CS671: Deep Learning and Applications
ASSIGNMENT 2 REPORT

Multilayer Feedforward Neural Network

Ankit Mehra(S22020)
Urvashi Goswami(S22024)
Ashish Rana(S22019)

March 7, 2023

Introduction

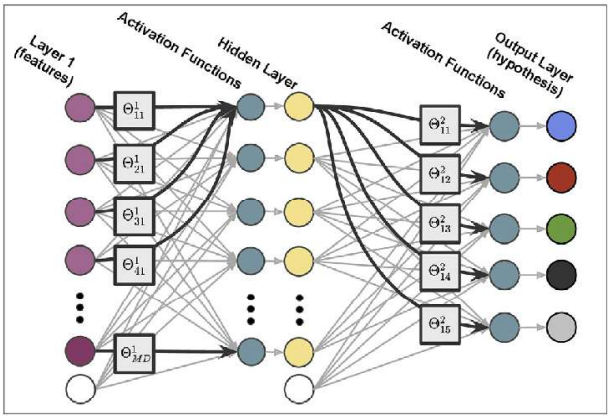


Fig 1. Image Source : www.researchgate.net

FCNN is the purest form of ANN in which input and data travel in only one direction. Data flows in an only forward direction; that's why it is known as the Feedforward Neural Network. The data passes through input nodes and exit from the output nodes. The nodes are not connected cyclically. It doesn't need to have a hidden layer. In FNN, there doesn't need to be multiple layers. It may have a single layer also.

It comprises of :

- Fully Connected Neural Network
- Auto Encoders
- Convolutional Neural Network

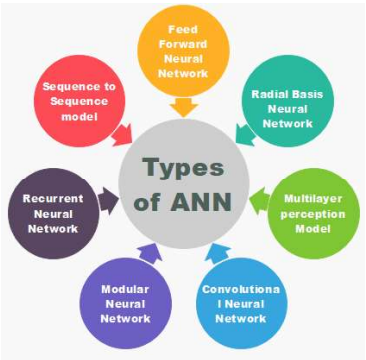


Fig 2. Image Source : www.javapoint.com

A FCNN has three or more layer. The data that cannot be separated linearly is classified with the help of this network. This network is a fully connected network that means every single node is connected with all other nodes that are in the next layer. A Nonlinear Activation Function is used in Multilayer Perceptron. It's input and output layer nodes are connected as a directed graph. It is a deep learning method so that for training the network it uses backpropagation.

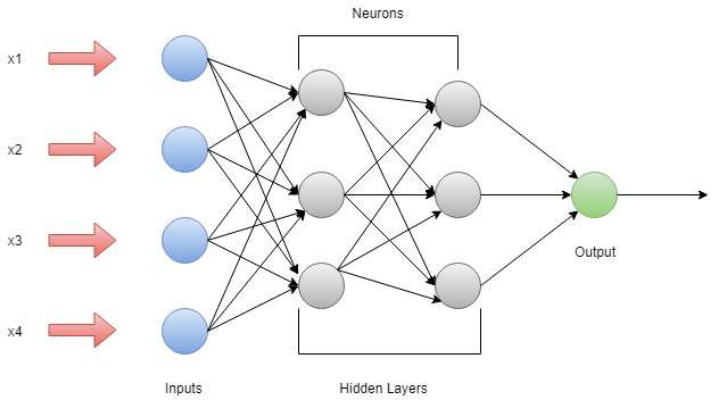


Fig 3. Image Source : www.javapoint.com

Forward propagation

The output of one layer is the input of the next one.

Fig 4. Image Source : www.javapoint.com

This is called forward propagation. Essentially, we give the input data to the first layer, then the output of every layer becomes the input of the next layer until we reach the end of the network. By comparing the result of the network (Y) with the desired output (let's say Y*), we can calculate an error E. The goal is to minimize that error by changing the parameters in the network.

Backward propagation

$$\frac{\partial E}{\partial X} \leftarrow \boxed{\text{layer}} \leftarrow \frac{\partial E}{\partial Y}$$

Fig 5. Image Source : www.towardsdatascience.com

When for a layer the derivative of the error with respect to its output ∂E / ∂Y, then we can calculate derivative of the error with respect to its input ∂E / ∂X.

$$\frac{\partial E}{\partial X} = \begin{bmatrix} \frac{\partial E}{\partial x_1} & \frac{\partial E}{\partial x_2} & \cdots & \frac{\partial E}{\partial x_i} \end{bmatrix}$$
$$\frac{\partial E}{\partial Y} = \begin{bmatrix} \frac{\partial E}{\partial y_1} & \frac{\partial E}{\partial y_2} & \cdots & \frac{\partial E}{\partial y_j} \end{bmatrix}$$

Where, E is a scalar (a number) and X and Y are matrices.

$$\frac{\partial E}{\partial w} = \sum_j \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial w}$$

Using chain rule,

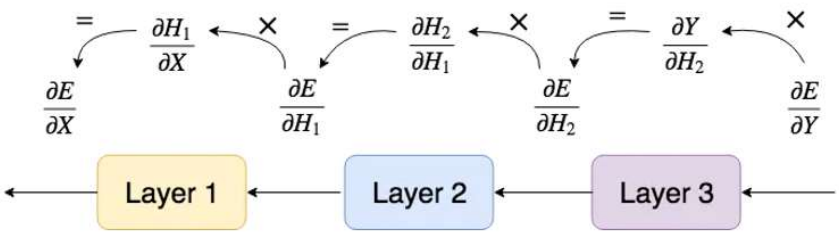


Fig 6. Image Source : towardsdatascience.com

The unknown is ∂y_j / ∂w which totally depends on how the layer is computing its output. So if every layer have access to ∂E / ∂Y, where Y is its own output, then we can update the parameters.

Layer 3 is going to update its parameters using ∂E / ∂Y, and is then going to pass ∂E / ∂H_2 to the previous layer, which is its own "∂E / ∂Y". Layer 2 is then going to do the same, and so on and so forth.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is an optimization algorithm used in deep learning to update the weights of a neural network during the training process. It is a variant of gradient descent that uses a small random subset of training data to compute the gradient at each step, rather than using the entire dataset.

The basic idea of SGD is to iteratively update the weights in the direction of the negative gradient of the loss function, with the step size determined by a learning rate hyperparameter. If the learning rate is too high, the algorithm may overshoot the minimum of the cost function and fail to converge. If the learning rate is too low, the algorithm may converge too slowly.

The update rule for a weight parameter w at iteration t is given by:

$$w_t + 1 = w_t - \alpha * \Delta L(w_t, x_i, y_i)$$

where L is the loss function, is the learning rate, L is the gradient of the loss function with respect to the weights, and (x_i, y_i) is a randomly selected mini-batch of training data.

The use of mini-batches in SGD allows for faster convergence to a local minimum of the loss function compared to using the full training set, as the computation of the gradient can be performed in parallel. However, it also introduces additional noise into the optimization process due to the random sampling of data.

The main steps of the stochastic gradient descent algorithm for deep learning are as follows:

1. Initialize the weights of the neural network randomly.
2. Divide the training data into small batches of random samples.
3. For each batch of samples:
   (a) Forward propagate the input through the neural network to get the output
   (b) Compute the loss function between the predicted output and the actual output.
   (c) Compute the gradient of the loss function with respect to the weights using backpropagation.
   (d) Update the weights by subtracting a small fraction of the gradient from the current weights. This fraction is called the learning rate and is typically a small positive value.

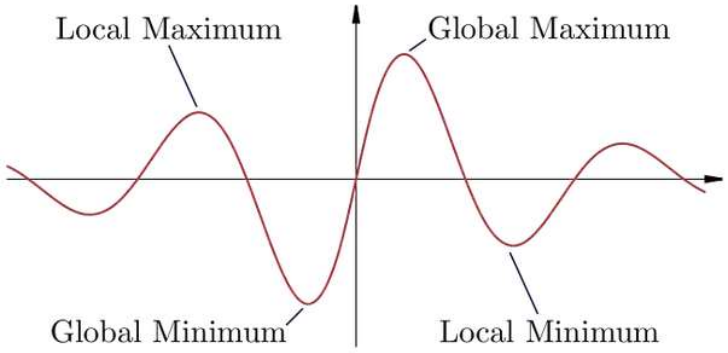


Fig 7. Image Source : Deepai.org

Number of datapoints used to perform different pattern analysis tasks.

