Report on

Deep Learning and Applications (CS671)

Assignment 2

Submitted by:

GROUP - 9

Shashank Kapoor (S22022)

Shubham Patwar (T22108)

Syed Rizwan Ali Quadri (T22113)

Table of Content

Topic Page No.

Figure of Content

Assignment Problem Statement

In this assignment , the key objective is to classify linearly separable data and non linearly separable data by training the fully connected neural network having single hidden layer and multi hidden layer respectively and infer the results by changing the hyperparameters.

Similarly, a regression task also needs to be done on the data provided,univariate and bivariate on a fully connected neural network on a single hidden layer and multi hidden layer and infer the results by updating the hyperparameters.

In this assignment we use a fully connected neural network (FCNN) with a stochastic gradient descent approach to update the weights for classification tasks and regression tasks.

1. Classification tasks:

In classification tasks we have to classify the linearly separable and non linearly separable data by most optimal neural architecture.

1.1. Linearly Separable:

In Linearly separable , we have been provided with three classes, class1, class2 and class3 respectively. We need to classify with single hidden layer neural architecture and infer the results on the bases of average error, decision region plot, accuracy and neuron output.

1.1.1.Dataset:

In linearly separable data we have been provided with the three classes consisting, two features for each class respectively.

- Class 1: 500 data points
- Class 2: 500 data points
- Class 3: 500 data points

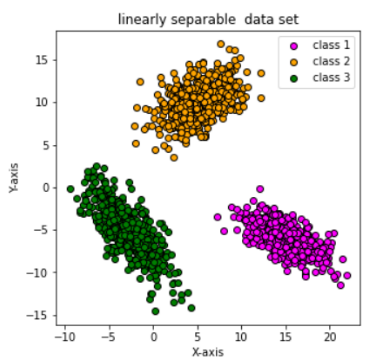
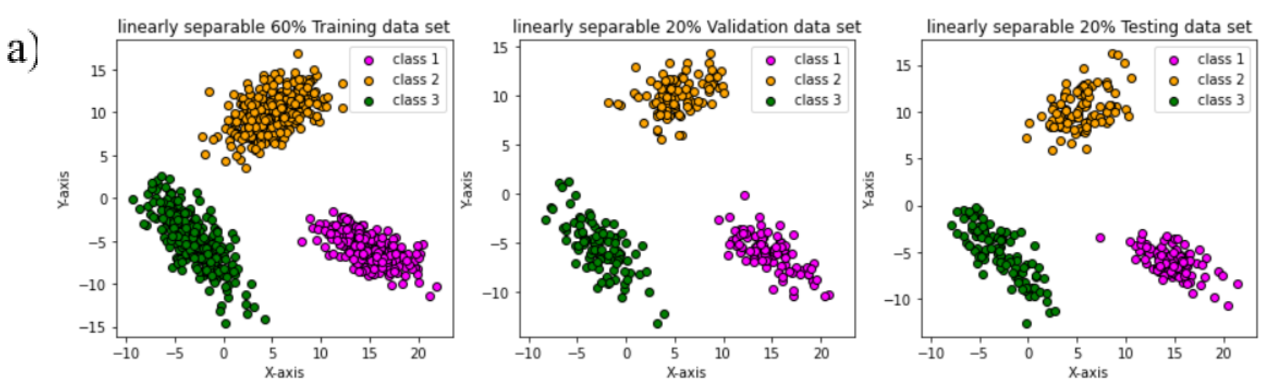


Fig. 1. Linearly separable dataset

In each class of linearly separable data we have been provided with the 500 data points in which we have to take 60% of the training data and 20% for validation and the remaining 20% is for testing .



a) **b)** **c)**

Fig. 2: Linearly separable data for a) Training 60% b) Validation 20% c) Testing 20%

1.1.2. Neural Architecture :

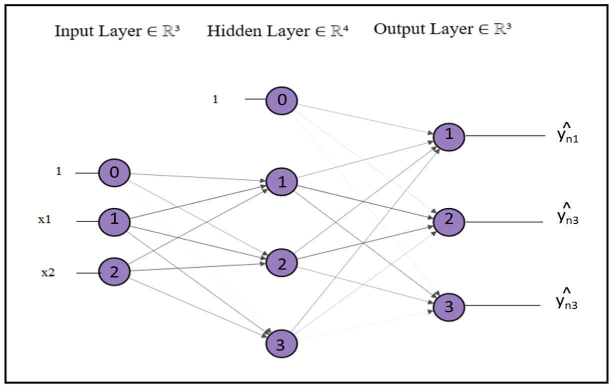


Fig. 3. FCNN with one hidden layer for linearly separable data.

- Given-training data: $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{500}$, $\mathbf{x}_n \in \mathbb{R}^2$ and $\mathbf{y}_n \in \mathbb{R}^3$
- Architecture:
  - Input layer:**2** neurons
  - Hidden layer one: **3** neurons
  - Output layer: **3** neurons

- Goal: Estimate parameters $\mathbf{W}_{ij}^{[h]}$ and $\mathbf{W}_{jk}^{[o]}$ for FCNN.
  - $\mathbf{W}_{ij}^{[h]}$ is the weight matrix of size **(2 + 1) x 3:**
    - Indicate the weights $W_{ij}^{[h]}$ in the connections between input and hidden layers.
    - $W_{ij}^{[h]}$ is the weight from i^{th} input neuron to j^{th} neuron in the hidden layer.
  - $\mathbf{W}_{jk}^{[o]}$ is the weight matrix of size **(3 + 1) x 3:**
    - Indicate the weights $W_{jk}^{[o]}$ in the connections between hidden and output layers.
    - $W_{jk}^{[o]}$ is the weight from j^{th} hidden neuron to the k^{th} neuron of the output layer.

- Steps followed:
  1. Initially the given data is labeled with the one hot notation i.e, for class1 to [1,0,0], class2 to [0,1,0] and class3 to [0,0,1], this is to represent the actual class. And the $W_{ij}^{[h]}$ and $W_{jk}^{[o]}$ with random values.
  2. Randomly chosen the training example $\mathbf{x}_n$.
  3. Forward propagation:
     Compute output of all output neuron: $\mathbf{\hat{Y}}_{nk}$
     In each forward propagation the activation value and the Logistic function is evaluated.

