CS-671 Deep Learning and Applications

Assignment 2

Implementing Fully Connected Neural Network for Classification and Regression

---

Submitted by:
Group 13
Prachi Sharma (V21078)
Ayush Tiwari (T22053)

Instructor:
Prof. A D Dileep

March 7, 2023

Table of Contents

1. Introduction

As we concluded in our previous assignment, a single neuron model does not perform well to classify non linearly separable data as well as cannot approximate a nonlinear curve to fit a nonlinear data. Therefore, a network of neurons is used to approximate a nonlinear boundary by piecewise stitching of linear functions from each neuron. This network usually has neuron stacks, called layers. A network with more than two (hidden) layers is called a *deep neural network* (DNN).

A fully connected neural network (FCNN) is a deep neural network of an input layer, two or more hidden layers and an output layer, such that each neuron in a layer is connected to each neuron in the next layer.

Fully Connected Neural Network Architecture

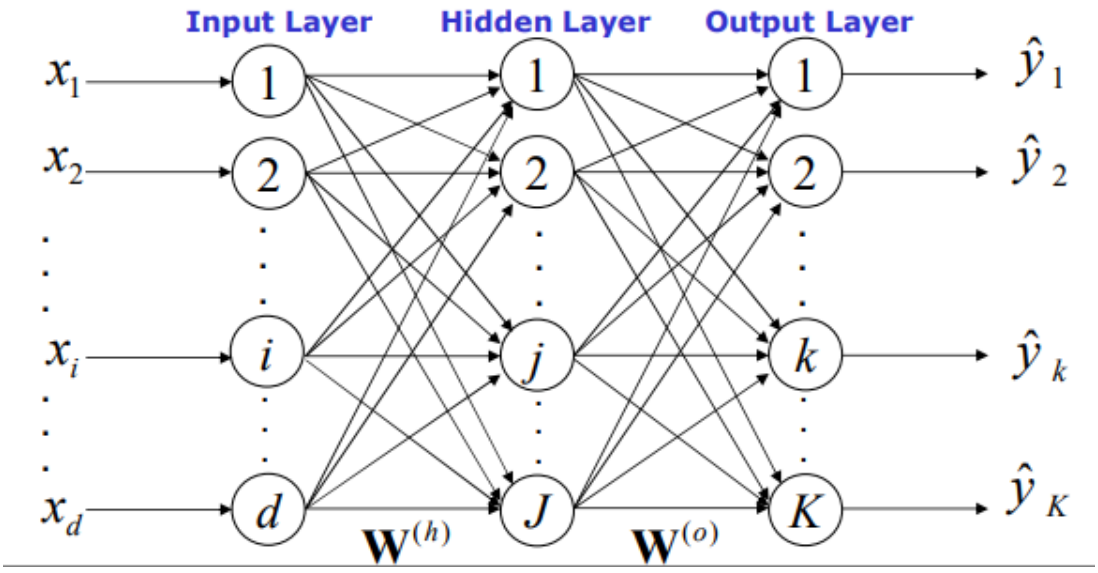


Figure 1.1: Fully connected neural network with single hidden layer

Forward Propagation and Backpropagation Algorithm

Figure 1.1 depicts the architecture of a FCNN with a single hidden layer. The model works in two parts, forward and backward propagation.

- The input layer consists of d linear neurons which return the input value as output value. Here, the input pattern is of dimension d. This input pattern is sent to the input layer.
- The hidden layer consists of J sigmoidal neurons (logistic or tanh) same as the perceptron. The outputs of the input layer are sent to the hidden layer.
- The output layer consists of K sigmoidal neurons in case of classification (K classes) or linear neurons in case of regression (K unknown variables). The outputs of the hidden layer are sent to the output layer.
- Each neuron is connected to every neuron in the next layer through weights.
- These weights help in approximating the complex nonlinear functions for classification/regression tasks.
- The number of neurons in each layer and the number of layers are decided experimentally.
- The activation value of the output layer is used to calculate the error function at the end of one forward pass.

To minimize the error, the weights are updated using gradient descent method. To apply gradient descent technique, we use the *backpropagation algorithm*.

2. Implementation of FCNN for Classification

Training Algorithm - Stochastic Gradient Descent (SGD):

1) Initialize $W^{(h)} \in R^{(d \times J)}$ and $W^{(o)} \in R^{(J+1 \times K)}$ with random values.
2) Choose a training pattern from training data and concatenate 1 (for bias term).
3) Forward Computation
   a) Calculate output of each neuron as $s_n = [\, s_{n_1}\ s_{n_2}\ \dots\ s_{n_K}\,]$
4) Backward Computation
   a) Compute instantaneous error, $E_n = \frac{1}{2}(y_n - s_n)^2$
   b) Update weights between hidden and output layers as

$$w_{jk}^{(o)} = w_{jk}^{(o)} + \Delta w_{jk}^{(o)}$$

where $$\Delta w_{jk}^{(o)} = -\ \eta \frac{\partial E_n}{\partial w_{jk}^{(o)}} = \eta (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} h_{nj}$$

Here, $j = 0, 1, 2, \dots, J$ and $k = 1, 2, \dots, K$.

   c) Update weights between input and hidden layers as

$$w_{ij}^{(h)} = w_{ij}^{(h)} + \Delta w_{ij}^{(h)}$$

where

$$\Delta w_{ij}^{(h)} = -\ \eta \frac{\partial E_n}{\partial w_{ij}^{(h)}} = \eta \left[\sum_{k=1}^{K} (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} w_{jk}^{(o)} \right] \frac{\partial g(a_{nk}^{(h)})}{\partial a_{nk}^{(h)}} x_{ni}$$

Here, $i = 0, 1, 2, \dots, d$ and $j = 1, 2, \dots, J$.

We have used $f(a) = tanh(a)$ as an activation function for output neurons and $g(a) = \frac{1}{1+e^{(-a)}}$ as an activation function for hidden neurons.

5) Repeat steps 2 to 4 for all training patterns.

6) Compute average error, $E_{av} = \frac{1}{N}\sum_{n=1}^{N} E_n$

7) Repeat steps 2 to 6 till the number of epochs has reached a certain limit or average error is less than a threshold value.

Testing Algorithm:

1) For a given test example $\bar{x}$, compute output at each of the output neurons using final weights obtained after training has ended.

$$\hat{y}_k = f(a_k^{(o)})$$

2) Classify $\bar{x}$ according to the rule,

Class label for $\bar{x} := arg\ max\{\hat{y}_k : k = 1, 2, ..., K\}$

2.1. Classifying linearly separable classes using a single hidden layer FCNN

Results:

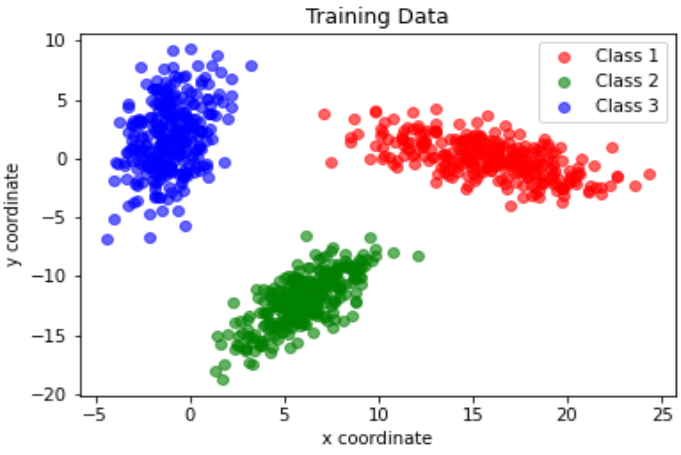


Figure 2.1.1: Scatter plot of data points. We can see that the data is linearly separable.

Cross-validation results

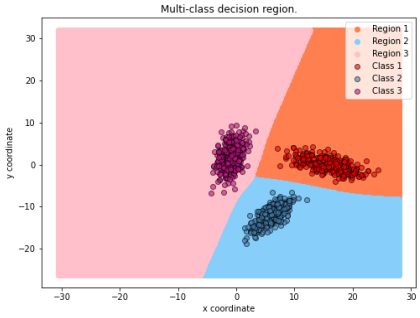
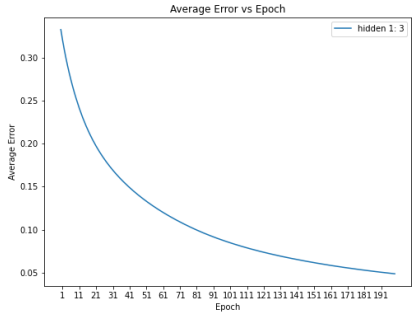
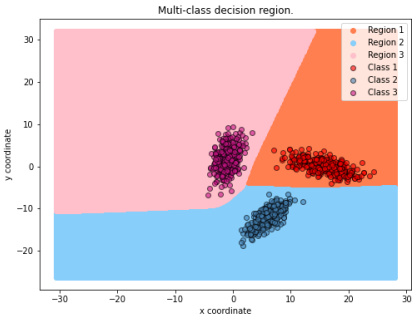
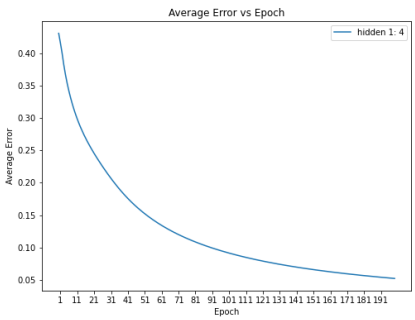
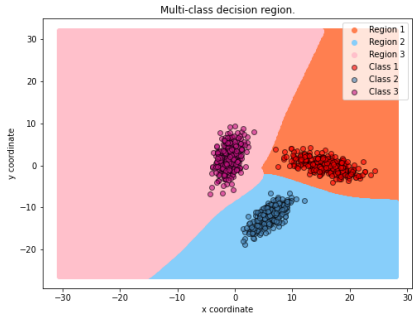
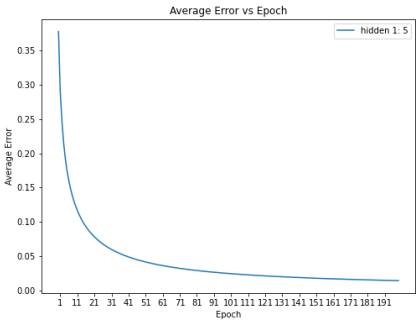
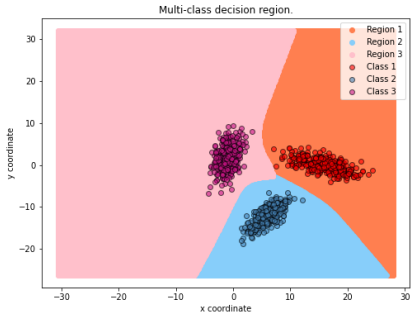
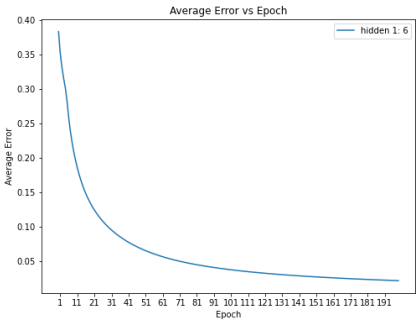
Model architecture	Decision boundary	Error v/s epoch	Training error	Confusion matrix	Classification accuracy
2I-3H-3O			0.04867	**[[100 0 0] [0 100 0] [0 0 100]]**	**100**
2I-4H-3O			0.05215	[[100 0 1] [0 100 0] [0 0 99]]	99.67
2I-5H-3O			**0.01417**	**[[100 0 0] [0 100 0] [0 0 100]]**	**100.0**
2I-6H-3O			0.02222	**[[100 0 0] [0 100 0] [0 0 100]]**	**100.0**