TASKS		DATA POINTS		
		TRAINING	VALIDATION	TESTING
CLASSIFICATION-LINEARLY SEPARABLE DATA	CLASS 1	300	100	100
	CLASS 2	300	100	100
	CLASS 3	300	100	100
CLASSIFICATION-NON LINEARLY SEPARABLE DATA	CLASS 1	781	260	262
	CLASS 2	782	260	262
REGRESSION UNIVARIATE DATA		600	200	201
REGRESSION BIVARIATE DATA		6120	2040	2041

Table 1. Dataset Stats

Classification Task

Case 1 :- Linearly Separable Data

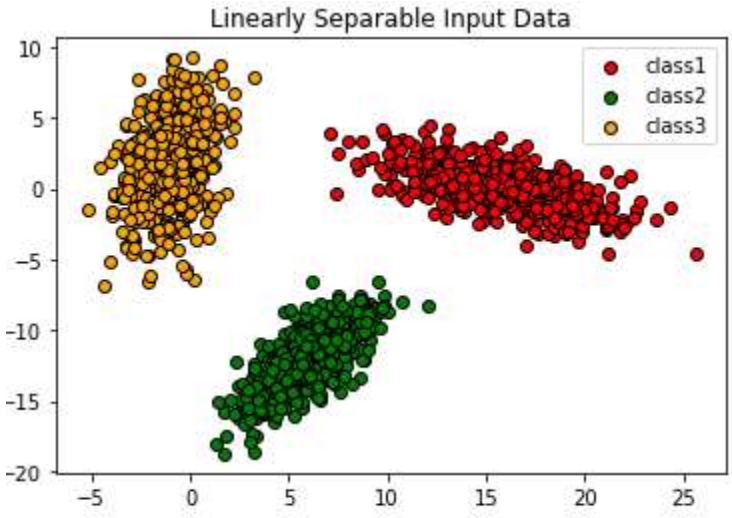


Fig.8 Linearly Separable Data Plot

- Dataset has 500 points in each class.Data is Split into Training set, Validation set and testing set in the ratio of 3:1:1
- Various models with different numbers of nodes in hidden layer and different numbers of epochs are tested and below is the comparison between a few of them.

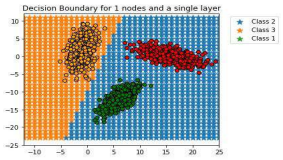
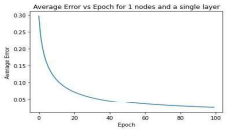
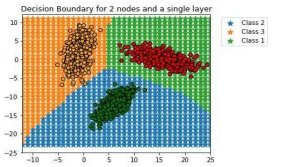
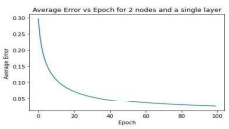
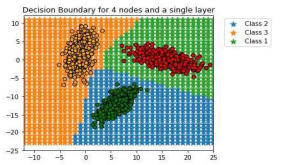
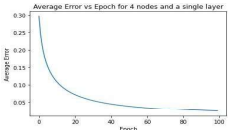
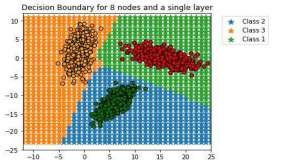
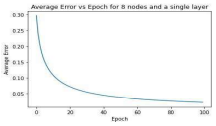
No. of nodes	Decision Boundary	Error v/s Epochs	Confusion Matrix	Accuracy
1			[[0 100 0] [0 100 0] [0 0 100]]	66.6%
2			[[100 0 0] [0 100 0] [0 0 100]]	100%
4			[[100 0 0] [0 100 0] [0 0 100]]	100%
8			[[100 0 0] [0 100 0] [0 0 100]]	100%

Table 2. Different architectures for linearly separable data

Below are the 4 Models that we tested for the optimum result.

Model 1	**Model 2**
• Number of Epochs =100 • Learning Rate=0.01 • Number of nodes in hidden layer =1	• Number of Epochs =100 • Learning Rate=0.01 • Number of nodes in hidden layer =2
Model 3	**Model 4**
• Number of Epochs =100 • Learning Rate=0.01 • Number of nodes in hidden layer =4	• Number of Epochs =100 • Learning Rate=0.01 • Number of nodes in hidden layer =8

Table 3. Model architecture

- **We got the best decision boundary only after increasing the nodes to 8 in a Single hidden layer after 100 Epochs.**

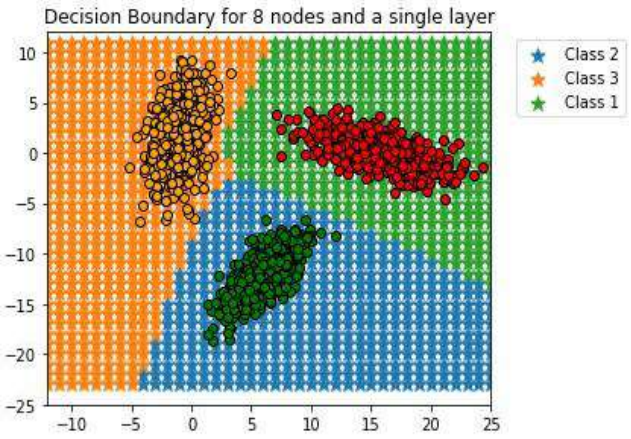


Fig 9. Decision Boundary plot

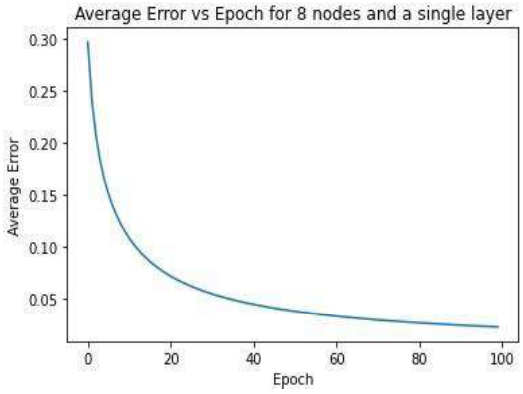


Fig 10.Error v/s Epoch Plot

Confusion Matrix

		Actual		
		Class1	*Class2*	*Class3*
Predicted	*Class1*	100	0	0
	Class2	0	100	0
	Class3	0	0	100

Table 4. Confusion Matrix

- For 1 hidden layer and 8 nodes we got 100% accuracy and the best decision boundary.

