-axis)

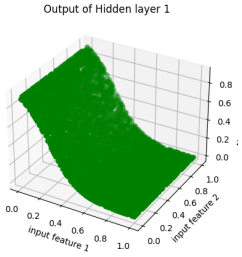
Output of neurons for first and second hidden layer for the best architecture on training dataset



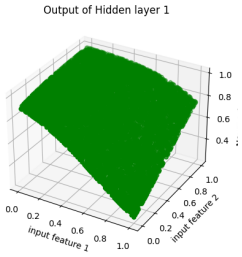
(a) Fig: neuron 1



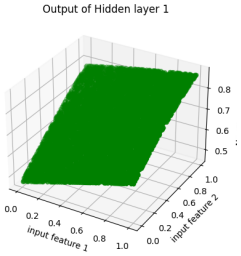
(b) Fig: neuron 2



(c) Fig: neuron 3

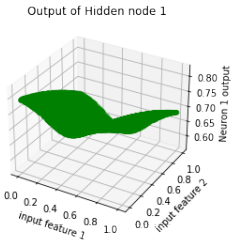


(d) Fig: neuron 4

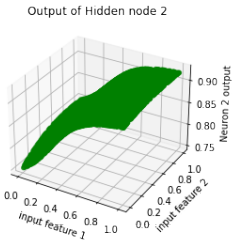


(e) Fig: neuron 5

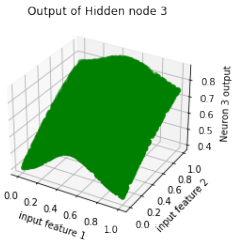
Figure 3.8: Output of neurons in first hidden layer for training data



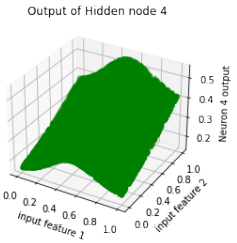
(a) Fig: neuron 1



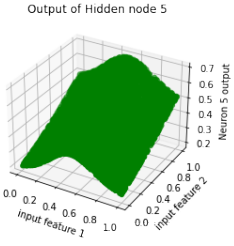
(b) Fig: neuron 2



(c) Fig: neuron 3



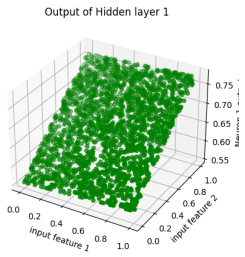
(d) Fig: neuron 4



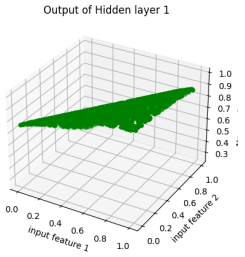
(e) Fig: neuron 5

Figure 3.9: Output of neurons in second hidden layer for training data

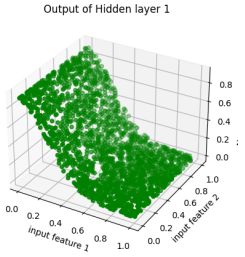
Output of neurons for first and second layer for the best architecture on testing dataset



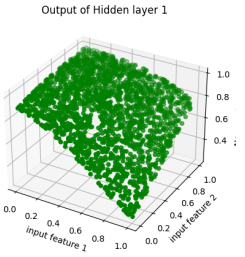
(a) Fig: neuron 1



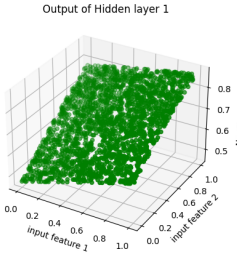
(b) Fig: neuron 2



(c) Fig: neuron 3

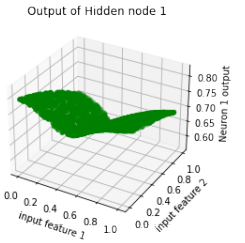


(d) Fig: neuron 4

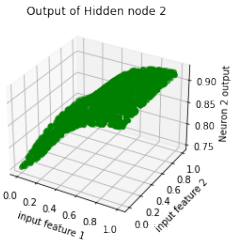


(e) Fig: neuron 5

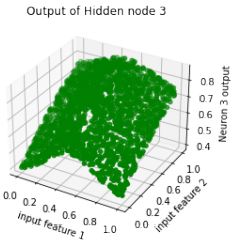
Figure 3.10: Output of neurons in first hidden layer for testing data