Table 2.1.1: The table compares results of various models. xI-yH-zO represents x units in the input layer, y units in the hidden layer and z units in the output layer. Each of the models have been trained for 200 epochs with "tanh" activation at hidden layer and "logistic" activation at output layer. The learning rate is 0.01. The confusion matrix and classification accuracy has been shown for validation data. Here, the columns represent **actual** class and rows represent **predicted** class.

After analyzing the above results of different models on validation data, we can see that the model with 5 neurons in the hidden layer performs the best. Since the data is linearly separable, the classification accuracy is nearly the same in all models but training error is least for the model with 5 neurons in the hidden layer.

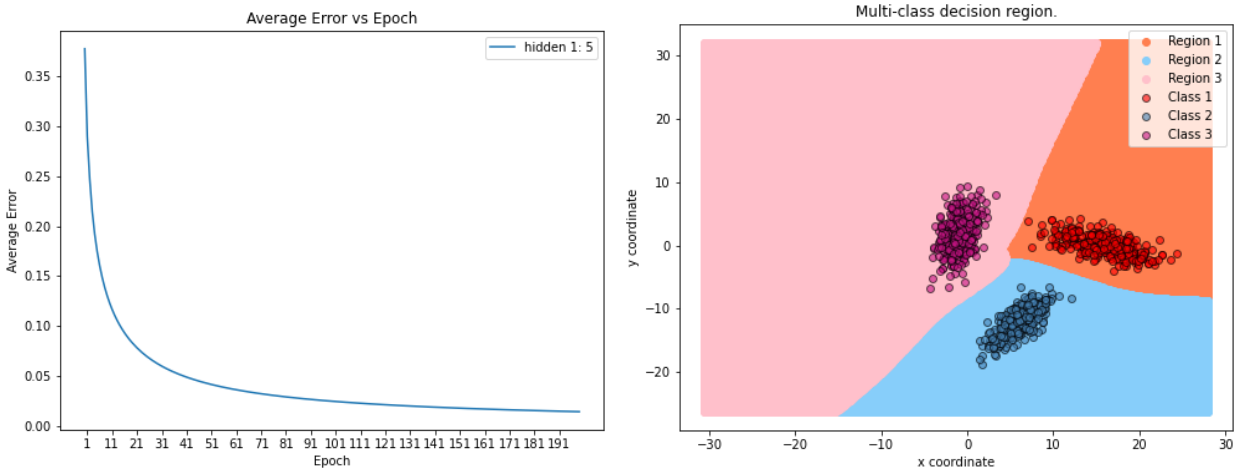


Figure 2.1.2: On the left, we have average error versus epoch plot. On the right, we have a decision region boundary for three class classification for the best selected model ie. 2I-5H-2O.

We can see in the above plot that the average training error is decreasing exponentially. The decision boundary is a nonlinear curve as we have used a network of sigmoidal neurons.

Performance measures on test data (best selected model) :

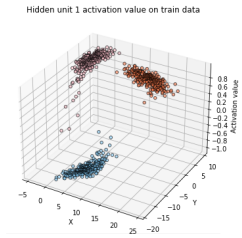
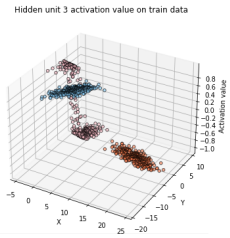
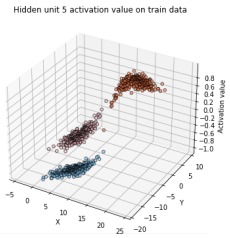
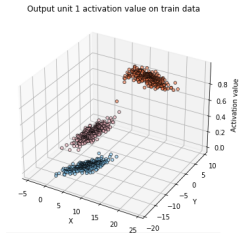
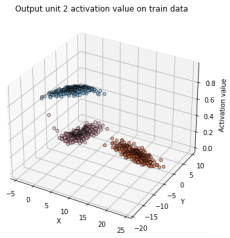
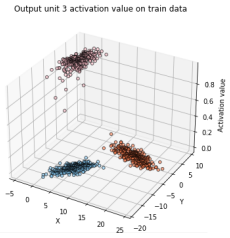
- Confusion matrix:

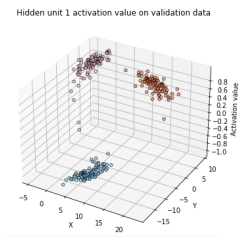
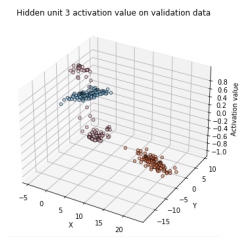
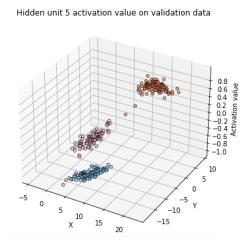
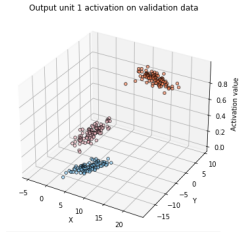
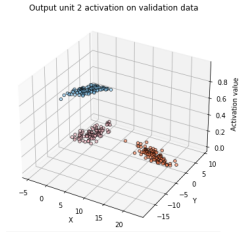
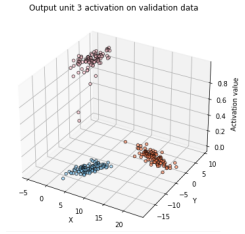
Predicted Class	Actual Class		
	100	0	0
	0	100	0
	0	0	100

- Accuracy of model: 100%
- Average precision: 1.0
- Average recall: 1.0
- Average F-measure: 1.0

Hence, the model performs really well on test data as well.

Plots for outputs of hidden as well as output units in each layer for training , validation and test data

Training data			
Hidden Neurons	 Hidden unit 1	 Hidden unit 3	 Hidden unit 5
Output Neurons	 Output unit 1	 Output unit 2	 Output unit 3

Validation data			
Hidden Neurons	 Hidden unit 1	 Hidden unit 3	 Hidden unit 5
Output Neurons	 Output unit 1	 Output unit 2	 Output unit 3

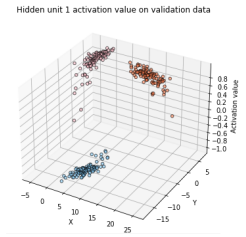
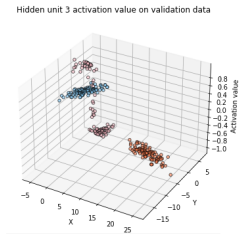
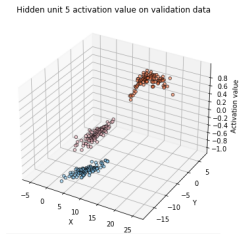
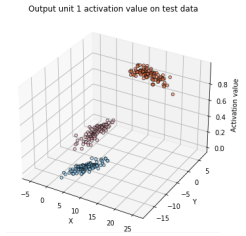
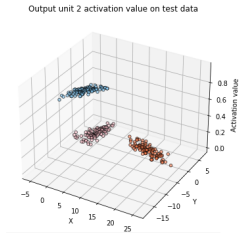
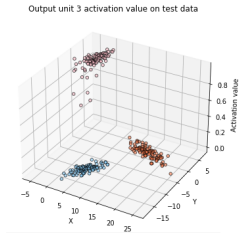
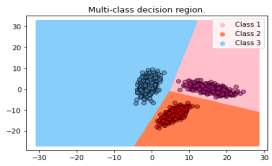
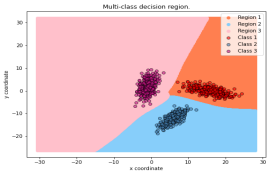
Test data			
Hidden Neurons	 Hidden unit 1	 Hidden unit 3	 Hidden unit 5
Output Neurons	 Output unit 1	 Output unit 2	 Output unit 3

Table 2.1.2: Above tables show 3D output plots of hidden units and output units on training, validation and test data for the best selected architecture.