Plots of outputs for each of the hidden nodes and output nodes in FCNN

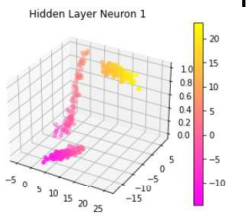
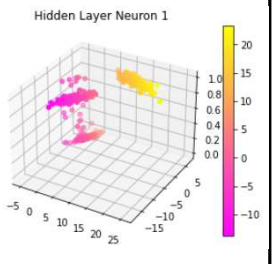
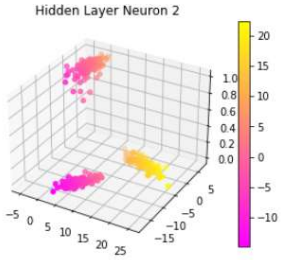
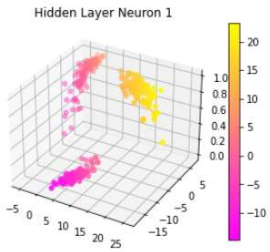
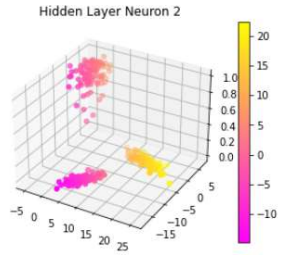
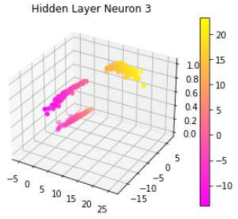
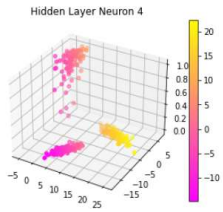
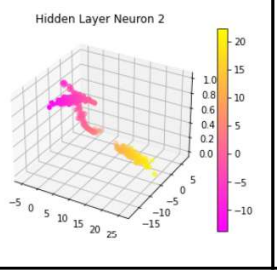
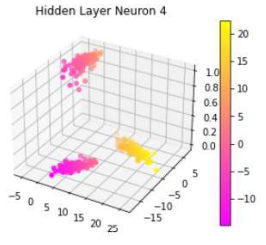
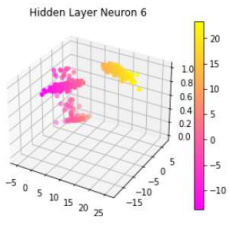
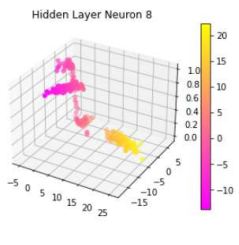
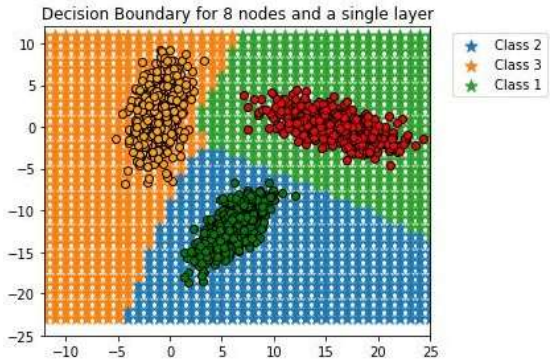
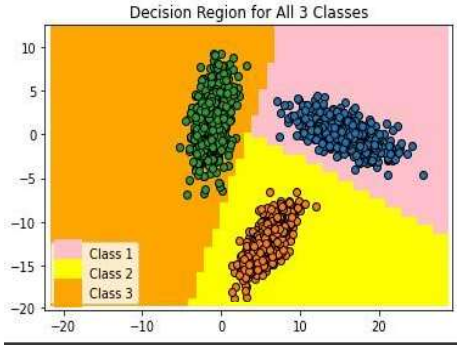
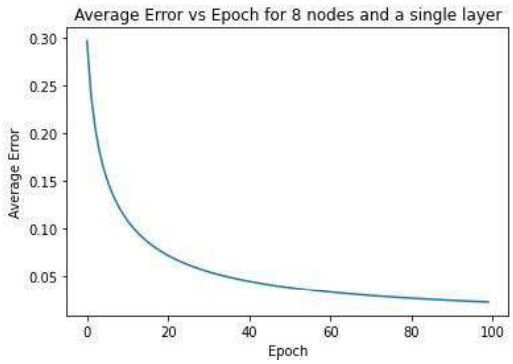
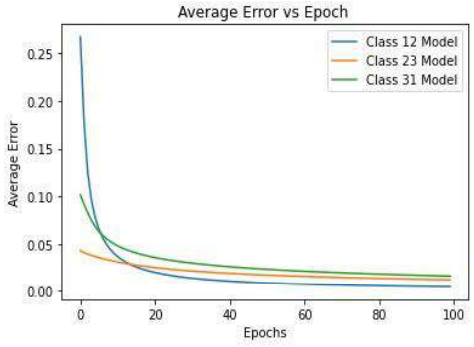
Model	Neuron 1			
Model 1				
	Neuron 1	**Neuron 2**		
Model 2				
	Neuron 1	**Neuron 2**	**Neuron 3**	**Neuron 4**
Model 3				
	Neuron 2	**Neuron 4**	**Neuron 6**	**Neuron 8**
Model 4				

Table 5.Contribution of each neuron in the output.

It can be observed that each neuron learns a different pattern and finally stitches them all together to give us the final output.

Comparison of FCNN and Single perceptron for classification task for linearly separable data.

Criterion	FCNN	Single Perceptron
Decision Boundary		
Avg Error V/s Epochs		

Confusion Matrix	`[[100  0  0]`
	`[ 0 100  0]`
	`[ 0  0 100]]`
		`[[300  0  0]`
		`[ 0 300  0]`
		`[ 6  2 292]]`

Accuracy	**100%**	**99.1%**

Table 6. Comparison between FCNN and Perceptron models

By looking at the above comparison we can infer that though classification for linearly separable classes can be done using a single perceptron model but using a hidden layer and multiple nodes gives us better decision boundaries and better accuracy .

Case2:- Non-linearly Separable Data

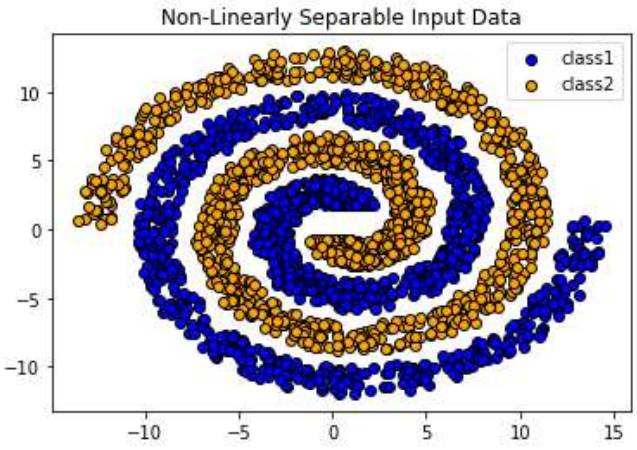


Fig 11. Non-linearly separable data plot

- Dataset has 1303 points in each class.Data is split into Training set, Validation set and Testing set in the ratio of 3:1:1 ,with 781 points in the training data set and 260 points in each validation and test data set.
- Various models with different numbers of nodes in hidden layers and different numbers of epochs are tested and below is the comparison between some of them.

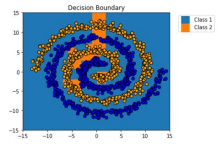
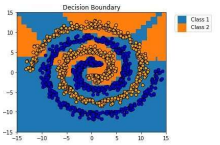
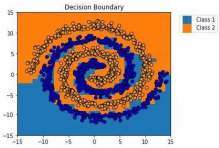
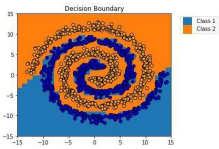
No. of nodes in 1st layer	No. of nodes in 2nd layer	Decision Boundary	Number of Epochs	Confusion Matrix	Learning Rate	Accuracy
8	8		1000	Confusion Matrix [[257 4] [241 20]]	0.05	53%
15	12		1500	Confusion Matrix [[144 117] [43 218]]	0.01	73.6%
18	10		5000	Confusion Matrix [[198 63] [22 239]]	0.001	88%
20	10		8000	Confusion Matrix [[254 7] [4 257]]	0.001	98%

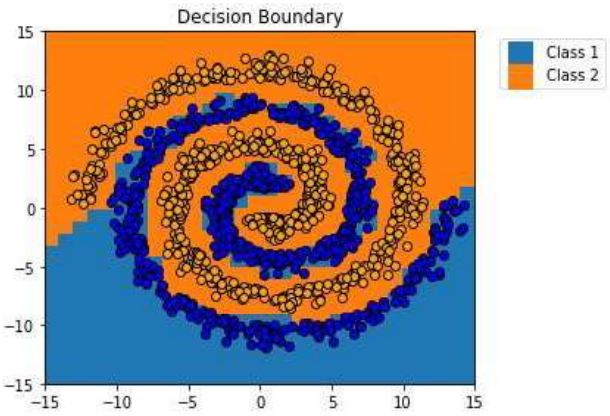
Table 7.Different architectures for non-linearly separable data

- Out of all the models we have tested, below are 4 Models for the optimum result.

Model 1	**Model 2**
• Number of Epochs =1000 • Learning Rate=0.05 • Number of nodes in layer 1 =8 • Number of nodes in layer 2 =8	• Number of Epochs =1500 • Learning Rate=0.01 • Number of nodes in layer 1 =18 • Number of nodes in layer 2 =12
Model 3	**Model 4**
• Number of Epochs =5000 • Learning Rate=0.001 • Number of nodes in layer 1 =18 • Number of nodes in layer 2 =10	• Number of Epochs =8000 • Learning Rate=0.001 • Number of nodes in layer 1 =20 • Number of nodes in layer 1 =10

Table 8.Model architecture

- **We got the best decision boundary only after increasing the nodes to 20 in the 1st hidden layer and 10 in the 2nd hidden layer . Accuracy of the best model is 98%.**



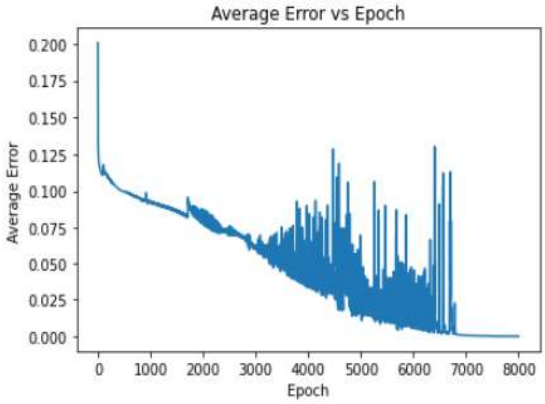


Fig 12.Decision boundary plot

Fig 13.Error v/s Epochs plot

Confusion Matrix

		Actual	
		Class1	*Class2*
Predicted	*Class1*	254	7
	Class2	4	257

Table 9.Confusion matrix

It can be inferred that 2 layers is sufficient to classify any non-linearly separable data

Plots of outputs for each of the hidden nodes and output nodes in FCNN

Model 1: 8 nodes in the first hidden layer and 8 nodes in the second hidden layer.