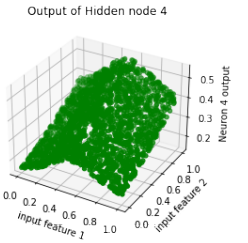
(a) Fig: neuron 1



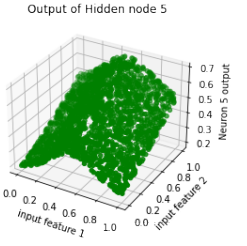
(b) Fig: neuron 2



(c) Fig: neuron 3



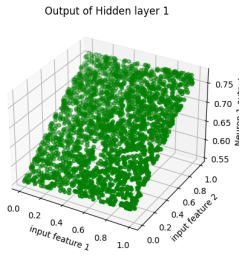
(d) Fig: neuron 4



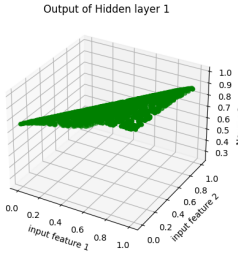
(e) Fig: neuron 5

Figure 3.11: Output of neurons in second hidden layer for test data

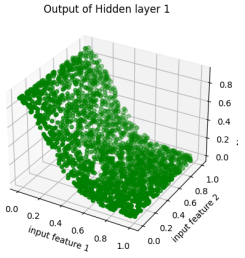
Output of neurons for first, second and output layer for the best architecture on validation dataset



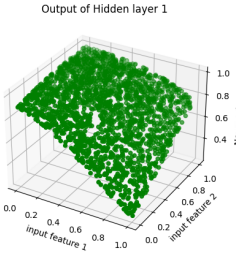
(a) Fig: neuron 1



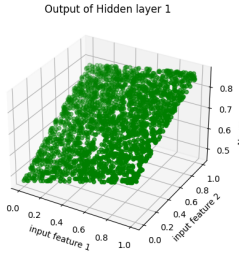
(b) Fig: neuron 2



(c) Fig: neuron 3

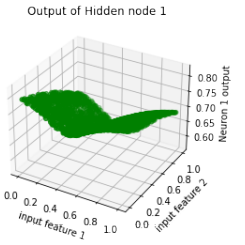


(d) Fig: neuron 4

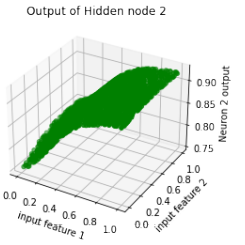


(e) Fig: neuron 5

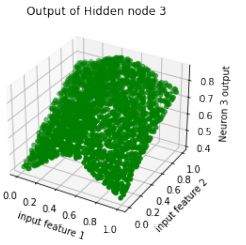
Figure 3.12: Output of neurons in first hidden layer for validation data