Since we have used sigmoidal neurons in hidden layers, the 3D plots show an S-shaped curve. This is because each of the hidden units is learning some S-shaped curve that lies in the range of (-1,1). Finally, the output units are able to separate the three classes.

Performance comparison with Single Neuron Model

Performance Parameters	Decision Region	Confusion Matrix	Classification Accuracy	Average Precision	Average Recall	Average f-measure
Single Neuron Model		**[[300 0 0] [0 300 0] [0 0 300]]**	**100.0**	**1.0**	**1.0**	**1.0**
Fully Connected Neural Network		**[[100 0 0] [0 100 0] [0 0 100]]**	**100.0**	**1.0**	**1.0**	**1.0**

Conclusion:

The implementation shows that a fully connected neural network with a single hidden layer on a linearly separable data performs well to approximate a nonlinear curve separating the classes. Since the data is linearly separable, there exists linear hyperplanes that separates the data. Therefore, a single neuron model can also learn appropriate weights to approximate the hyperplanes and classify the data with high accuracy.

2.2. Classifying non linearly separable classes using two hidden layers FCNN

The training algorithm is the same as we used in the single layer model. Only the backpropagation operations are changed as follows:

Backward Computation

a) Compute instantaneous error, $E_n = \frac{1}{2}(y_n - s_n)^2$

b) Update weights between hidden layer 2 and output layer as

$$w_{lk}^{(o)} = w_{lk}^{(o)} + \Delta w_{lk}^{(o)}$$

where $$\Delta w_{lk}^{(o)} = -\eta \frac{\partial E_n}{\partial w_{lk}^{(o)}} = \eta(y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} h_{nl}$$

Here, $l = 0, 1, 2, \ldots, L$ and $k = 1, 2, \ldots, K$

c) Update weights between hidden layer 1 and hidden layer 2 as

$$w_{jl}^{(h2)} = w_{jl}^{(h2)} + \Delta w_{jl}^{(h2)}$$

where

$$\Delta w_{jl}^{(h2)} = -\eta \frac{\partial E_n}{\partial w_{jl}^{(h2)}} = \eta \left[\sum_{k=1}^{K} (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} w_{lk}^{(o)} \frac{\partial g(a_{nl}^{(h2)})}{\partial a_{nl}^{(h2)}} \right] h_{nj}$$

Here, $l = 0, 1, 2, \ldots, L$ and $j = 1, 2, \ldots, L$

d) Update weights between input layer and hidden layer 1 as

$$w_{ij}^{(h1)} = w_{ij}^{(h1)} + \Delta w_{ij}^{(h1)}$$

where

$$\Delta w_{ij}^{(h1)} = -\eta \frac{\partial E_n}{\partial w_{ij}^{(h1)}} = \eta \left[\sum_{l=1}^{L} \left[\sum_{k=1}^{K} (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} w_{lk}^{(o)} \frac{\partial g(a_{nl}^{(h2)})}{\partial a_{nl}^{(h2)}} \right] w_{jl}^{(h2)} \frac{\partial g(a_{nj}^{(h1)})}{\partial a_{nj}^{(h1)}} \right] x_i$$

Here, $j = 0, 1, 2, \ldots, d$ and $j = 1, 2, \ldots, J$

We have used $f(a) = tanh(a)$ as an activation function for output neurons and $g(a) = \frac{1}{1+e^{(-a)}}$ as an activation function for hidden neurons in both hidden layers.

Results:

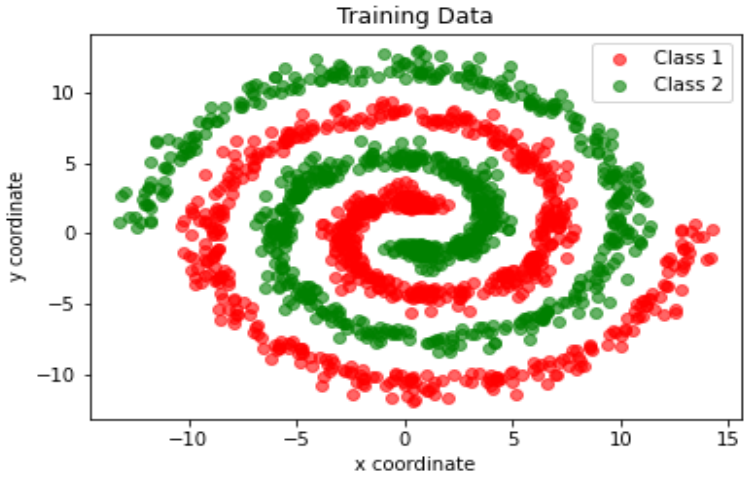


Figure 2.2.1: Scatter plot of data points.

We can see that the data is spiral in shape, hence it is nonlinear.

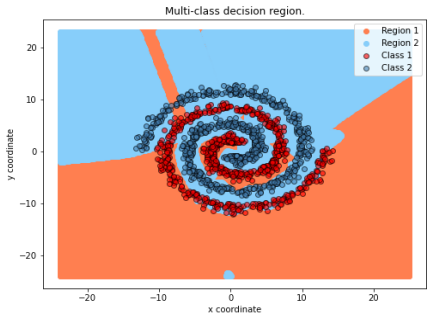
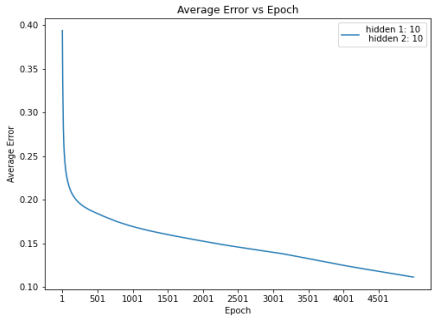
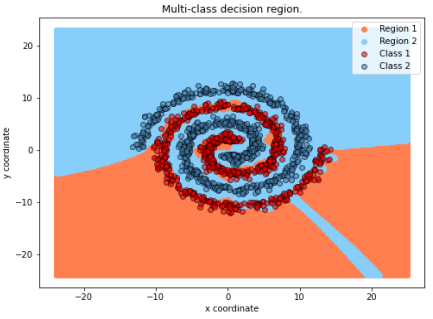
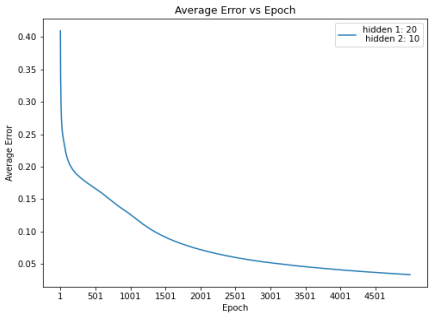
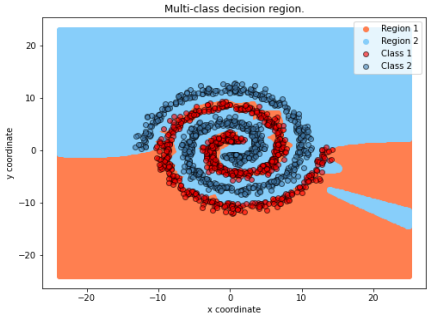
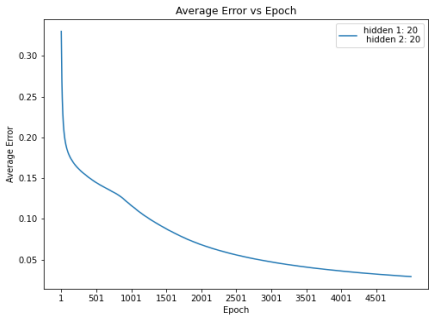
Cross-validation results					
Model architecture	Decision boundary	Error v/s epoch	Training error	Confusion matrix	Classification accuracy
2I-10H-10 H-2O			0.11128	[[196 17] [65 245]]	84.32%
2I-20H-10 H-2O			0.03361	[[220 12] [40 248]]	90.00%
2I-20H-20 H-2O			**0.02913**	**[[238 9] [22 251]]**	**94.04%**

Table 2.2.1: The table compares results of various models. xI-yH1wH2-zO represents x units in the input layer, y units in the hidden layer 1, w units in the hidden layer 2 and z units in the output layer. Each of the models have been trained for 5000 epochs with tanh activation at hidden layer and logistic activation at output layer. The learning rate is 0.01 for all models. The confusion matrix and classification accuracy has been shown for validation data. Here, the columns represent actual class and rows represent predicted class.

After analyzing the above results of different models on validation data, we can see that the model with 20 neurons in the both hidden layers performs the best. Since the data is inherently complex and is not linearly separable, the model is taking too long (5000 epochs) to learn the underlying patterns in data. Also, when I tried using a smaller learning rate (0.001) to train the model, the results were not as

good because then the model was taking too long (>5000 epochs) to reach an optimal solution. Models with an even higher number of hidden units (>20) were also tried with different learning rate and training time but there was no significant improvement in the performance accuracy. Better initialization techniques may improve the solution.

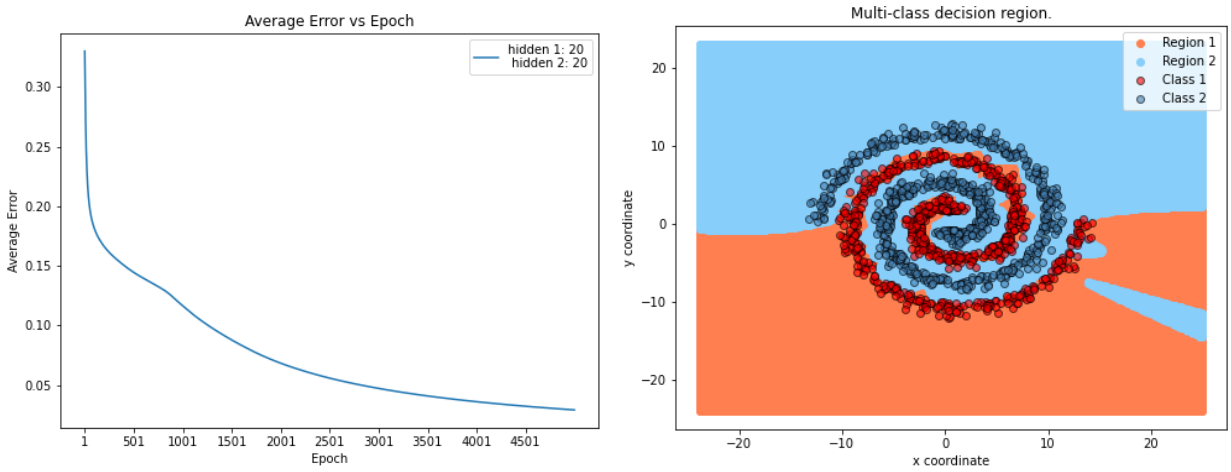


Figure 2.1.2: On the left, we have average error versus epoch plot. On the right, we have the decision region for two class classification.