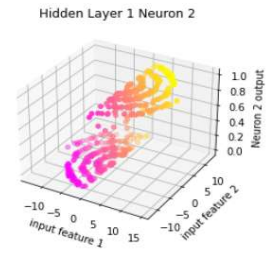
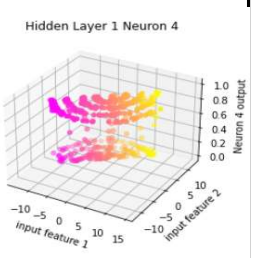
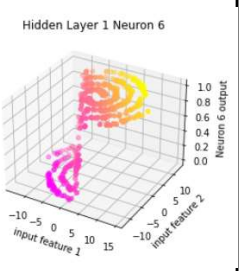
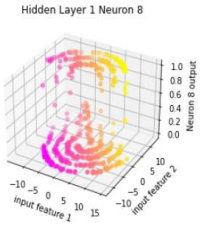
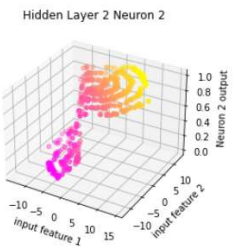
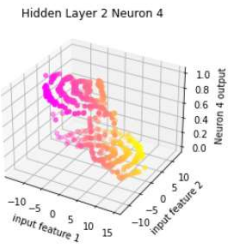
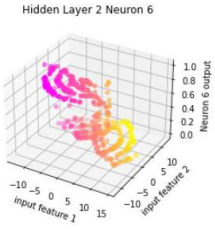
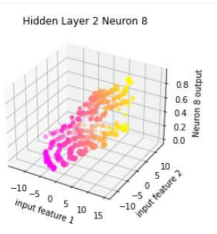
Model 1	Neuron 2	Neuron 4	Neuron 6	Neuron 8
Hidden Layer 1				
Hidden Layer 2				

Table 10.Contribution of each neuron in the output for model 1.

Model 4 (Best Model) : 20 nodes in the first hidden layer and 10 nodes in second hidden layer.

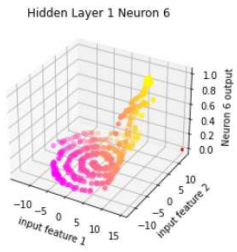
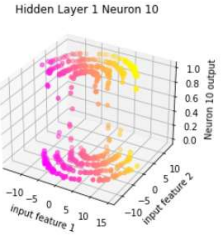
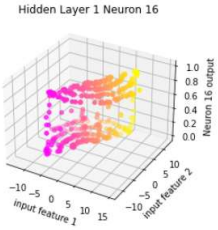
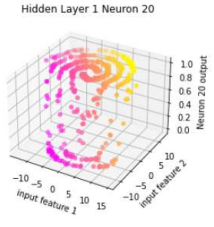
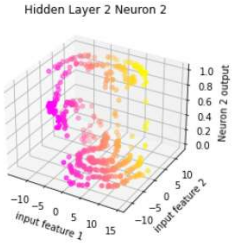
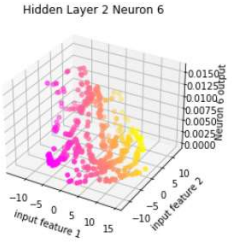
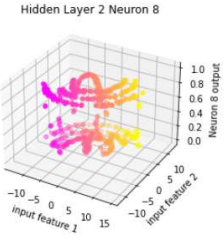
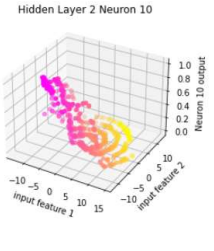
Model 4	Neuron 6	Neuron 10	Neuron 16	Neuron 20
Hidden Layer 1				
	Neuron 2	**Neuron 6**	**Neuron 8**	**Neuron 10**
Hidden Layer 2				

Table 11.Contribution of each neuron in the output for model 4.

- **Comparison of FCNN and Single perceptron for classification task for linearly non-separable data.**

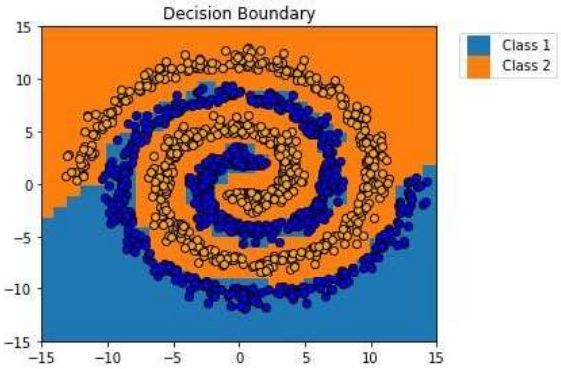
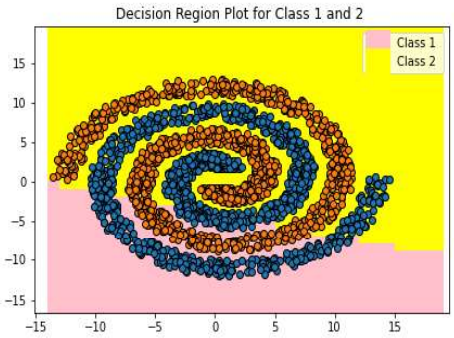
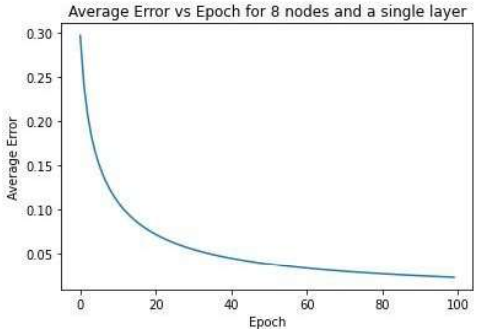
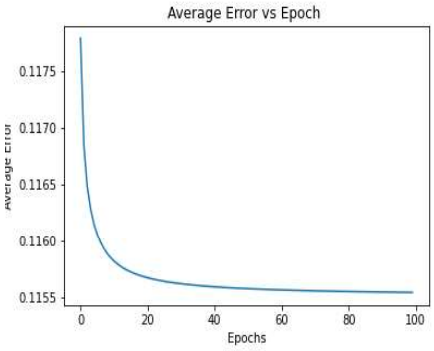
Criterion	FCNN	Single Perceptron
Decision Boundary		
Avg Error V/s Epochs		
Confusion Matrix	Confusion Matrix [[254 7] [4 257]]	[[142 249] [80 311]]
Accuracy	98%	50%

Table 12.Comparison of FCNN and Single perceptron

It can be observed that the single perceptron is not sufficient to classify non-linearly separable data ,but the network of perceptrons can classify it efficiently.

Regression Task

Case1 :- Univariate Dataset

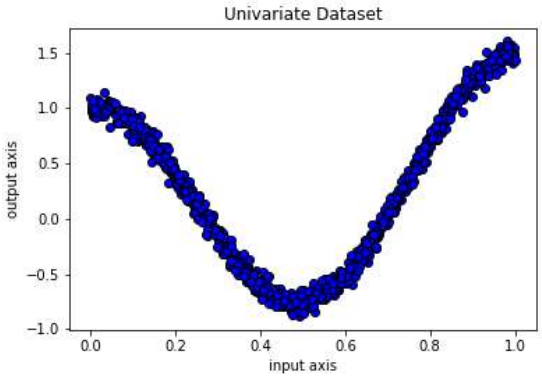


Fig 14.Univariate data plot

- Dataset has 1000 points .Data is Split into Training set, Validation set and Testing set in the ratio of 3:1:1 ,with 600 points in the training data set and 200 points in each validation and test data set.
- Various models with different numbers of nodes in hidden layers and different numbers of epochs are tested and below is the comparison between a few of them.