     Activation value = a = $\sum w^T x$ logistic = $f(a) = \frac{1}{1+e^{-\beta a}}$

4. Backward propagation:

Compute instantaneous error: $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - \hat{y}_{nk})^2$

Update weights between *hidden* and *output* layer (j = 1,2,3 and k = 1,2,3)

$\Delta w_{jk}^{(o)} = -\eta \frac{\partial E_n}{\partial w_{jk}^{(o)}}$ $w_{jk}^{(o)} = w_{jk}^{(o)} + \Delta w_{jk}^{(o)}$ $\Delta w_{jk}^{(o)} = \eta \delta_{nk}^{(o)} h_{nj}$ where $\delta_{nk}^{(o)} = (y_{nk} - \hat{y}_{nk}) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}}$

Update weights between *input* and *hidden* layer (i=1, 2 and j=1, 2, 3):

$\Delta w_{ij}^{(h)} = -\eta \frac{\partial E_n}{\partial w_{ij}^{(h)}}$ $w_{ij}^{(h)} = w_{ij}^{(h)} + \Delta w_{ij}^{(h)}$ $\Delta w_{ij}^{(h)} = \eta \delta_{nj}^{(h)} x_i$ where $\delta_{nj}^{(h)} = \left[\sum_{k=1}^{K} \delta_{nk}^{(o)} w_{jk}^{(o)}\right] \frac{\partial g(a_{nj})}{\partial a_{nj}}$

5. Repeat the steps 2 to 4 till all the training examples are presented once (Epoch)
6. Compute the average error: $E_{av} = \frac{1}{N}\sum_{n=1}^{N} E_n$
7. Now repeat steps 2 to 6 till the convergence criterion is satisfied.

1.1.3. Training:

In training we have used a sigmoidal activation function for each neuron to get the predicted output and thus calculate the instantaneous error which further helps in updating the weights in every iteration and average error in each epoch and also to backpropagate the updated weights in the network.

Parameters :

- No. of epochs : 100
- learning Factor : 0.05
- Training Data : 60% (900 data points)
- Validation Data: 20% (300 data points)

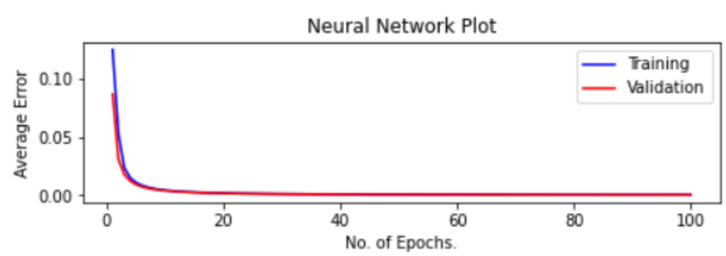


Fig. 4. Average Error vs epochs for training model of linear separable data

Observation: It is clearly seen that the model converges in 100 epochs and here average error is monotonically decreasing from that we can infer that there is a rapid change in the weights to the optimal weights.

1.1.4. Testing:

- **Decision Region Plot:**
  After the training has been done and get the optimal weights, we classify the training data into different class region and also form the decision boundary between the classes from the trained model.

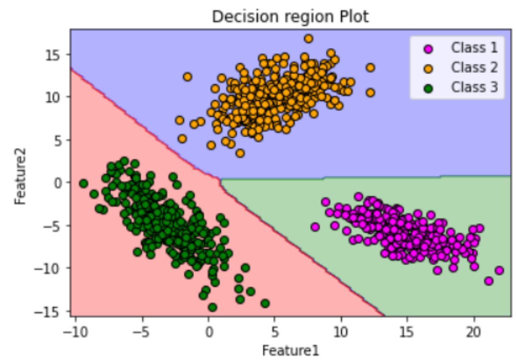


Fig.5. Decision region plot for linearly separable classes

Observation: It is clearly seen that our model is classifying the dataset into different classes and also forming the boundary region between the classes.

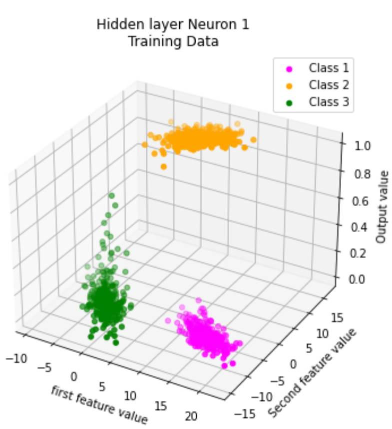
- **Confusion matrix :**
  In testing of the linearly separable data with the current neural architecture, It is found that we are getting accuracy of 100% ,recall of 100% and F1 score of 100%.

```
Confusion matrix:
[[100   0   0]
 [  0 100   0]
 [  0   0 100]]
Acccuracy: 1.0
Recall: 1.0
f1-score: 1.0
```

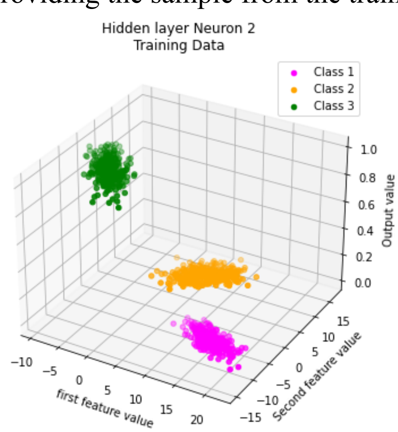
Fig.6. Confusion matrix for linearly separable classes

Observation: It is seen that our model is classifying the dataset into different classes and giving the accurate result with the ground truth value and hence giving 100% accuracy, similar we are getting 100% for recall and F1 score for all the classes in the dataset.