It seems that during the first 800 epochs of training, the error decreased rapidly in an exponential manner. However, after this point, the error started to increase for a few epochs before decreasing again. It is normal for training error to increase during early stages of training as the model is still learning the complex nature of data.

Performance measures on test data (best selected model) :

- Confusion matrix:

	Actual Class	
Predicted Class	239	8
	22	254

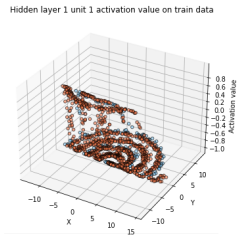
- Accuracy of model: 94.26%

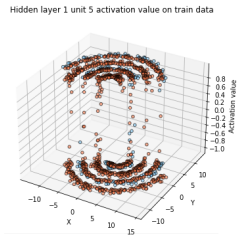
- Precision for class 1: 0.9676
  Precision for class 2: 0.9202
  Average precision: 0.9439

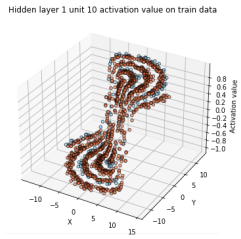
- Recall for class 1: 0.9157
  Recall for class 2: 0.9694
  Average recall: 0.9425

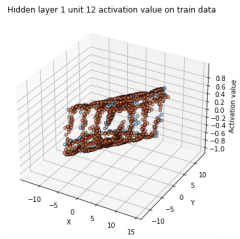
- F-measure for class 1: 0.9157
  F-measure for class 2: 0.9694
  Average f-measure: 0.9425

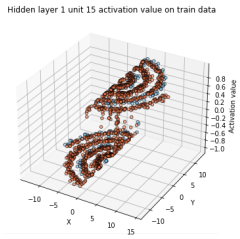
The model performs well on the test data as well but still, it is not able to correctly classify 100% of the data. It may be possible to capture more complex information of data with deeper networks.

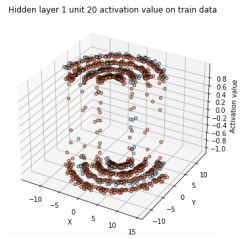
<table>
<tr><th colspan="4">Training data</th></tr>
<tr><td rowspan="2">Hidden layer 1 neurons</td><td> 

Hidden neuron 1</td><td> 

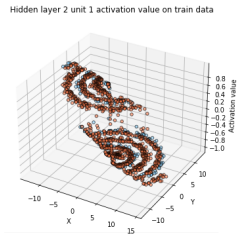
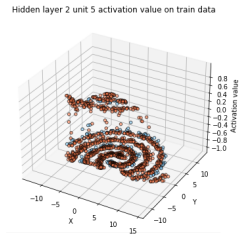
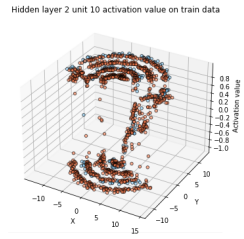
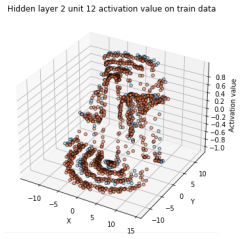
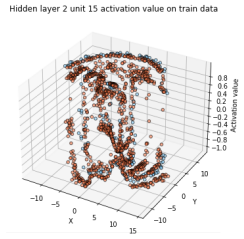
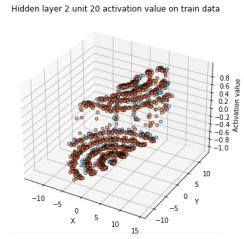
Hidden neuron 5</td><td> 

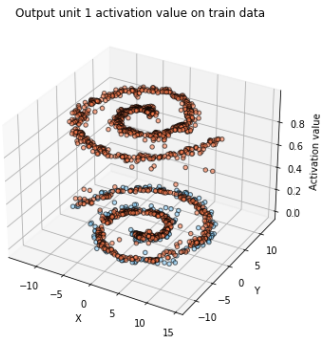
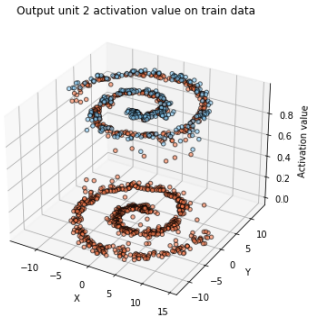
Hidden neuron 10</td></tr>
<tr><td> 

Hidden neuron 12</td><td> 

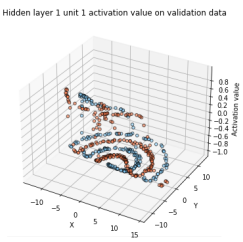
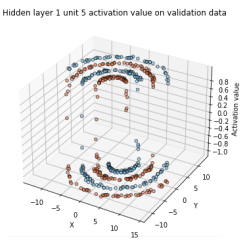
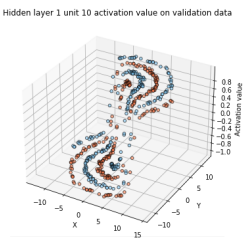
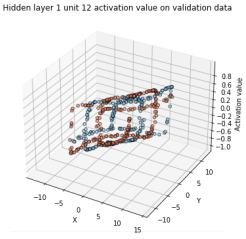
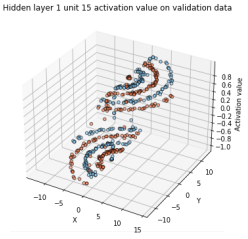
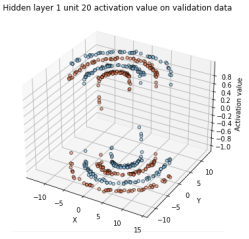
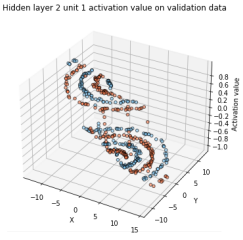
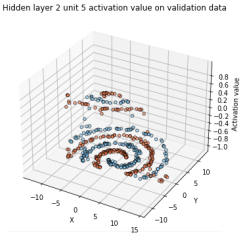
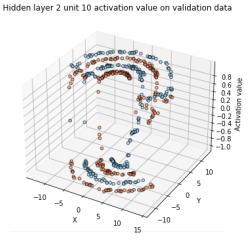
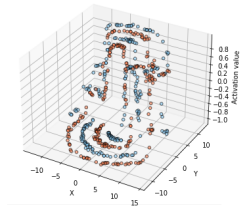
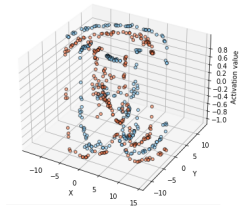
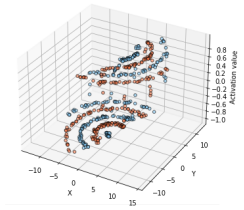
Hidden neuron 15</td><td> 

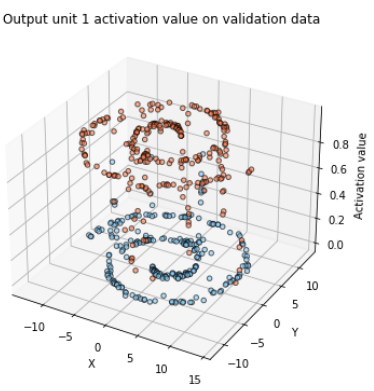
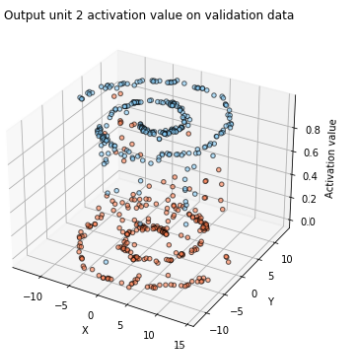
Hidden neuron 20</td></tr>
</table>

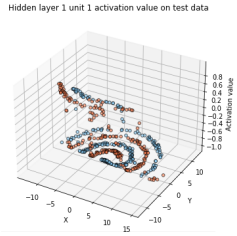
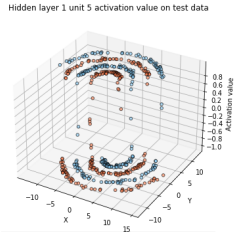
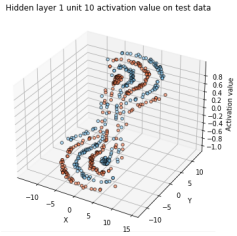
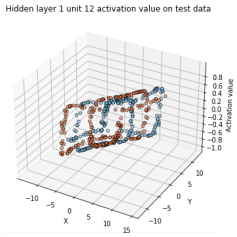
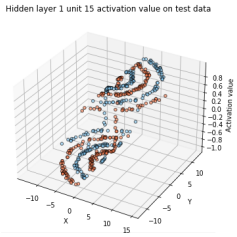
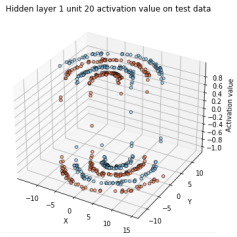
Hidden layer 2 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20