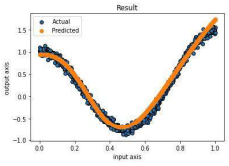
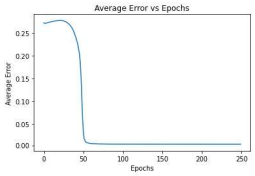
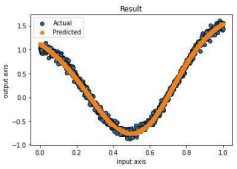
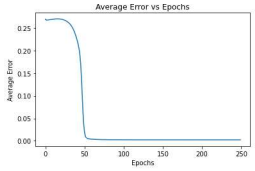
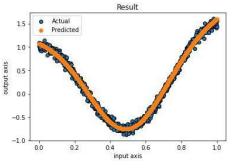
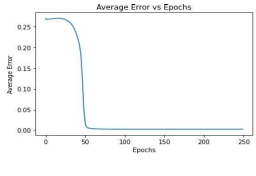
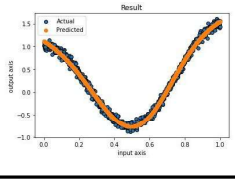
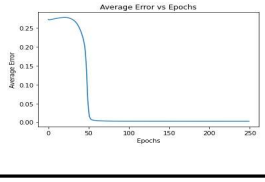
No. of nodes	Decision Boundary	Error v/s Epochs	Number of Epochs	Learning Rate
1			250	0.001
2			250	0.05
3			250	0.01
4			250	0.05

Table 13.Different architectures for univariate data

- Out of all the models we have tested, 4 Models for the optimum result

Model 1	**Model 2**
• Number of Epochs =250 • Learning Rate=0.001 • Number of nodes in hidden layer =1	• Number of Epochs =250 • Learning Rate=0.05 • Number of nodes in hidden layer =2
Model 3	**Model 4**
• Number of Epochs =250 • Learning Rate=0.01 • Number of nodes in hidden layer =3	• Number of Epochs =250 • Learning Rate=0.05 • Number of nodes in hidden layer =4

Table 14.Model architecture

- **We got the best approximation after increasing the nodes to 4 in the hidden layer with a learning rate equal to 0.05 and running it for 250 epochs.**

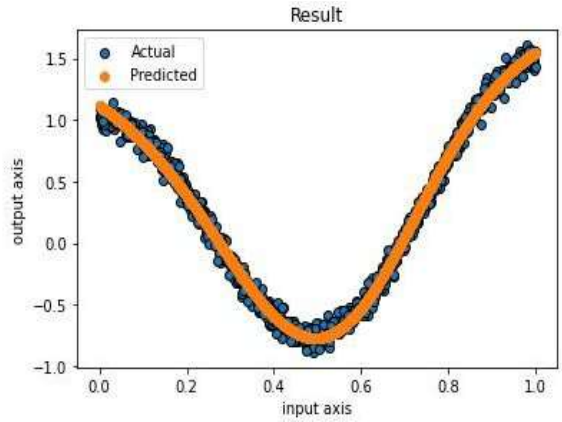


Fig 15.Curve fitting on univariate data

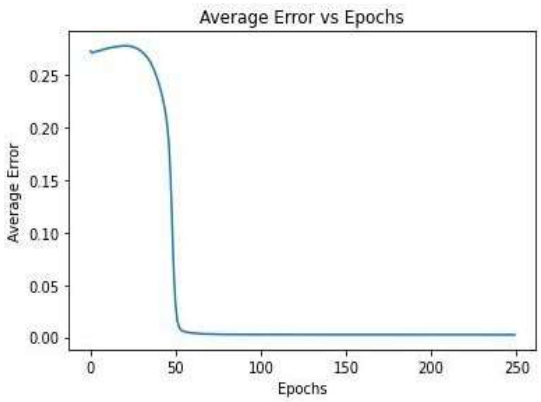


Fig 16.Error v/s Epochs

- **MSE values of training, validation and test data for different models.**

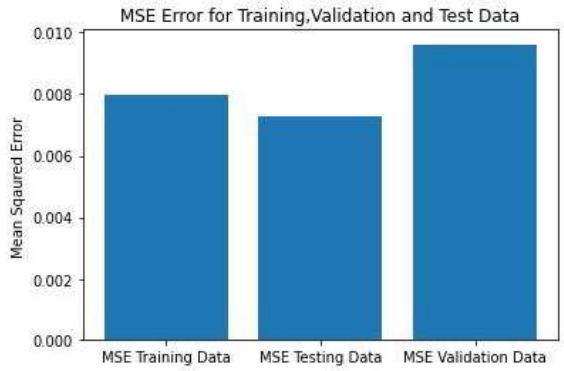
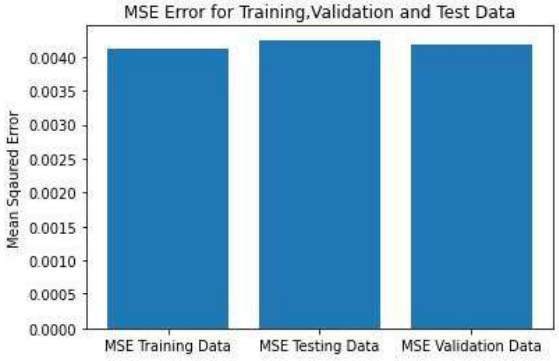
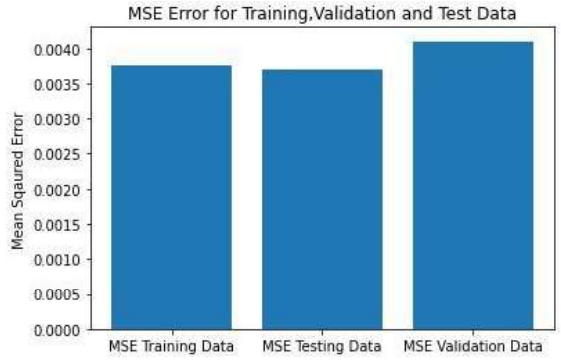
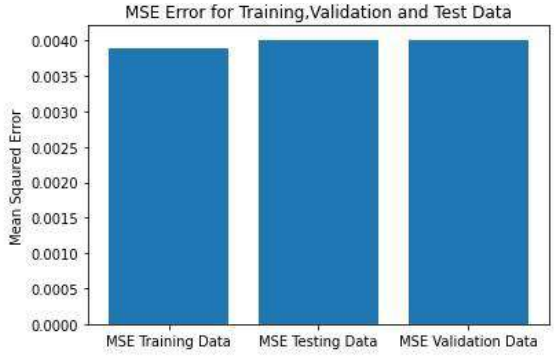
Model 1	Model 2
	
Model 3	**Model 4**
	

Table 15.MSE for different models

Mean Squared Error for the test data of the best architecture that is Model 4 is 0.0038

- **Plots of model output and target output for training data, validation data and test data**

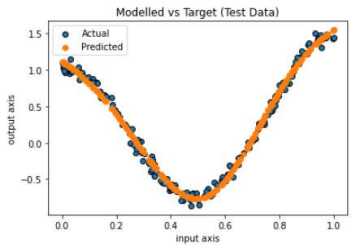
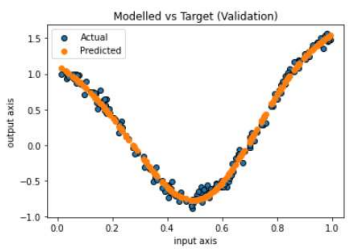
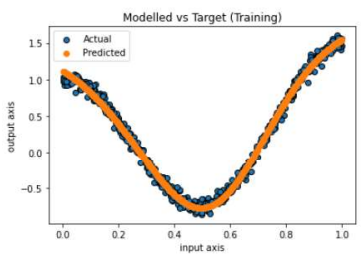
Training Data	Validation Data	Testing Data
		

Table 16. Plot for train ,validation,test output for best model.