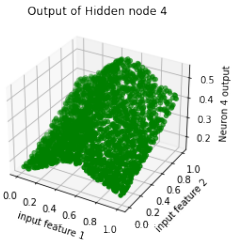
(a) Fig: neuron 1



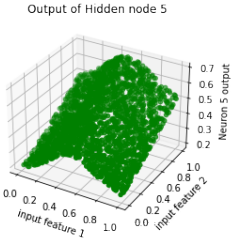
(b) Fig: neuron 2



(c) Fig: neuron 3



(d) Fig: neuron 4



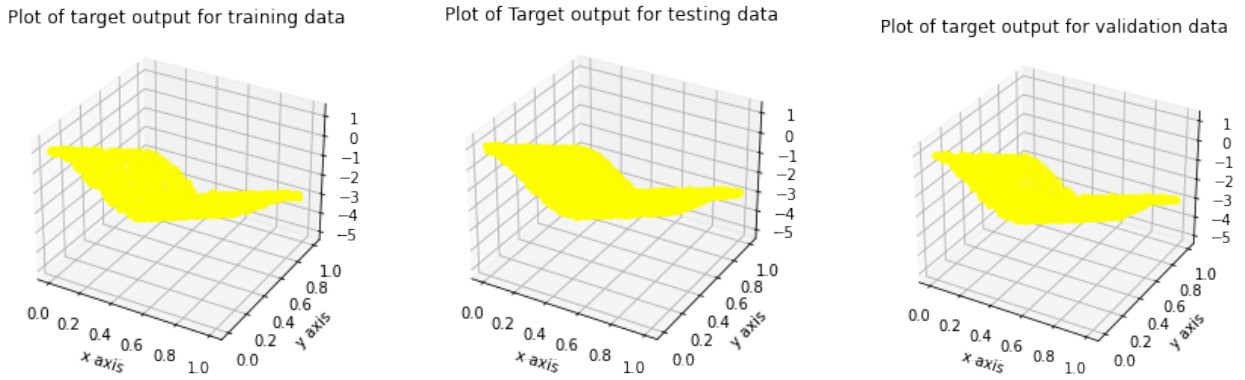
(e) Fig: neuron 5

Figure 3.13: Output of neurons in second hidden layer for validation data

Fig 3.8 to Fig 3.13 show the outputs for each of the hidden nodes in first and second layer for training, validation, and test data. It shows how each neuron learns the shape of the data in parts in first hidden layer and then

tries to stitch back the learned shape in the second layer

Plots for model and target output for training, testing and validation dataset

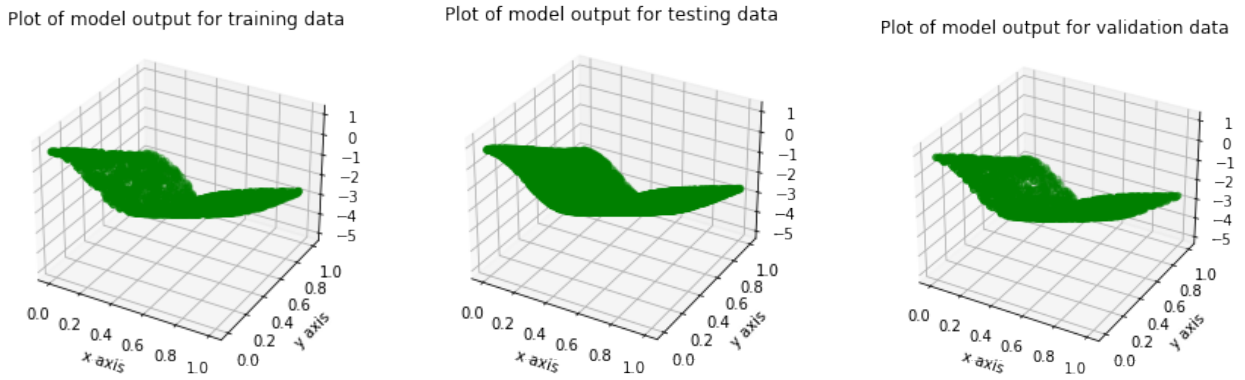


(a) Fig: target output for training data

(b) Fig: target output for testing data

(c) Fig: target output for validation data

Figure 3.14: Plots of target output for training data, validation data and test data



(a) Fig: model output for training data

(b) Fig: model output for testing data

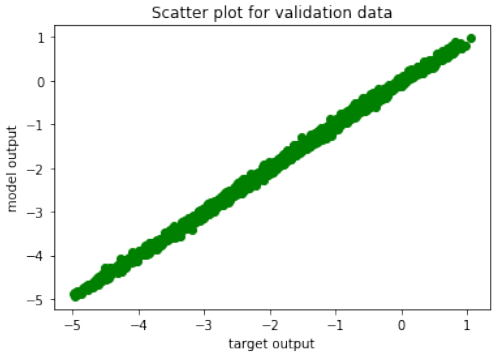
(c) Fig: model output for validation data

Figure 3.15: Plots of model output for training data, validation data and test data

In Fig 3.15 it shows that due to FCNN the model is easily able to approximate a nonlinear boundary for the nonlinear dataset