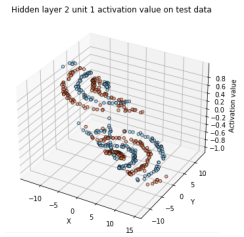
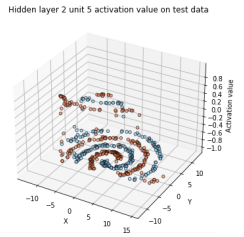
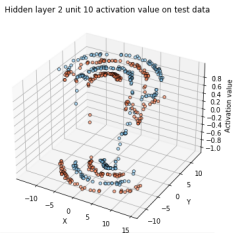
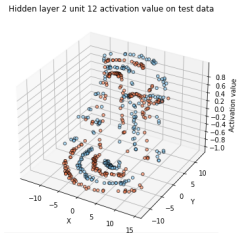
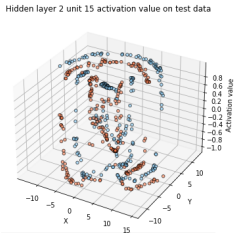
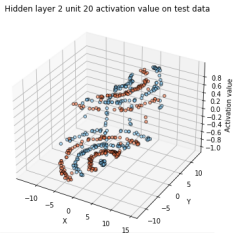
Output neurons	 Output neuron 1	 Output neuron 2

Validation data

Hidden layer 1 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20
Hidden layer 2 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20

Output neurons	 Output neuron 1	 Output neuron 2

Test data			
Hidden layer 1 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20

Hidden layer 2 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20

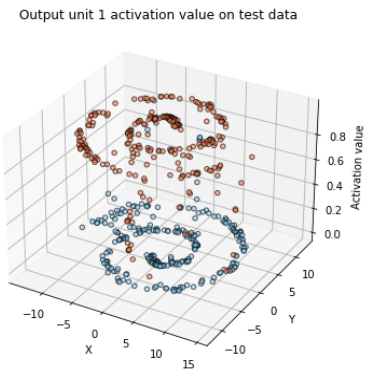
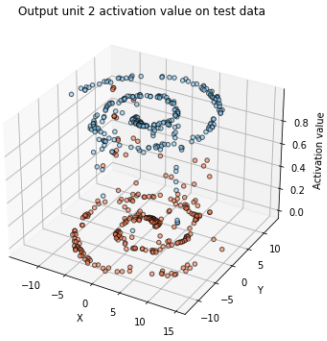
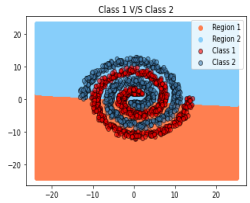
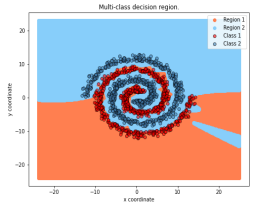
Output neurons	 Output neuron 1	 Output neuron 2

Table 2.2.2: Above tables show 3D output plots of hidden units and output units on training, validation and test data for the best selected architecture.

Since we have used sigmoidal neurons in hidden layers, the 3D plots show an S-shaped curve. This is because each of the hidden units is learning some S-shaped curve that lies in the range of (-1,1). Finally, the output units are able to separate the two classes.

Performance comparison with Single Neuron Model

Performance Parameters	Decision Region	Training Error	Confusion Matrix	Classification Accuracy	Average Precision	Average Recall	Average f-measure
Single Neuron Model		0.11739	[[198 193] [189 202]]	51.15%	0.514	0.513	0.514
Fully Connected Neural Network		**0.01417**	**[[239 8] [22 254]]**	**94.26%**	**0.943**	**0.942**	**0.942**

Conclusion:

The implementation shows that a fully connected neural network with 2 or more hidden layers on a nonlinear data performs well to approximate a complex function. It can capture more information from the data in comparison to a single neuron model.

3. Implementation of FCNN for Regression

Training Algorithm - Stochastic Gradient Descent (SGD):

1) Initialize $W^{(h)} \in R^{(d \times J)}$ and $W^{(o)} \in R^{(J+1 \times K)}$ with random values.
2) Choose a training pattern from training data and concatenate 1 (for bias term).
3) Forward Computation
   a) Calculate output of each neuron as $s_n = [\, s_{n_1}\ s_{n_2}\ \dots\ s_{n_K}\,]$
4) Backward Computation
   a) Compute instantaneous error, $E_n = \frac{1}{2}(y_n - s_n)^2$
   b) Update weights between hidden and output layers as

$$w_{jk}^{(o)} = w_{jk}^{(o)} + \Delta w_{jk}^{(o)}$$

where $$\Delta w_{jk}^{(o)} = -\eta \frac{\partial E_n}{\partial w_{jk}^{(o)}} = \eta (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} h_{nj}$$

Here, $j = 0, 1, 2, \dots, J$ and $k = 1, 2, \dots, K$.

   c) Update weights between input and hidden layers as

$$w_{ij}^{(h)} = w_{ij}^{(h)} + \Delta w_{ij}^{(h)}$$

where

$$\Delta w_{ij}^{(h)} = -\eta \frac{\partial E_n}{\partial w_{ij}^{(h)}} = \eta \left[\sum_{k=1}^{K} (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} w_{jk}^{(o)} \right] \frac{\partial g(a_{nk}^{(h)})}{\partial a_{nk}^{(h)}} x_{ni}$$

Here, $i = 0, 1, 2, \dots, d$ and $j = 1, 2, \dots, J$.

We have used $f(a) = a$ as an activation function for output neurons and $g(a) = tanh(a)$ as an activation function for hidden neurons.

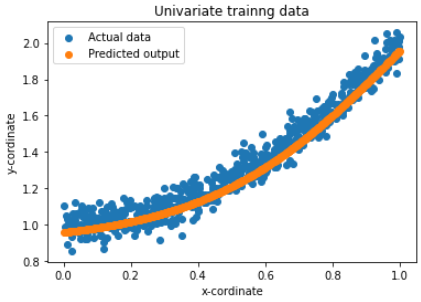
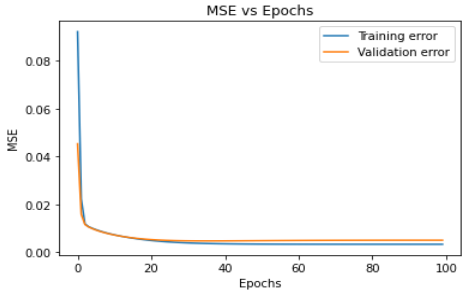
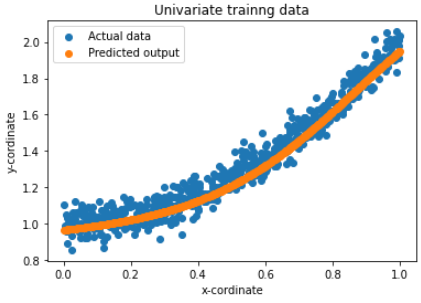
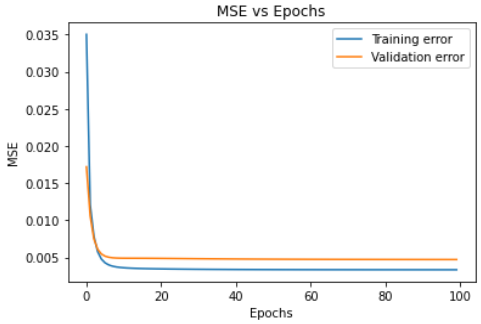
5) Repeat steps 2 to 4 for all training patterns.

6) Compute average error, $E_{av} = \frac{1}{N}\sum_{n=1}^{N} E_n$

7) Repeat steps 2 to 6 till the number of epochs has reached a certain limit or average error is less than a threshold value.

3.1. Regression using one hidden layer

Results:

Cross-validation results				
Model architecture	Best fit curve	Average error vs epoch for train and validation	MSE for train	MSE for validation
1I-3H-1O			0.003371665 2846384307	0.97134304
1I-4H-1O			0.003351699 642232598	0.97458649

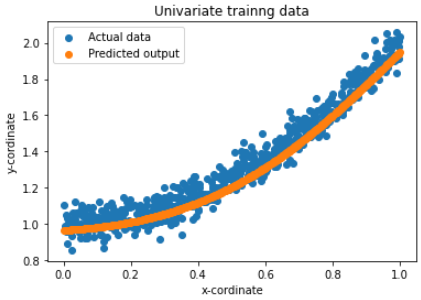
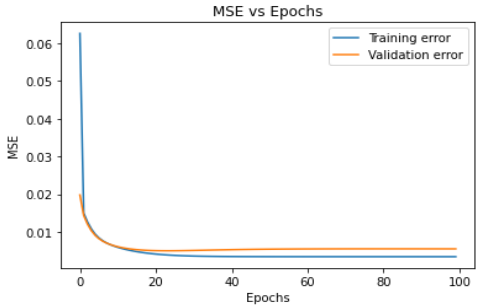
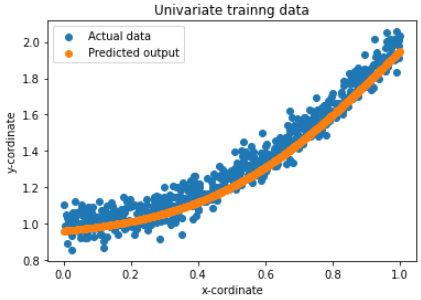
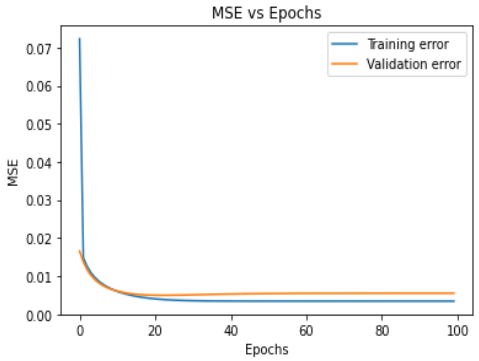
1I-5H-1O			0.003407739085377779	0.97997721
1I-6H-1O			0.003411010 5424279666	0.97089381

Table 3.1.1: The table compares results of various models. Each of the models have been trained for 100 epochs with tanh activation at hidden layer and linear activation at output layer.