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

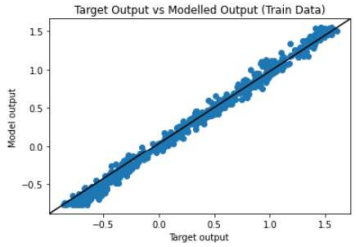
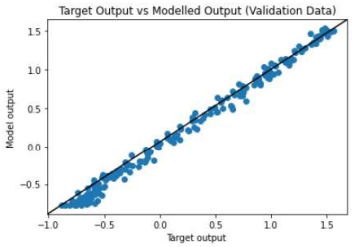
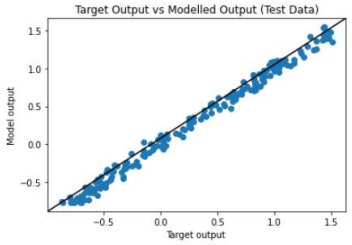
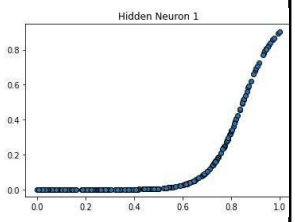
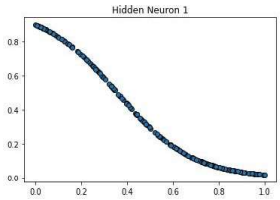
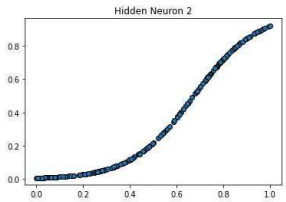
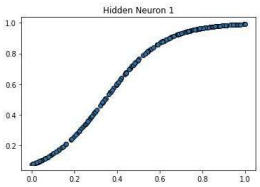
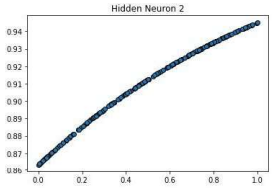
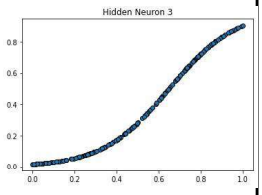
Training Data	Validation Data	Testing Data
		

Table 17.Scatter plot for train ,validation,test output for best model.

- **Plots of outputs for each of the hidden nodes and output nodes in FCNN**

Model	Neuron 1	Neuron 2	Neuron 3	Neuron 4
Model 1				
Model 2				
Model 3				

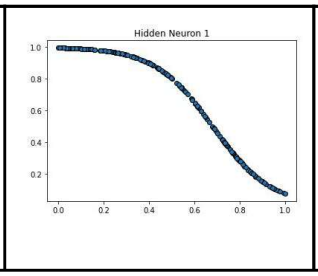
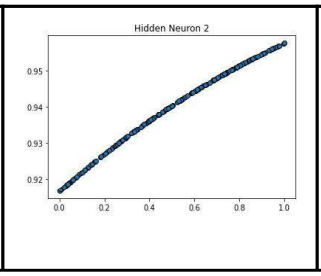
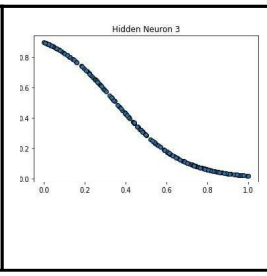
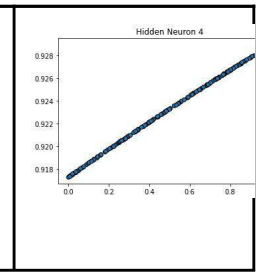
Model 4				

Table 18.Contribution of different neurons for the output.

It can be observed that different neurons learn a different pattern and stitch them together to give the result

- **Comparison of the performance of FCNN with that of the single neuron model.**

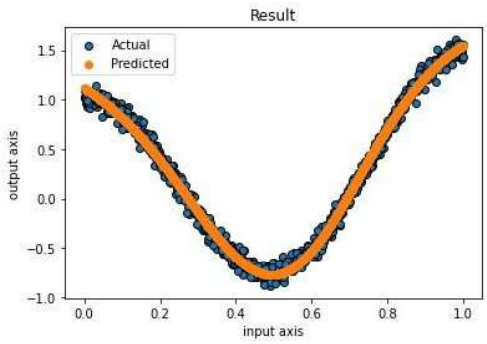
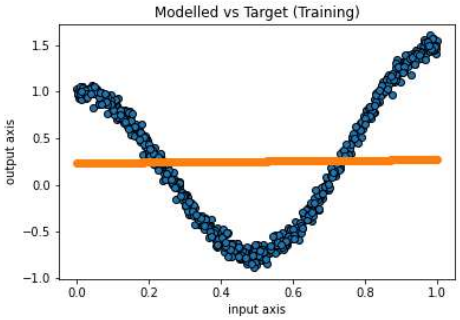
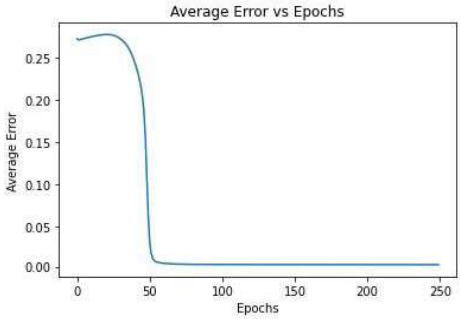
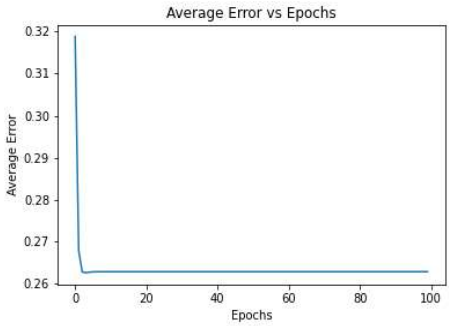
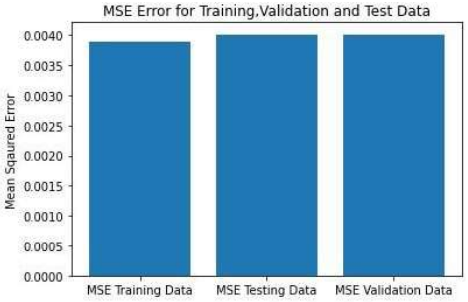
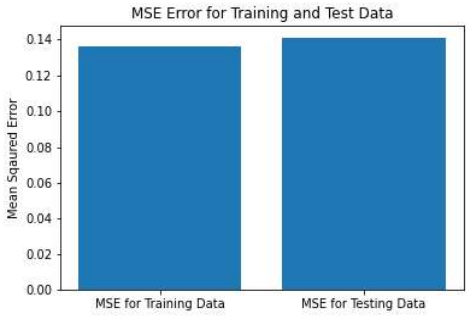
Criterion	FCNN	Single Perceptron
Decision Boundary		
Avg Error V/s Epochs		
Mean Squared Error		

Table 19.Comparison of the performance of FCNN with that of the single neuron model

It can be observed that the single perceptron is not sufficient to fit a nonlinear curve on the data,but the network of perceptrons can capture it efficiently.

Case2 :- Bivariate Dataset

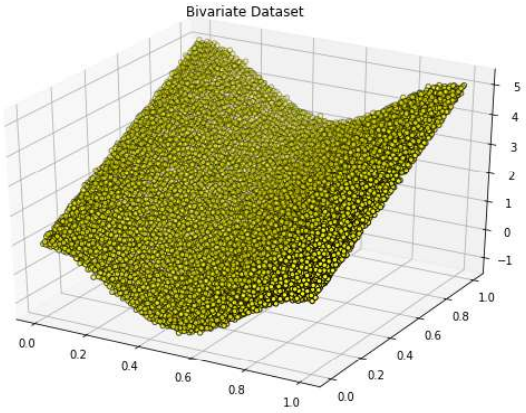
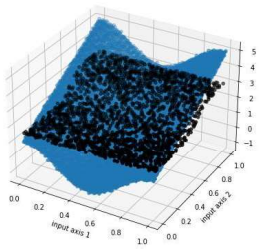
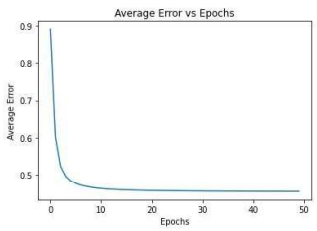
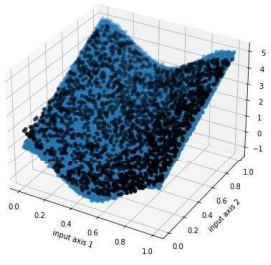
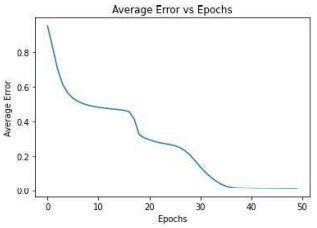


Fig 17.Plot for Bivariate data

- Dataset has 2-D data with 10200 points. Data is split into Training set, Validation set and testing set in the ratio of 3:1:1, with 6120 points in the training data set and 2040 points in each validation and test data set.
- Various models with different numbers of nodes in hidden layers and different numbers of epochs are tested and below is the comparison between a few of them.