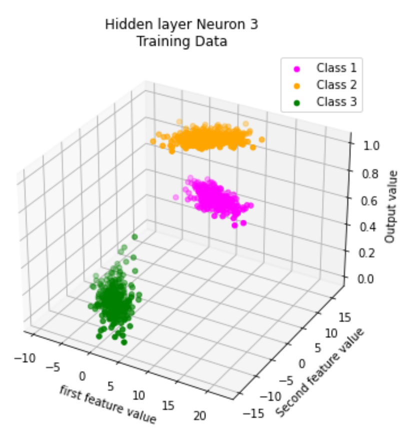
- **Hidden layer Neuron:**
  We have 3 neurons in the hidden layer for the classification of the linearly separable data. output is shown for each neuron providing the sample from the trained dataset.



a)



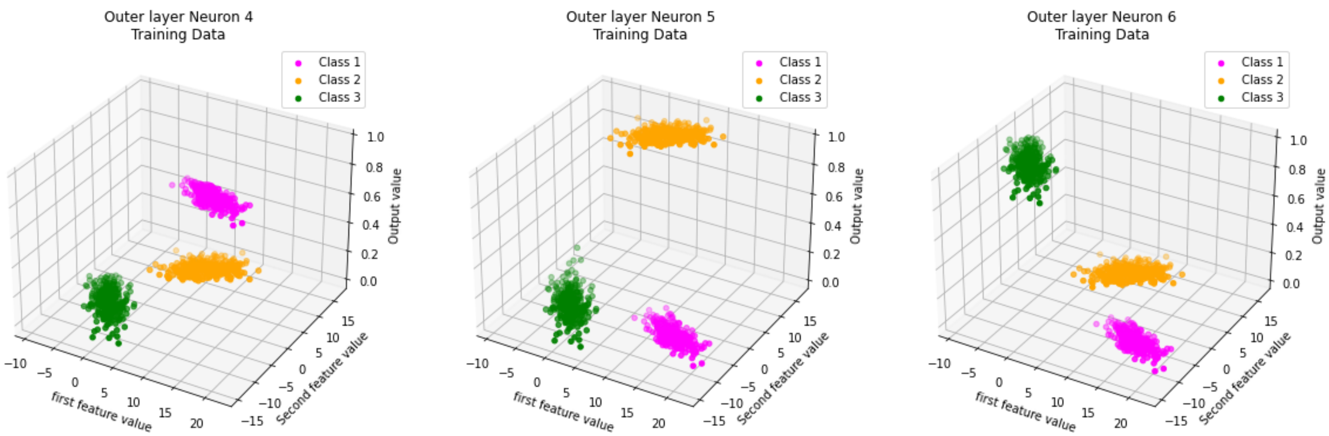
b)



c)

Fig.7. Classification of the class1 , class2 and class 3 by a) neuron 1 b) neuron 2 c) neuron 3

- **Output layer Neuron:**
  We have 3 neurons in the outer layer corresponding to each class for the classification of the linearly separable data. output is shown for each neuron providing the sample from the trained dataset.

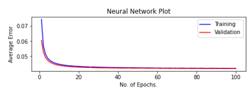
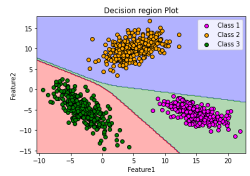
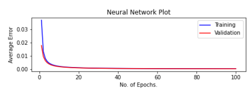
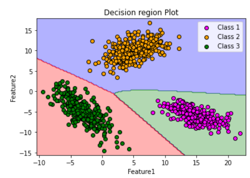
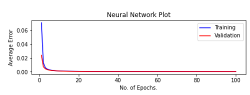
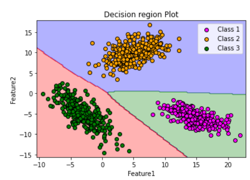
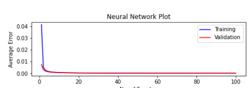
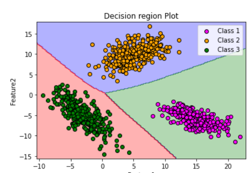


a) b) c)

Fig.8. Classification of the class1 , class2 and class 3 by a) neuron 4 b) neuron 5 c) neuron 6

1.1.5. Other Architecture :

Hyper-Parameters: Epoch = 100, Learning Rate = 0.05

Architecture	**Average Error Plot**	**Decision Region Plot**	**Confusion Matrix**	**Observation**
-Input layer:**2** node -H1 one: **2** node -output layer: **3** node			Confusion matrix: [[99 1 0] [0 100 0] [0 0 100]] Acccuracy: 0.9966666666666667 Recall: 0.9966666666666667 f1-score: 0.9983305509181971	As the number of neurons in the hidden layer increases, 1. In less number of epochs the average converges to **zero.** 2. The initial validation average error compared training has a significant change. 3. Its efficiency to classify also increases.
-Input layer:**2** node -H1 one: **6** node -output layer: **3** node			Confusion matrix: [[100 0 0] [0 100 0] [0 0 100]] Acccuracy: 1.0 Recall: 1.0 f1-score: 1.0	
-Input layer:**2** node -H1 one: **10** node -output layer: **3** node			Confusion matrix: [[100 0 0] [0 100 0] [0 0 100]] Acccuracy: 1.0 Recall: 1.0 f1-score: 1.0	
-Input layer:**2** node -H1 one: **14** node -output layer: **3** node			Confusion matrix: [[100 0 0] [0 100 0] [0 0 100]] Acccuracy: 1.0 Recall: 1.0 f1-score: 1.0	

1.2. Non -Linearly Separable:

In Non- Linearly separable , we have been provided with three classes, class1, class2 and class3 respectively. We need to classify with two hidden layer neural architecture and infer the results on the bases of average error, decision region plot, accuracy and neuron output.

1.2.1.Dataset:

In Non linearly separable data we have been provide with the three classes consisting, two features for each class respectively.

- Class 1: 500 data points
- Class 2: 500 data points
- Class 3: 700 data points

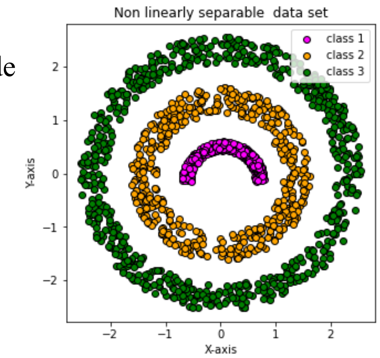
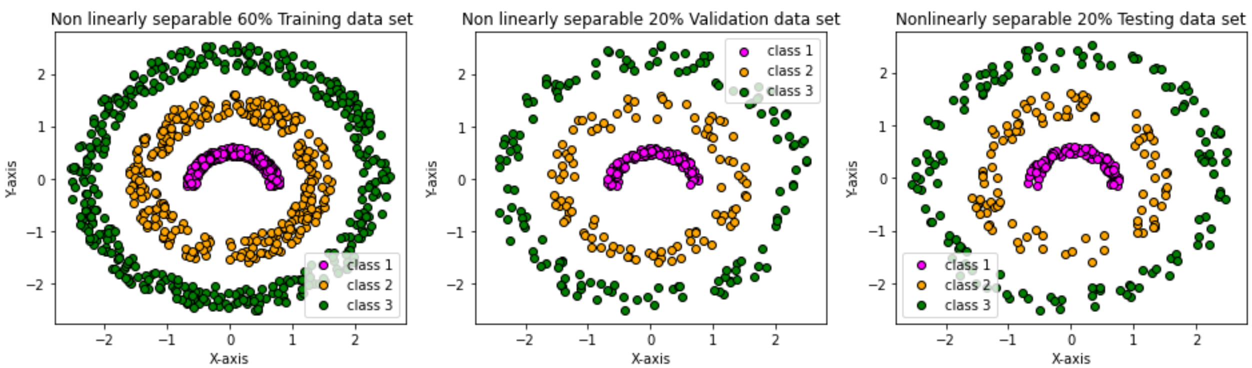