After analyzing the above results of different models on validation data, we can see that the model with 6 neurons in the hidden layer performs the best.

The test MSE for the architecture with 1 input neuron, 6 hidden neurons and 1 output neuron is **1.070124.**

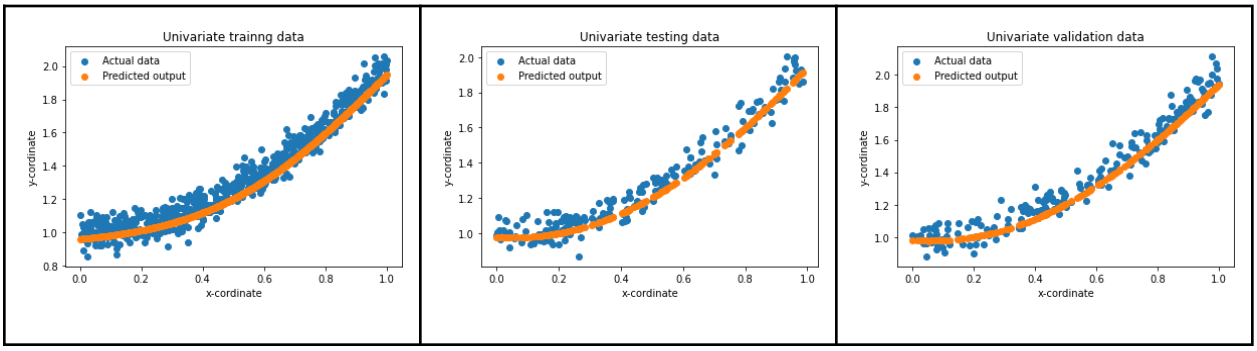


Table 3.1.2: Plots of model output and target output for training data, validation data and test data.

The predicted line is closely following the original data distribution curve for training, testing and the validation data which can be inferred from the above plots.

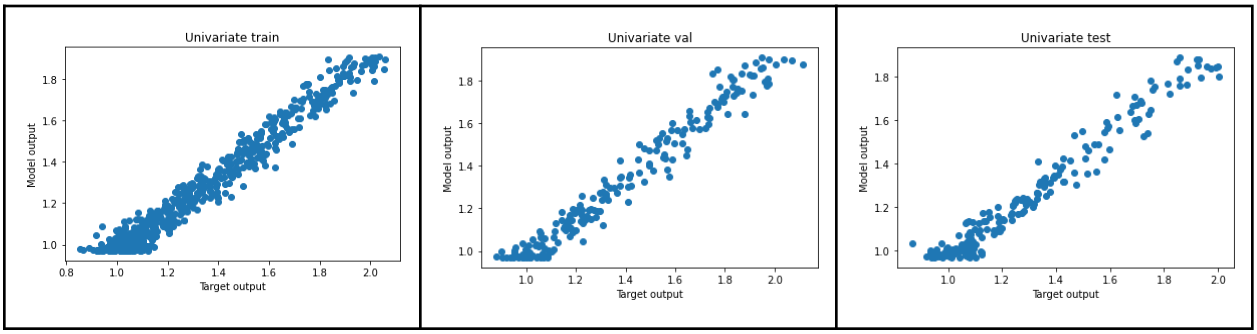
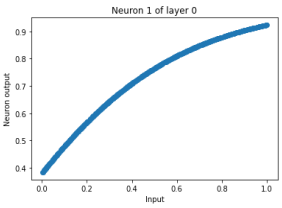
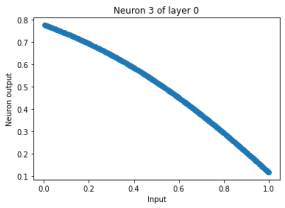
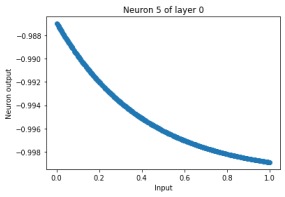
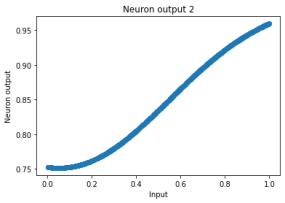
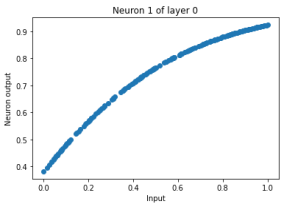
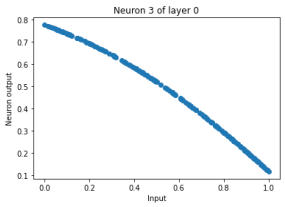
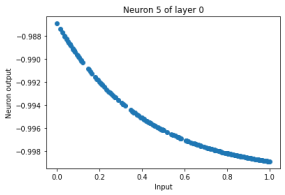
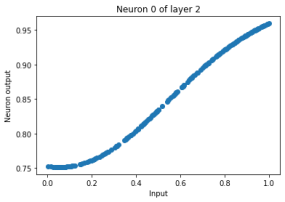


Table 3.1.3: Model vs target output for best architecture of univariate data.

Since the Target vs Model output is approximating to a straight line, it shows that the actual and the predicted values are close to each other, and this indicate a lesser error in predicting the values.

Training data			
Hidden Neurons	 Hidden unit 1	 Hidden unit 3	 Hidden unit 5
Output Neurons	 Output unit 1		

Validation data

Hidden Neurons	 Hidden unit 1	 Hidden unit 3	 Hidden unit 5
Output Neurons	 Output unit 1		

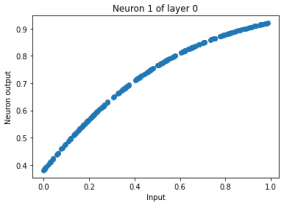
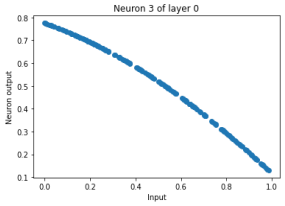
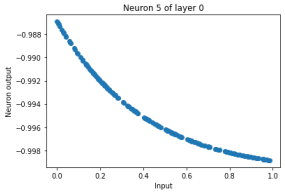
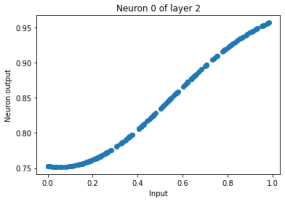
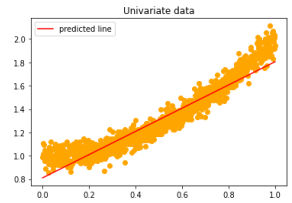
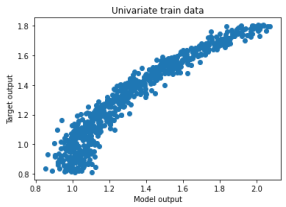
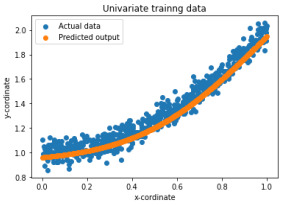
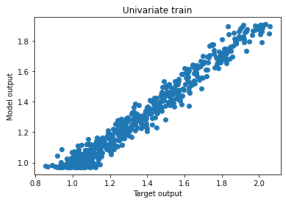
Test data			
Hidden Neurons	 Hidden unit 1	 Hidden unit 3	 Hidden unit 5
Output Neurons	 Output unit 1		

Table 3.1.4: Above tables show output plots of hidden units and output units on training, validation and test data for the best selected architecture.

Performance Parameters	Best fit curve	MSE	Model vs Target output
Single Neuron Model		2.589723	
Fully Connected Neural Network		1.070124	

Single neuron model tried to fit a straight line on the dataset which is not a good approximation of the underlying distribution. FCNN estimated a more accurate curve for the training data. The target vs model output is more linear for FCNN as compared to single neuron, indicating that predicted outputs are much closer to the target output.

3.2. Regression using two hidden layers on bivariate data using FCNN

The training algorithm is the same as we used in the single layer model. Only the backpropagation operations are changed as follows:

Backward Computation

a) Compute instantaneous error, $E_n = \frac{1}{2}(y_n - s_n)^2$

b) Update weights between hidden layer 2 and output layer as

$$w_{lk}^{(o)} = w_{lk}^{(o)} + \Delta w_{lk}^{(o)}$$

where $\Delta w_{lk}^{(o)} = -\eta \frac{\partial E_n}{\partial w_{lk}^{(o)}} = \eta(y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} h_{nl}$

Here, $l = 0, 1, 2, \ldots, L$ and $k = 1, 2, \ldots, K$

c) Update weights between hidden layer 1 and hidden layer 2 as

$$w_{jl}^{(h2)} = w_{jl}^{(h2)} + \Delta w_{jl}^{(h2)}$$

where

$$\Delta w_{jl}^{(h2)} = -\eta \frac{\partial E_n}{\partial w_{jl}^{(h2)}} = \eta \left[\sum_{k=1}^{K} (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} w_{lk}^{(o)} \frac{\partial g(a_{nl}^{(h2)})}{\partial a_{nl}^{(h2)}} \right] h_{nj}$$