No. of nodes in 1st layer	No. of nodes in 2nd layer	Decision Boundary	Error v/s Epochs	Number of Epochs	Learning Rate
1	1			50	0.01
2	2			50	0.05

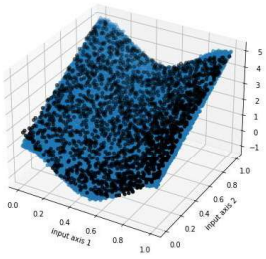
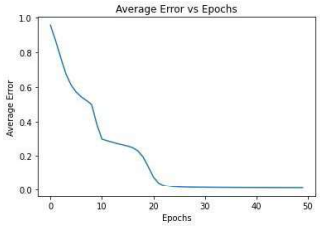
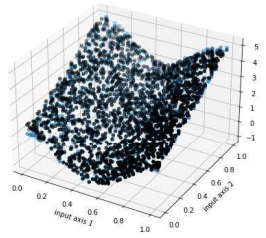
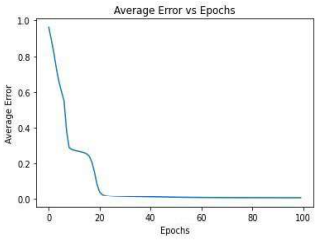
3	3			50	0.05
4	4			100	0.01

Table 20.Different architectures for bivariate data

- We have tested 4 Models for the optimum result

Model 1	**Model 2**
• Number of Epochs =50 • Learning Rate=0.01 • Number of nodes in 1st layer =1 • Number of nodes in 2nd layer =1	• Number of Epochs =50 • Learning Rate=0.05 • Number of nodes in 1st layer =2 • Number of nodes in 2nd layer =2
Model 3	**Model 4**
• Number of Epochs =50 • Learning Rate=0.05 • Number of nodes in 1st layer =3 • Number of nodes in 2nd layer =3	• Number of Epochs =100 • Learning Rate=0.01 • Number of nodes in 1st layer =4 • Number of nodes in 2nd layer =4

Table 21.Model architecture.

- **We got the best approximation after increasing the nodes to 4 in the hidden layer.**

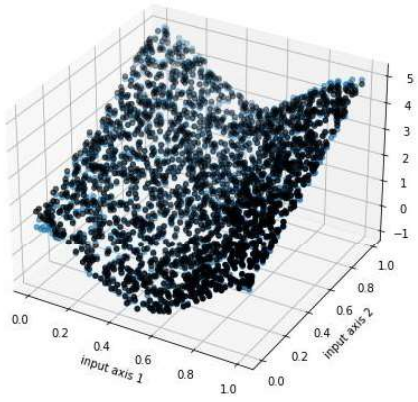


Fig 18.Surface fitting for bivariate data

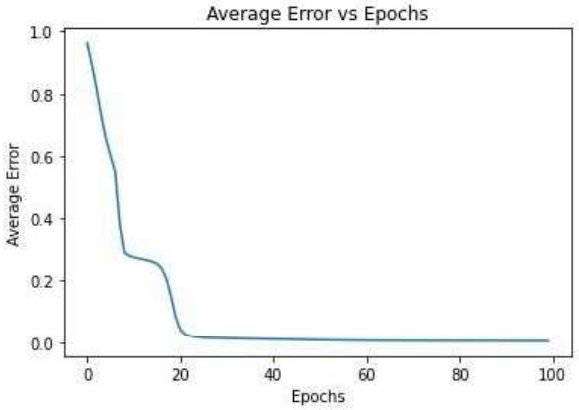


Fig 19.Error v/s Epochs

- **MSE values of training, validation and test data for different models.**

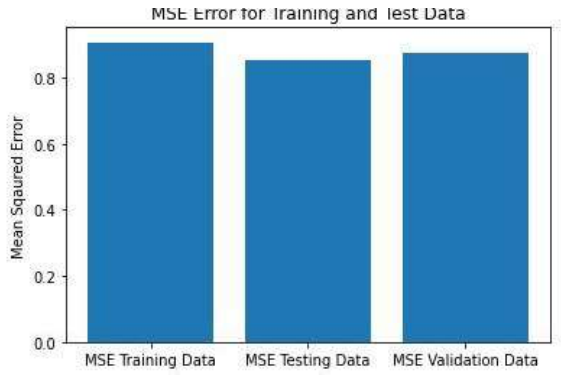
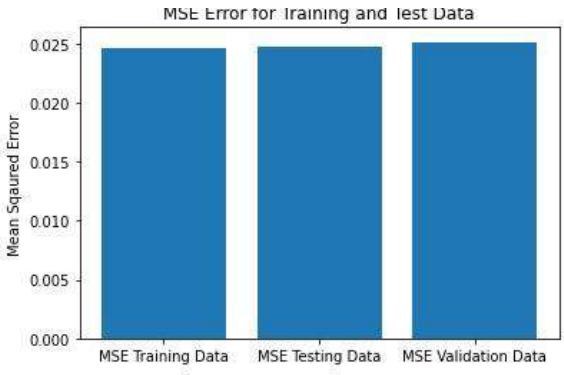
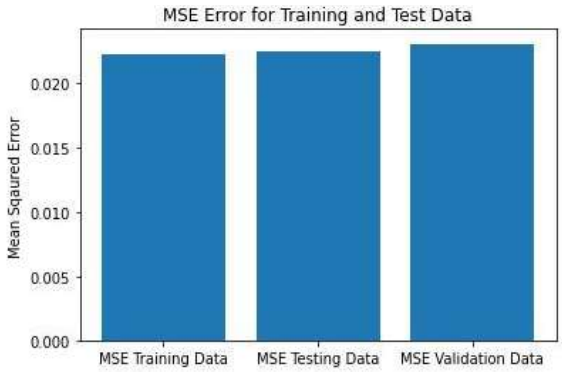
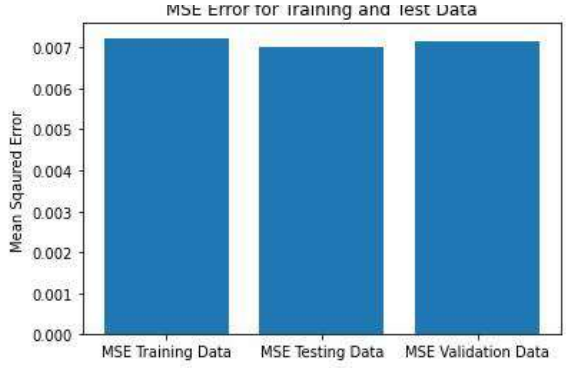
Model 1	Model 2
	
Model 3	**Model 4**
	

Table 22.MSE for different models

- **Mean Squared Error for the test data of the best architecture that is Model 4 is 0.0068**

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

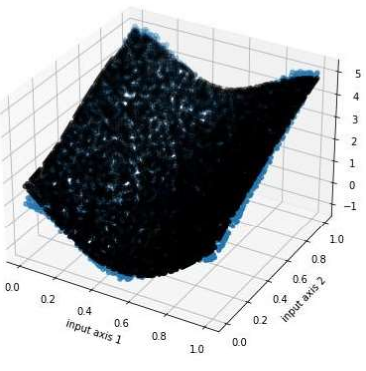
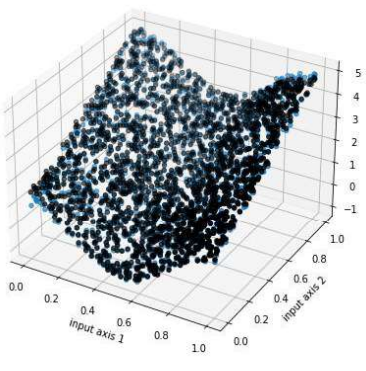
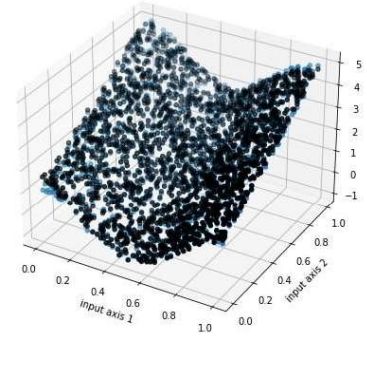
Training Data	Validation Data	Testing Data
		

Table 23.Plot for train ,validation,test output for best model.