Fig.9 . Non Linearly separable dataset

In each class of Non linearly separable data we have been provided with the 500, 500 and 700 data points respectively in which we have to take 60% of the training data and 20% for validation and the remaining 20% is for testing .



a) **b)** **c)**

Fig. 10. Non Linearly separable data for a) Training 60% b) Validation 20% c) Testing 20%

1.2.2. Neural Architecture:

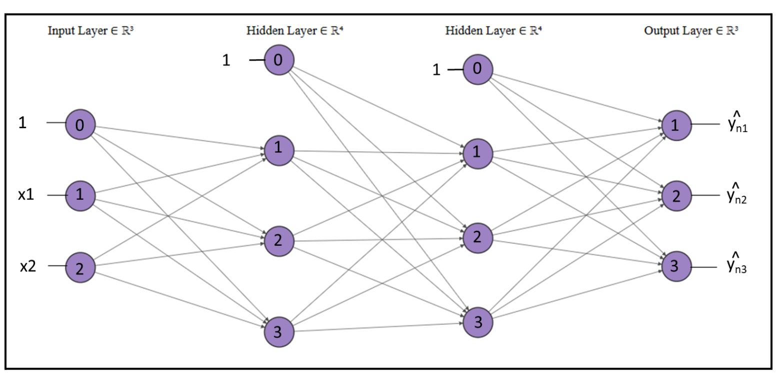


Fig. 11. FCNN with two hidden layers for Non linearly separable data.

- Given-training data: $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{500}, \ \mathbf{x}_n \in \mathbb{R}^2$ and $\mathbf{y}_n \in \mathbb{R}^3$
- Architecture:
  - Input layer: **2** neurons
  - Hidden layer one: **3** neurons
  - Hidden layer Two: **3** neuron
  - Output layer: **3** neurons

- Goal: Estimate parameters $\mathbf{W}_{ij}^{[h]}$ and $\mathbf{W}_{jk}^{[o]}$ for FCNN.
  - $\mathbf{W}_{ij}^{[h1]}$ is the weight matrix of size **(2 + 1) x 3:**
    - Indicate the weights $W_{ij}^{[h1]}$ in the connections between input and hidden layer 1.
    - $W_{ij}^{[h1]}$ is the weight from i^{th} input neuron to j^{th} neuron in the hidden layer 1.
  - $\mathbf{W}_{jl}^{[h2]}$ is the weight matrix of size **(3 + 1) x 3:**
    - Indicate the weights $W_{jl}^{[h2]}$ in the connections between hidden layer 1 and hidden layer 2.
    - $W_{jl}^{[h2]}$ is the weight from j^{th} hidden layer 1 neurons to l^{th} neurons in the hidden layer 2.

  - $\mathbf{W}_{lk}^{[o]}$ is the weight matrix of size **(3 + 1) x 3:**
    - Indicate the weights $W_{lk}^{[o]}$ in the connections between hidden layer 2 and output layer.
    - $W_{lk}^{[o]}$ is the weight from l^{th} hidden layer 2 neurons to the k^{th} neurons of the output layer.

- Steps followed:
  1. Initially the given data is labeled with the one hot notation i.e, for class1 to [1,0,0], class2 to [0,1,0] and class3 to [0,0,1], this is to represent the actual class. And the $W_{ij}^{[h1]}$, $W_{jl}^{[h2]}$ and $W_{lk}^{[o]}$ with random values.
  2. Randomly chosen the training example $\mathbf{x_n}$.
  3. Forward propagation:
     Compute output of all output neuron: $\mathbf{\hat{Y}_{nk}}$
     In each forward propagation the activation value and the Logistic function is evaluated.

     Activation value = a = $\sum w^T x$ logistic = $f(a) = \dfrac{1}{1+e^{-\beta a}}$

  4. Backward propagation:
     Compute instantaneous error: $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - \hat{y}_{nk})^2$

  5. Repeat the steps 2 to 4 till all the training examples are presented once (Epoch)

6. Compute the average error: $$E_{av} = \frac{1}{N}\sum_{n=1}^{N} E_n$$

7. Now repeat steps 2 to 6 till the convergence criterion is satisfied.

1.2.3. Training:

In training we have used for each neuron a sigmoidal activation function to get the predicted output and thus calculate the instantaneous error which further helps in updating the weights and average error for each epoch and also to backpropagate the updated weights in the network.

Parameters :

- No. of epochs : 500
- learning Factor : 0.05
- Training Data : 60% (1020 data points)
- Validation Data: 20% (340 data points)

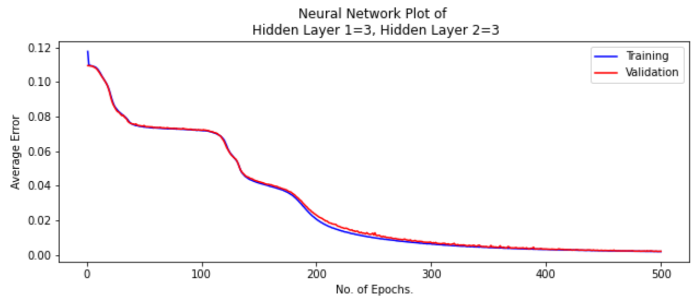


Fig. 12. Average Error vs Epochs for non linearly separable data

Observation: It is clearly seen that the model is converging to get the average error of the model on prediction tends to zero and hence successfully classifies the non linear separable data by updating the optimal weights.

1.2.4. Testing:

- **Decision Region Plot:**
  After the training has been done and get the optimal weights, we classify the training data into different class region and also form the decision boundary between the classes from the trained model.