Here, $j = 0, 1, 2, \ldots, d$ and $k = 1, 2, \ldots, K$

d) Update weights between input layer and hidden layer 1 as

$$w_{ij}^{(h1)} = w_{ij}^{(h1)} + \Delta w_{ij}^{(h1)}$$

where

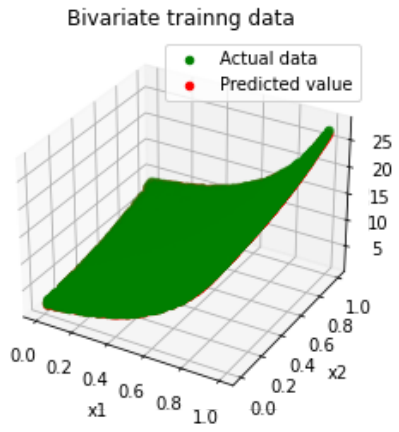
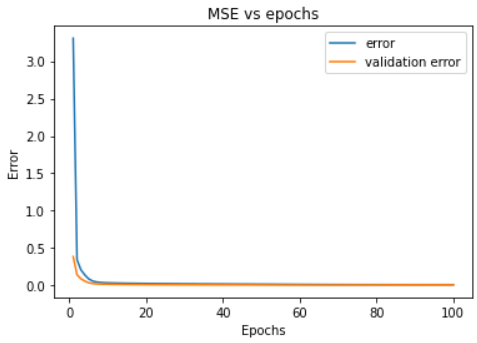
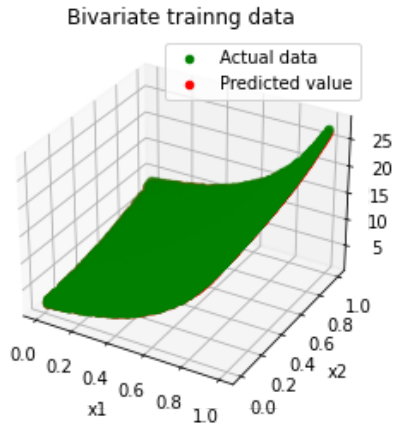
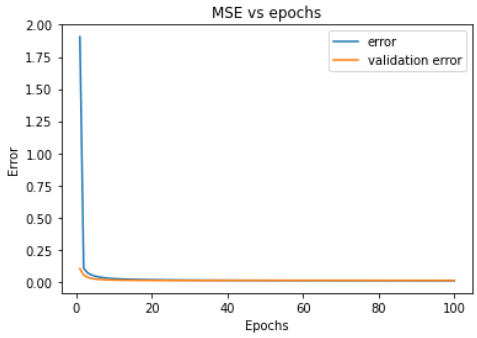
$$\Delta w_{ij}^{(h1)} = -\eta \frac{\partial E_n}{\partial w_{ij}^{(h1)}} = \eta \left[\sum_{l=1}^{L} \left[\sum_{k=1}^{K} (y_n - s_n) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}} w_{lk}^{(o)} \frac{\partial g(a_{nl}^{(h2)})}{\partial a_{nl}^{(h2)}} \right] w_{jl}^{(h2)} \frac{\partial g(a_{nj}^{(h1)})}{\partial a_{nj}^{(h1)}} \right] x$$

Here, $j = 0, 1, 2, \ldots, d$ and $k = 1, 2, \ldots, K$

e)

We have used $f(a) = tanh(a)$ as an activation function for output neurons and $g(a) = a$ as an activation function for hidden neurons.

Results:

Cross-validation results				
Model architecture	Best fit curve	Average error vs epoch for train and validation	MSE for train	MSE for validation
1I-5H-5H-1O			0.012708825305763926	4.25181956
1I-5H-10H-1O			0.009379676466201433	4.25913988

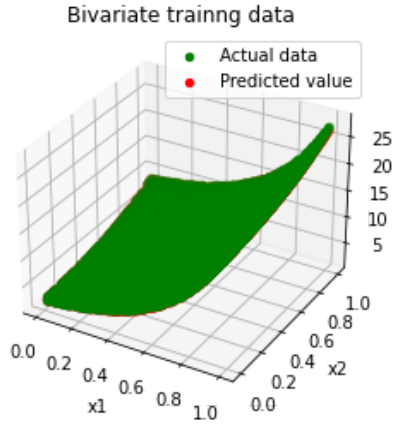
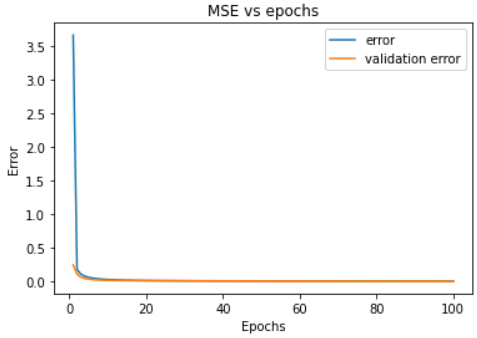
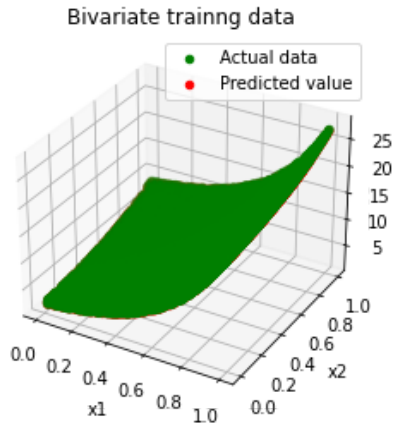
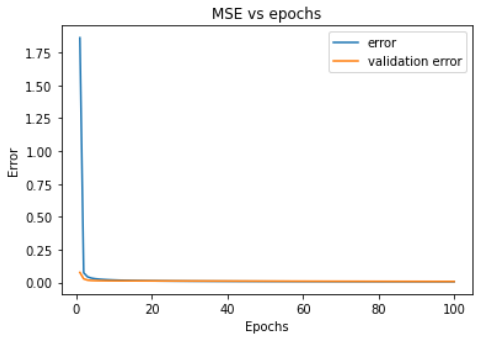
1I-10H-5H-1O			0.008384381079499676	4.20365677
1I-10H-10H-1O			0.006676333380660599	4.22987894

Table 3.2.1: Above tables show 3D output plots of hidden units and output units on training, validation and test data for the best selected architecture.

The best architecture has 10 hidden units in first layer and 5 hidden units in second layer and its test MSE is 3.97848158

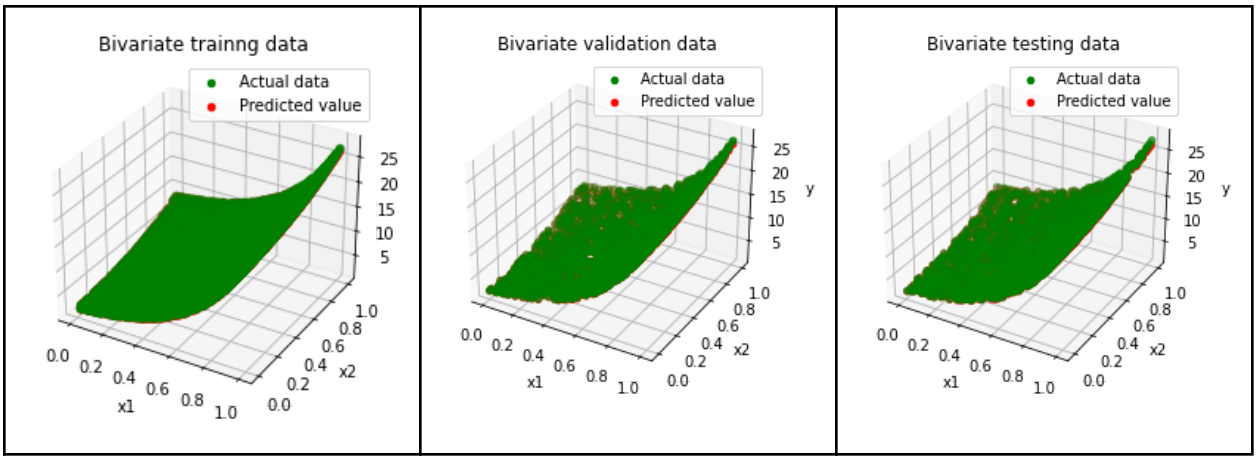


Table 3.2.2: Plots of model output and target output for training data, validation data and test data.

The approximated hyperplane (in red) has a very good fit for training, testing and the validation data which can be inferred from the above plot.