Scatter Plots for model and target output for training, testing and validation dataset

(a) Fig: Scatter plot for training data



(b) Fig:Scatter plot for validation data

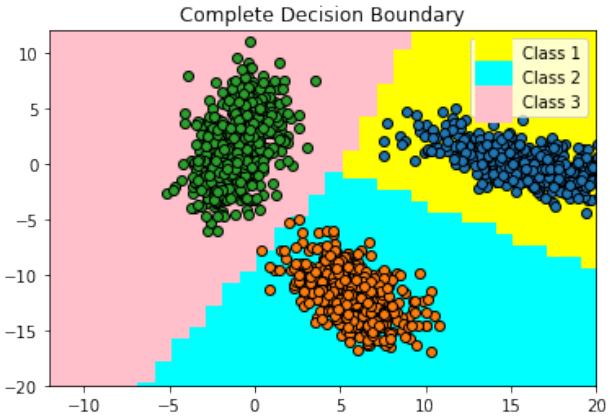
(c) Fig: Scatter plot for testing data

Figure 3.16: Scatter plot with target output on x-axis and model output on y-axis, for training data, validation data and test data

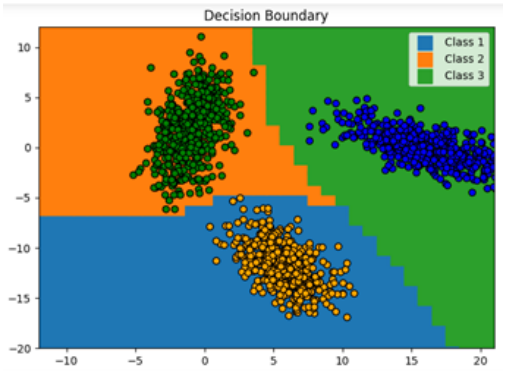
In Fig 3.9 above as target output and model output almost from a line replicating y = x. It shows that y_pred and y_out give the same output on the input features which proves that our model is trained well

Chapter 4

Comparison with single neuron model



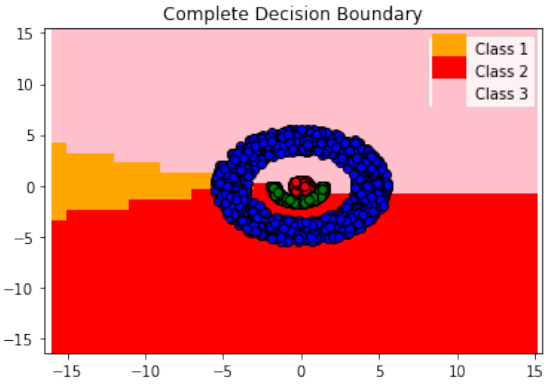
(a) Fig: Classification using single perceptron



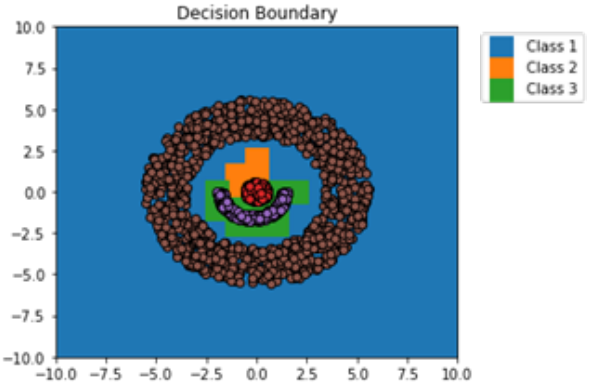
(b) Fig: Classification using single hidden layer

Figure 4.1: Classification for Linearly separable data

From fig 4.1 it can be seen that single neuron model is enough to classify the 3 classes , therefore using the hidden layer will only increase complexity.