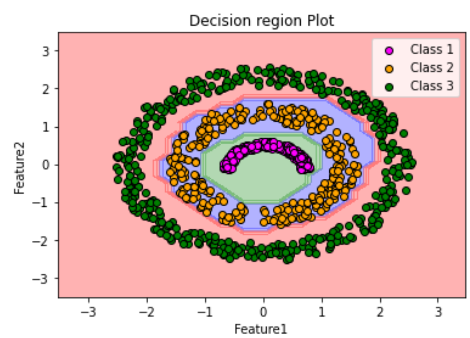


Fig.13. Decision region plot for Non linearly separable classes

Observation: It is clearly seen that our model is accurately classifying the dataset into different classes and also forming the boundary region between the classes.

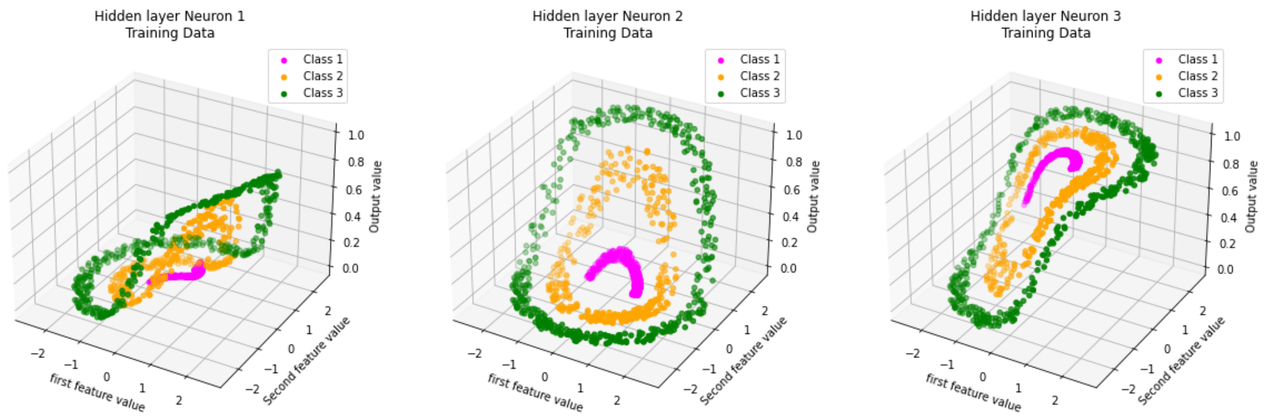
- **Confusion matrix :**
  In testing of the Non linearly separable data with the current neural architecture, It is found that we are getting accuracy of 100% ,recall of 100% and F1 score of 100%.

```
Confusion matrix:
[[100   0   0]
 [  0 100   0]
 [  0   0 140]]
Acccuracy: 1.0
Recall: 1.0
f1-score: 1.0
```

Fig.14. Confusion matrix for Non linearly separable classes

Observation: It is seen that our model is classifying the dataset into different classes and giving the accurate result with the ground truth value and hence giving 100% accuracy, similar we are getting 100% for recall and F1 score for all the classes in the dataset.

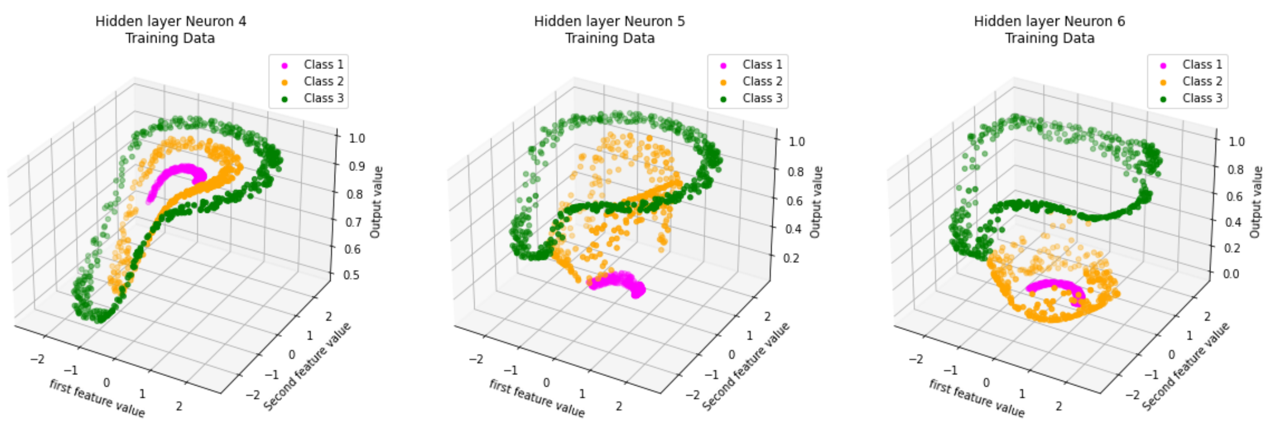
- **Hidden layer 1 Neuron:**
  We have 3 neurons in the hidden layer 1 for the classification of the Non linearly separable data. output is shown for each neuron providing the sample from the trained dataset.



a) b) c)

Fig.15. Classification of the class1 ,class2 and class 3 by a) neuron 1 b) neuron 2 c) neuron 3

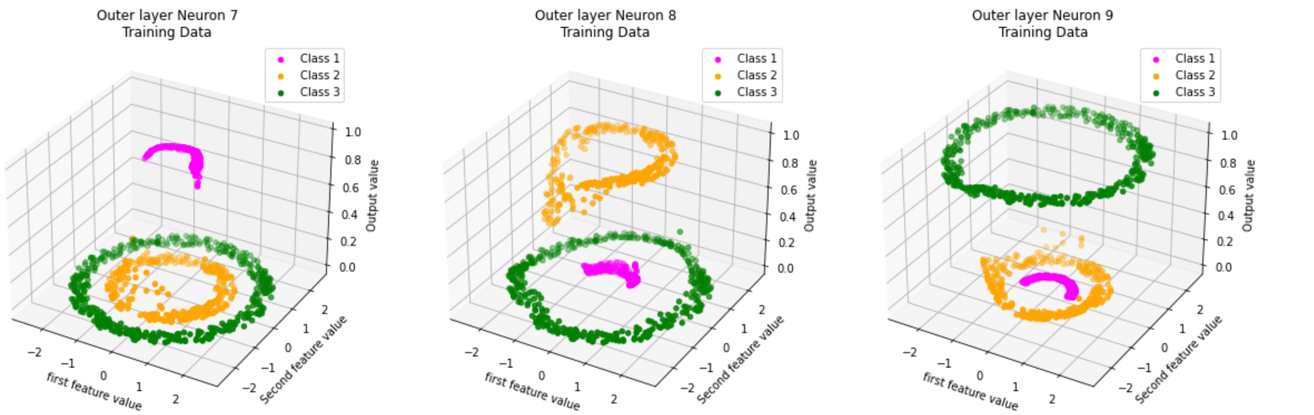
- **Hidden layer 2 Neuron:**
  We have 3 neurons in the hidden layer 2 for the classification of the Non linearly separable data. output is shown for each neuron providing the sample from the trained dataset.



a) b) c)

Fig.16. Classification of the class1 ,class2 and class 3 by a) neuron 4 b) neuron 5 c) neuron 6

- **Output layer Neuron:**
  We have 3 neurons in the outer layer corresponding to each class for the classification of the Non linearly separable data. output is shown for each neuron providing the sample from the trained dataset.