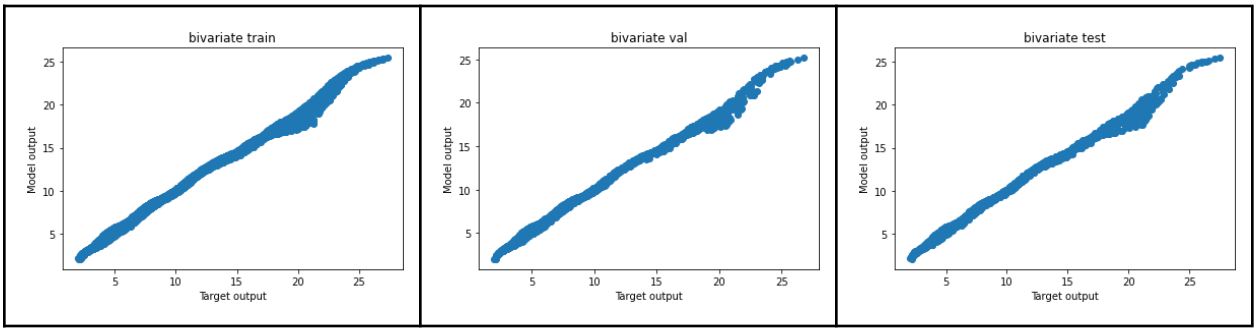
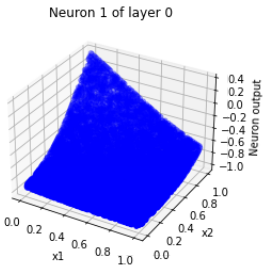
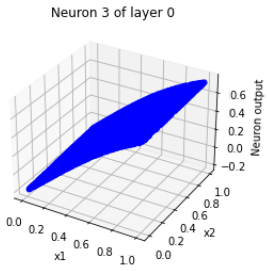
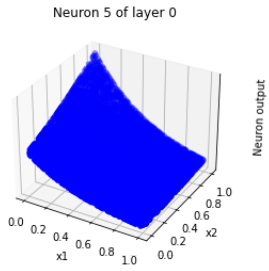
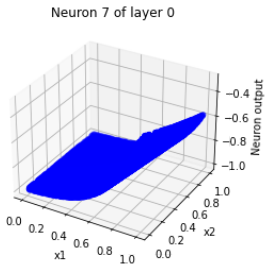
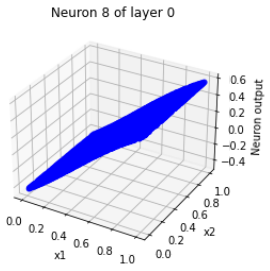
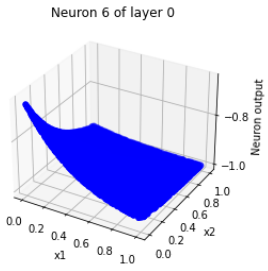
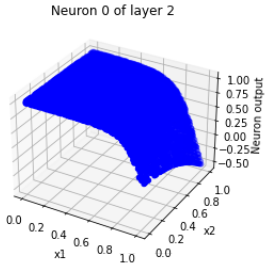
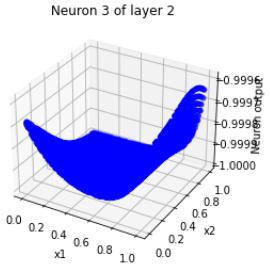
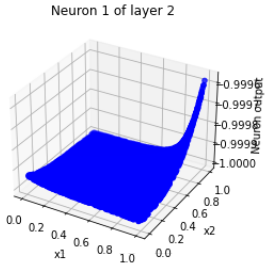
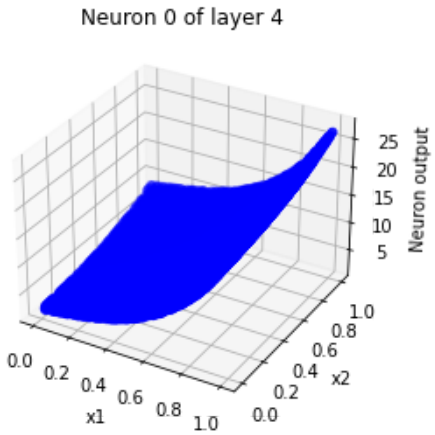
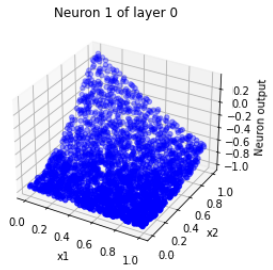
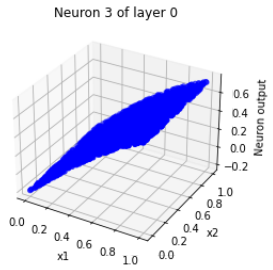
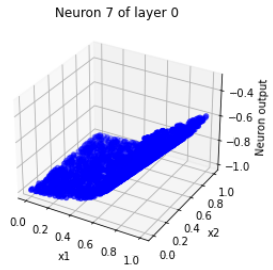
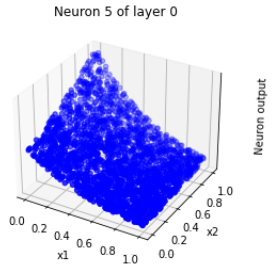
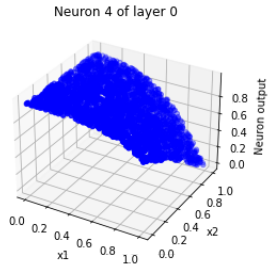
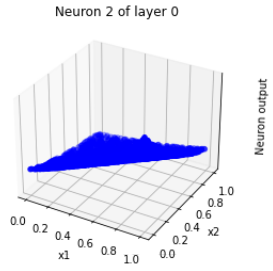
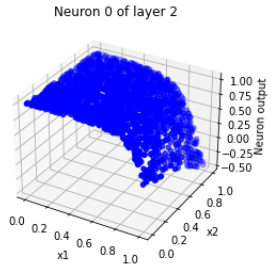
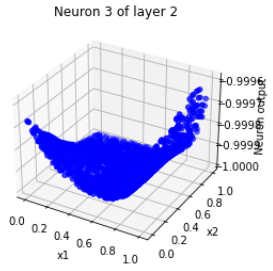
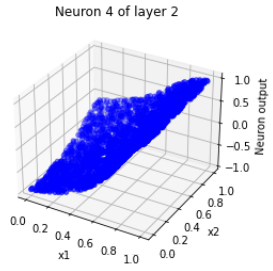
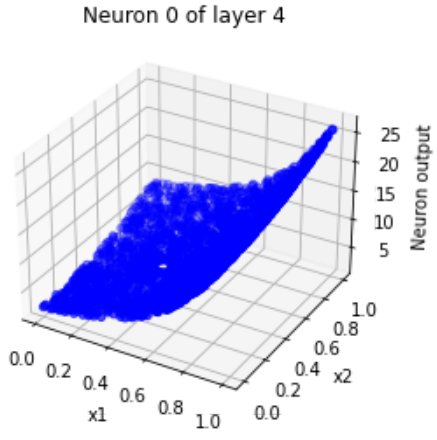


Table 3.2.3: Model vs target output for best architecture of bivariate data.

It can be observed that, target vs model output is almost a straight line which indicates that the predicted values are close to the actual values.

Training data			
Hidden layer 1 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10

	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20
Hidden layer 2 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
Output neurons	 Output neuron 1		

Validation data			
Hidden layer 1 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20
Hidden layer 2 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
Output neurons			

	Output neuron 1

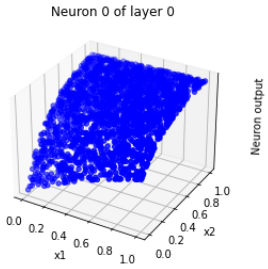
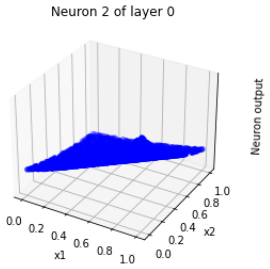
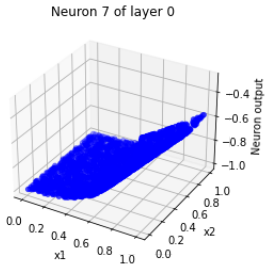
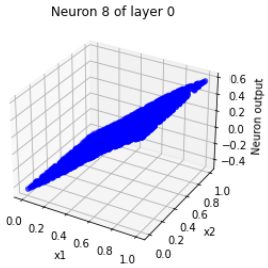
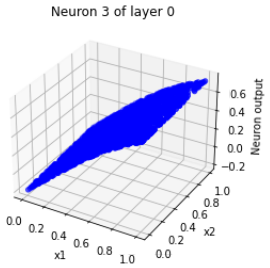
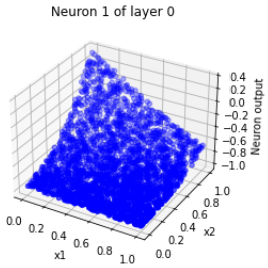
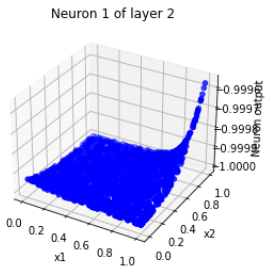
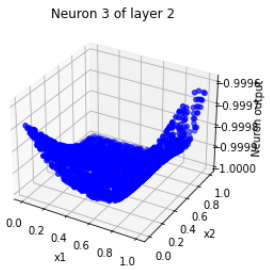
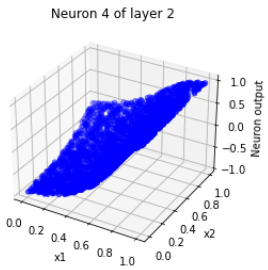
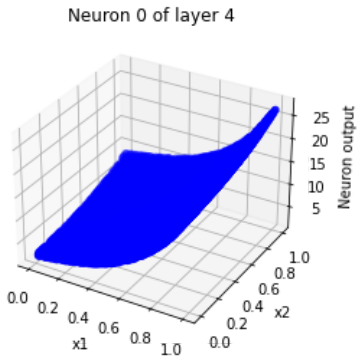
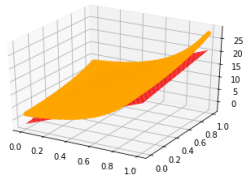
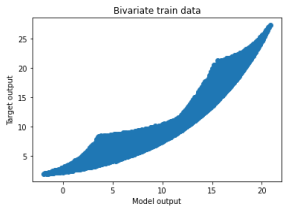
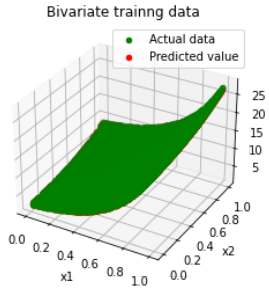
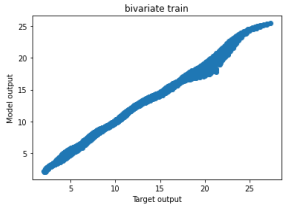
Test data			
Hidden layer 1 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
	 Hidden neuron 12	 Hidden neuron 15	 Hidden neuron 20
Hidden layer 2 neurons	 Hidden neuron 1	 Hidden neuron 5	 Hidden neuron 10
Output neurons	 Output neuron 1		

Table 3.2.4: Above tables show 3D output plots of hidden units and output units on training, validation and test data for the best selected architecture.

Performance Parameters	Best fit curve	MSE	Model vs Target output
Single Neuron Model		7.56894537	
Fully Connected Neural Network		3.97848158	

The difference in performance of both the models can be easily visualized by looking at the Best fit curve. The Single neuron model tries to fit a straight hyperplane which doenot approximates well the non-linear dataset. However the FCNN is able to fit the actual data distribution. This behavior can also be visualized in the target vs model output plot which is more linear in the case of FCNN.