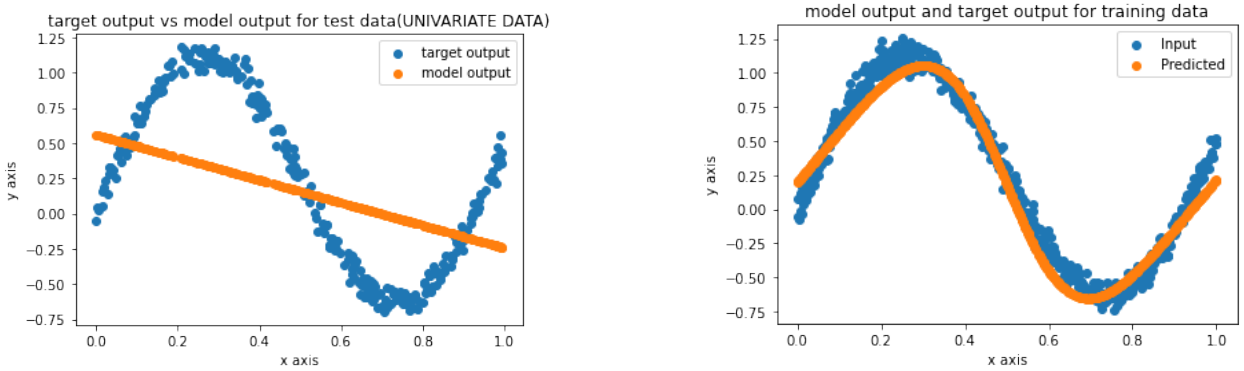
(a) Fig: Classification using single perceptron



(b) Fig: Classification using double hidden layer

Figure 4.2: Classification for Non Linearly separable data

From fig 4.2, it can be seen that the single neuron model is not enough for classifying the non linearly separable data.However, when multiple neurons with 2 hidden layers are used then the boundary formed is non-linear which classifies the non-linearly separable data.This boundary is formed by stitching the output of each of the neuron from a particular layer.

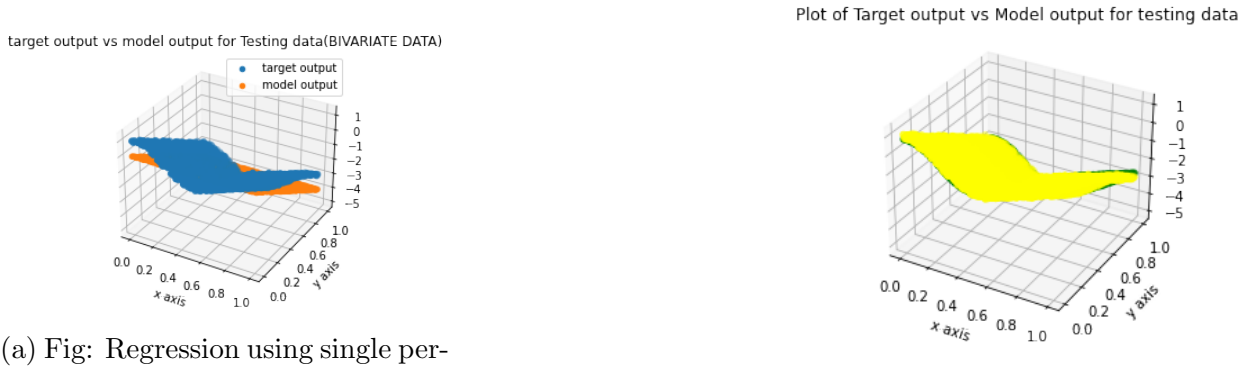


(a) Fig: Regression using single perceptron

(b) Fig: Regression using FCNN

Figure 4.3: Regression for Univariate data

From fig4.3, it can be observed that a single neuron is not enough to perform regression.The output of a single neuron model is a line which is unable to fit the data.However,on using multiple neurons in a single layer,the model fits on the data and is able to perform regression on it.



(a) Fig: Regression using single perceptron

(b) Fig: Regression using FCNN

Figure 4.4: Regression for Bivariate data

From fig 4.4,it can be observed that the single neuron model cannot perform regression .However, on using 2 hidden layers the model output fits on the actual data. Hence,it is able to perform regression on bivariate data.

Chapter 5

References

[1] https://www.kdnuggets.com/2019/11/designing-neural-networks.html

[2] Lecture notes of CS671 Feb 2023-24 by Dr.Dileep A D