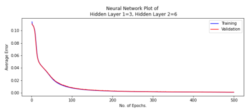
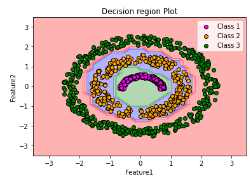


a) **b)** **c)**

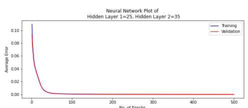
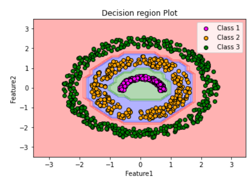
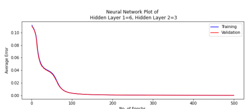
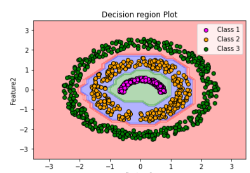
Fig.17. Classification of the class1 ,class2 and class 3 by a) neuron 7 b) neuron 8 c) neuron 9

1.2.5. Other Architecture :

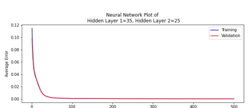
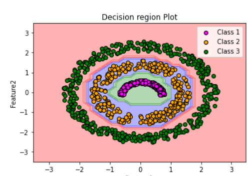
Hyper-Parameters: Epoch = 500, Learning Rate = 0.05

<table>
<tr><th>Architecture</th><th>Average Error Plot</th><th>Decision Region Plot</th><th>Confusion Matrix</th><th>Observation</th></tr>
<tr>
<td>-Input layer:2 node
-H1 one: 3 node
-H2 one: 6 node
-output layer: 3 node</td>
<td></td>
<td></td>
<td>Confusion matrix:
[[100 0 0]
[0 100 0]
[0 0 140]]
Acccuracy: 1.0
Recall: 1.0
f1-score: 1.0</td>
<td rowspan="2">Considered increasing number of neurons in respective hidden layers.

As the number of neurons increases in the hidden layer, in less number of epochs the average converges to zero.

The decision boundary becomes smoother.</td>
</tr>
<tr>
<td>-Input layer:2 node
-H1 one: 25 node
-H2 one: 35 node
-output layer: 3 node</td>
<td></td>
<td></td>
<td>Confusion matrix:
[[100 0 0]
[0 100 0]
[0 0 140]]
Acccuracy: 1.0
Recall: 1.0
f1-score: 1.0</td>
</tr>
<tr>
<td>-Input layer:2 node
-H1 one: 6 node
-H2 one: 3 node
-output layer: 3 node</td>
<td></td>
<td></td>
<td>Confusion matrix:
[[100 0 0]
[0 100 0]
[0 0 140]]
Acccuracy: 1.0
Recall: 1.0
f1-score: 1.0</td>
<td rowspan="2">Considered a decreasing number of neurons in respective hidden layers.

As the number of neurons increases in the hidden layer, in less number of epochs the average converges to zero.

The decision boundary becomes smoother.</td>
</tr>
<tr>
<td>-Input layer:2 node
-H1 one: 35 node
-H2 one: 25 node
-output layer: 3 node</td>
<td></td>
<td></td>
<td>Confusion matrix:
[[100 0 0]
[0 100 0]
[0 0 140]]
Acccuracy: 1.0
Recall: 1.0
f1-score: 1.0</td>
</tr>
</table>

2. **Regression tasks:**

In Regression tasks we have to predict the output from the trained neural architecture by giving the training dataset for the model to learn and get .

2.1. Univariate:

In Univariate , we have been provided with the single feature and the actual output. We need to predict with single hidden layer neural architecture and infer the results on the bases of average error, decision region plot, accuracy and neuron output.

2.1.1.Dataset:

In univariate data we have been provided with the feature and corresponding actual output.

- Univariate : 1000 data points

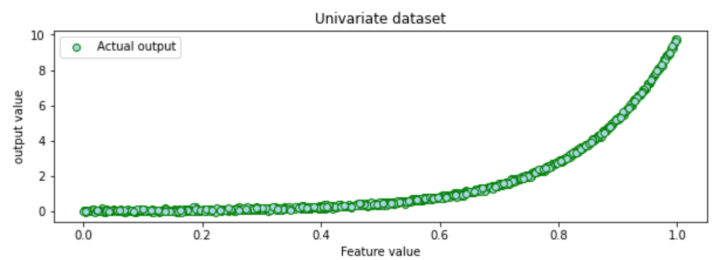
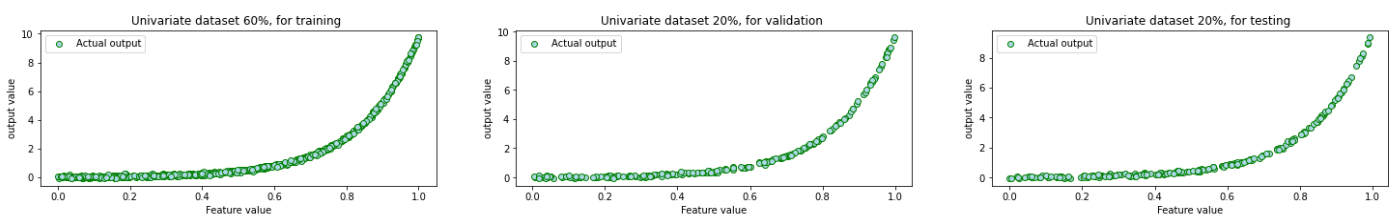