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

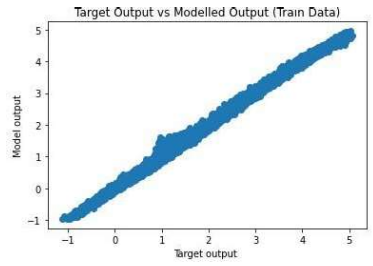
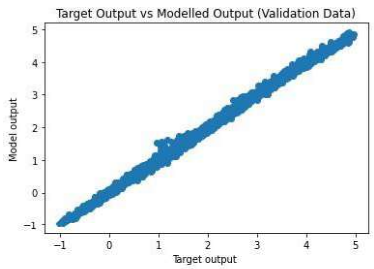
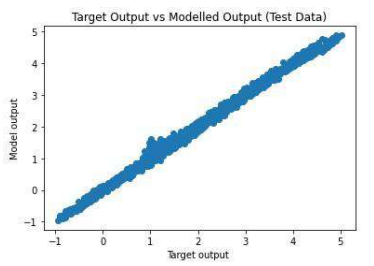
Training Data	Validation Data	Testing Data
		

Table 24.Scatter plot for train ,validation,test output for best model.

Plots of outputs for each of the hidden nodes and output nodes in FCNN

Model 1 : 4 nodes in hidden layer one 2 nodes in hidden layer two.

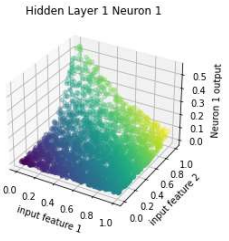
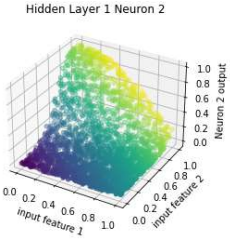
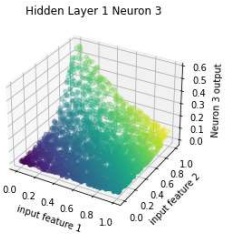
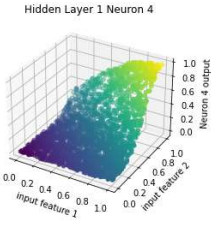
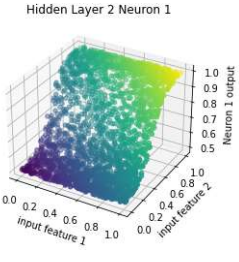
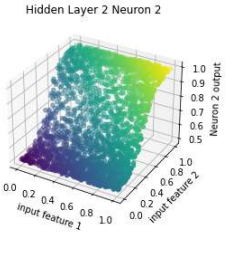
Model 1	Neuron 1	Neuron 2	Neuron 3	Neuron 4
Hidden Layer 1				
Hidden Layer 2				

Table 25.Contribution of different neurons for model 1.

It can be observed that each neuron learns a different pattern and finally stitches them all together to give us the final output.

Model 4 (Best Model) : 4 nodes in hidden layer one 4 nodes in hidden layer two.

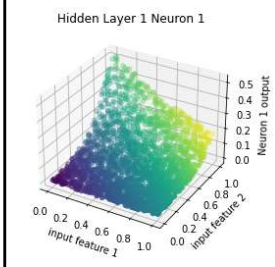
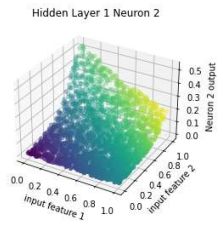
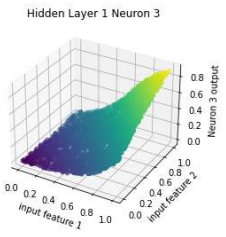
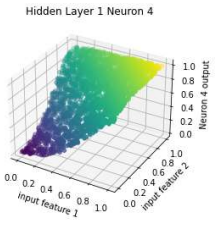
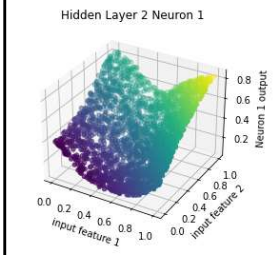
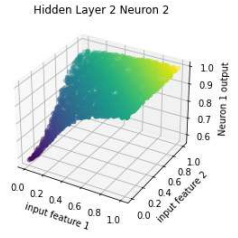
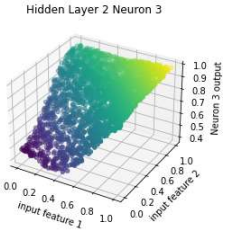
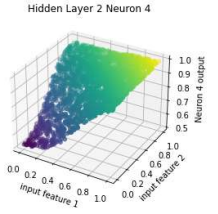
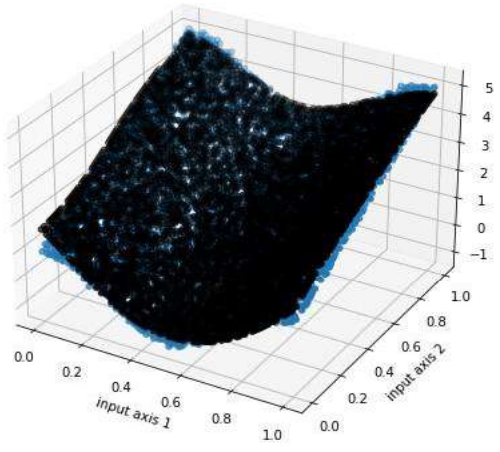
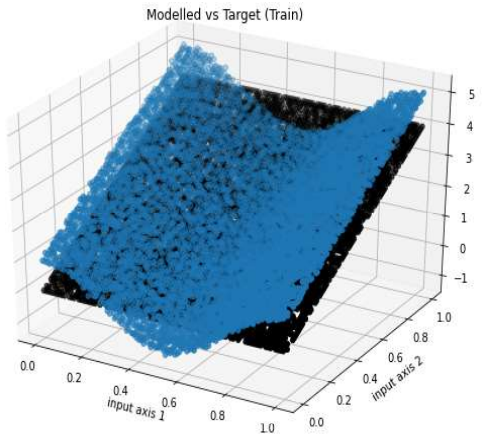
Model 4	Neuron 1	Neuron 2	Neuron 3	Neuron 4
Hidden Layer 1				
Hidden Layer 2				

Table 26.Contribution of different neurons for best model.

Comparison of the performance of FCNN with that of the single neuron model.

Criterion	FCNN	Single Perceptron
Decision Boundary		

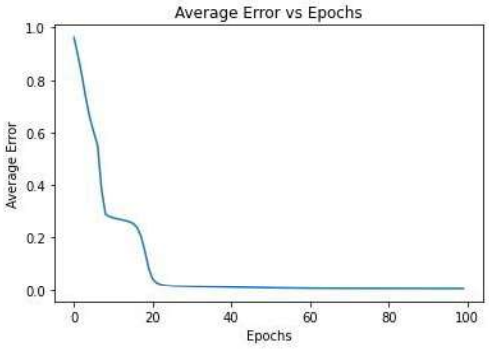
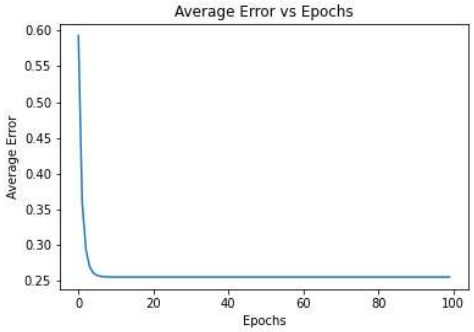
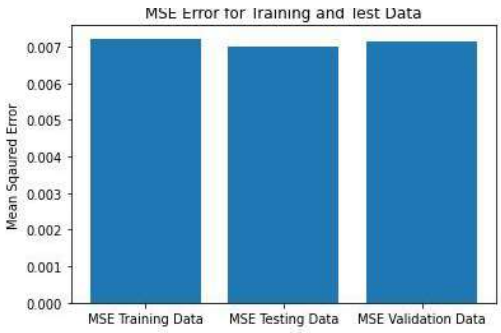
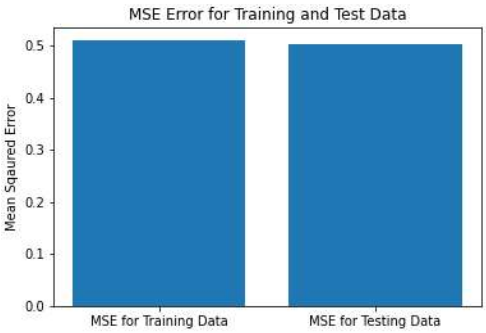
Avg Error V/s Epochs		
Mean Squared Error		

Table 27.Comparison of the performance of FCNN with that of the single neuron model.

It can be observed that the single perceptron is not sufficient to fit a nonlinear surface on the data,but the network of perceptrons can capture it efficiently.