Fig.18. Univariate dataset

In a univariate dataset, we have been provided with the 1000 data points in which we have to have take 60% as the training data and 20% for validation and the remaining 20% is for testing .



a) b) c)

Fig.19 . Univariate dataset for a) Training 60% b) Validation 20% c) Testing 20%

2.1.2. Neural Architecture:

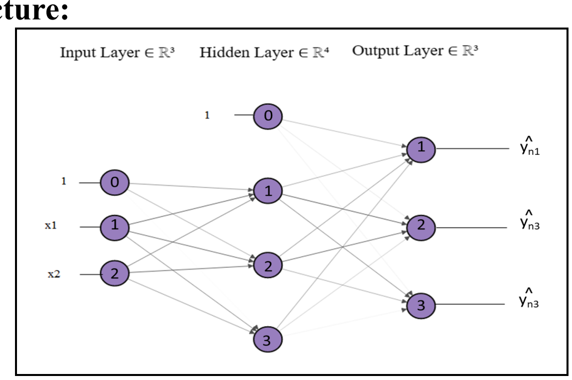


Fig. 20. FCNN with a single hidden layer for univariate data.

- Given-training data: $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{500}, \ \mathbf{x}_n \in \mathbb{R}^2$ and $\mathbf{y}_n \in \mathbb{R}^3$
- Architecture:
  - Input layer:**2** neurons
  - Hidden layer one: **3** neurons
  - Output layer: **3** neurons

- Goal: Estimate parameters $\mathbf{W}_{ij}^{[h]}$ and $\mathbf{W}_{jk}^{[o]}$ for FCNN.
  - $\mathbf{W}_{ij}^{[h]}$ is the weight matrix of size **(2 + 1) x 3:**
    - Indicate the weights $W_{ij}^{[h]}$ in the connections between input and hidden layers.
    - $W_{ij}^{[h]}$ is the weight from i^{th} input neuron to j^{th} neuron in the hidden layer.
  - $\mathbf{W}_{jk}^{[o]}$ is the weight matrix of size **(3 + 1) x 3:**
    - Indicate the weights $W_{jk}^{[o]}$ in the connections between hidden and output layers.
    - $W_{jk}^{[o]}$ is the weight from j^{th} hidden neuron to the k^{th} neuron of the output layer.

- Steps followed:
  3. Initially the given data is labeled with the one hot notation i.e, for class1 to [1,0,0], class2 to [0,1,0] and class3 to [0,0,1], this is to represent the actual class. And the $W_{ij}^{[h]}$ and $W_{jk}^{[o]}$ with random values.
  4. Randomly chosen the training example $\mathbf{x}_n$.
  3. Forward propagation:
     Compute output of all output neuron: $\mathbf{Y}^{\wedge}_{nk}$
     In each forward propagation the activation value and the Logistic function is evaluated.

     Activation value = a = $\sum w^T x$ logistic = $f(a) = \dfrac{1}{1+e^{-\beta a}}$

  4. Backward propagation:
     Compute instantaneous error: $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - \hat{y}_{nk})^2$

     Update weights between *hidden* and *output* layer (j = 1,2,3 and k = 1,2,3)

$$\Delta w_{jk}^{(o)} = -\eta \frac{\partial E_n}{\partial w_{jk}^{(o)}} \qquad w_{jk}^{(o)} = w_{jk}^{(o)} + \Delta w_{jk}^{(o)} \qquad \Delta w_{jk}^{(o)} = \eta \delta_{nk}^{(o)} h_{nj} \quad \text{where } \delta_{nk}^{(o)} = (y_{nk} - \hat{y}_{nk}) \frac{\partial f(a_{nk}^{(o)})}{\partial a_{nk}^{(o)}}$$

     Update weights between *input* and *hidden* layer (i=1, 2 and j=1, 2, 3):

$$\Delta w_{ij}^{(h)} = -\eta \frac{\partial E_n}{\partial w_{ij}^{(h)}} \qquad w_{ij}^{(h)} = w_{ij}^{(h)} + \Delta w_{ij}^{(h)} \qquad \Delta w_{ij}^{(h)} = \eta \delta_{nj}^{(h)} x_i \quad \text{where } \delta_{nj}^{(h)} = \left[\sum_{k=1}^{K} \delta_{nk}^{(o)} w_{jk}^{(o)}\right] \frac{\partial g(a_{nj})}{\partial a_{nj}}$$

  5. Repeat the steps 2 to 4 till all the training examples are presented once (Epoch)

6. Compute the average error: $E_{av} = \frac{1}{N}\sum_{n=1}^{N} E_n$

7. Now repeat steps 2 to 6 till the convergence criterion is satisfied.

2.1.3. Training:

In training we have used for each neuron a sigmoidal activation function to get the predicted output and thus calculate the instantaneous error which further helps in updating the weights and average error for each epoch and also to backpropagate the updated weights in the network.

Parameters :

- No. of epochs : 100
- learning Factor : 0.05
- Training Data : 60% (900 data points)
- Validation Data: 20% (300 data points)

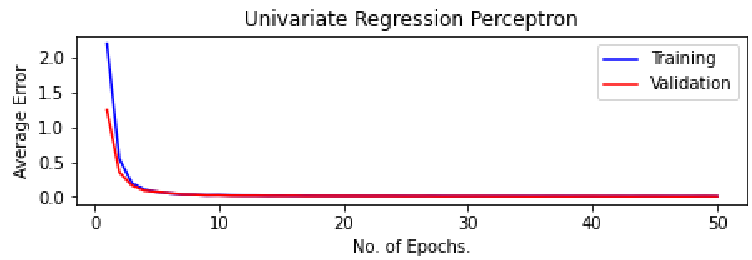


Fig. 21.Average Error vs Epochs for univariate dataset

Observation: It is clearly seen that the model is converging to get the average error of the model on prediction tends to zero and hence successfully predicts the model output data by updating the optimal weights.

2.1.4. Testing:

- **Mean Squared Error:**
  After the training has been done and get the optimal weights, we calculated mean squared error on the optimal weights for training validation and testing dataset.

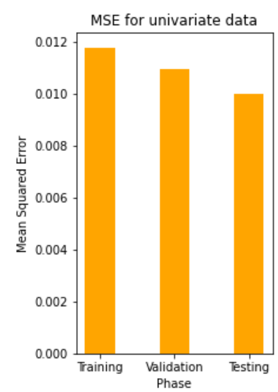
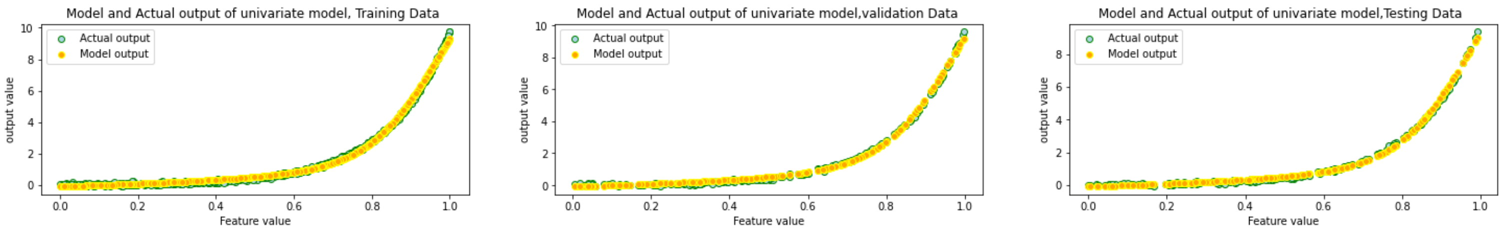


Fig.22. Mean Squared error computed for Training, Validation and Testing Phase

Observation: It is observed that the testing have the minimum Mean squared error then the training and validation set which is correct as in training the model try to learn and make error and on that error it updates optimal weight and after training is done, model has learn there is minimum chance now to make the model error therefore it has less MSE in testing.

- **Model vs Actual output**

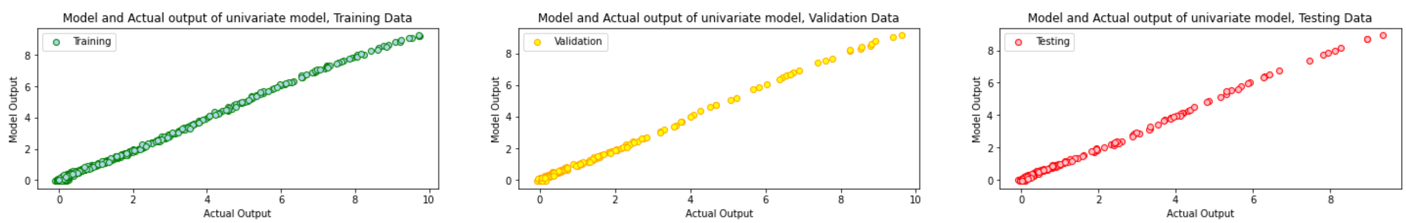
  After the model is trained on the training dataset,we computed the optimal weights and on that weight we predicted the model output for the corresponding training ,validation and testing dataset.



a) b) c)

Fig.23. Features vs output for both model and actual output for a) Training b)Validation c) Testing

Observation: It is clearly seen that the model is accurately predicting the output value to the actual value for the corresponding Training , validation and testing dataset. Hence we can say that our model will nearly predict the actual output.



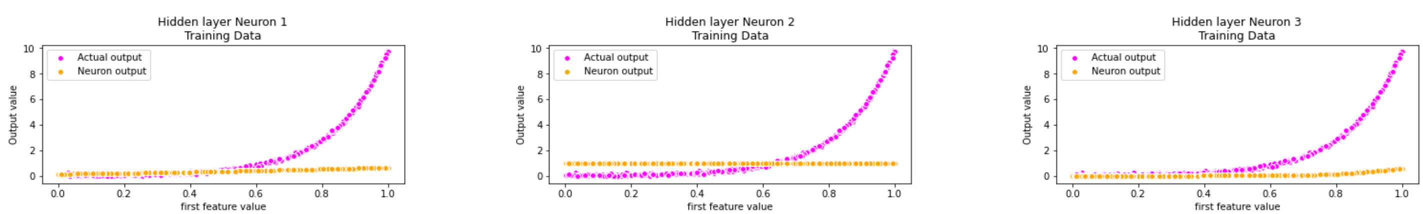
a) b) c)

Fig.24. Model vs Actual output for a) Training b)Validation and c) Testing dataset

Observation: It is seen that model output is nearly equal to the actual output, which stats that it is following (y=x) line equation and Hence our model is nearly predicting the actual output.

- **Hidden layer Neuron:**

  We have taken 3 neurons in the hidden layer 1 for the prediction of the model output. Output is shown for each neuron provided the sample is from the trained dataset.



a) b) c)

Fig.25. Model output from first hidden layer of a) neuron 1 b) neuron 2 c) neuron 3

- **Output layer Neuron:**

  We have taken single neurons in the outer layer of the network architecture for the prediction of the model output . Output is shown for each neuron provided the sample is from the trained dataset.

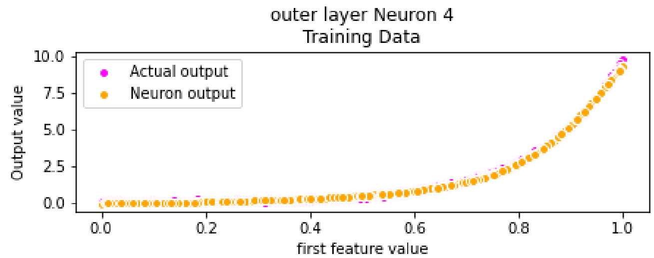
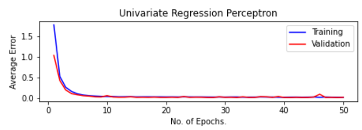
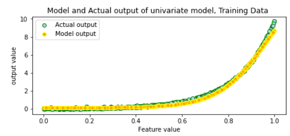
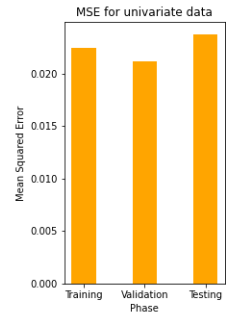
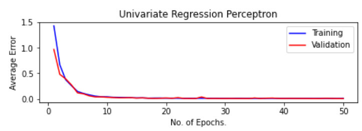
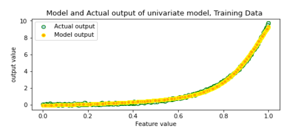
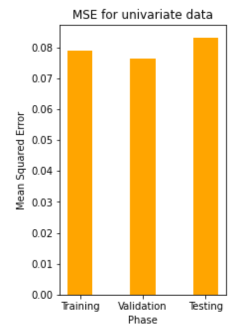
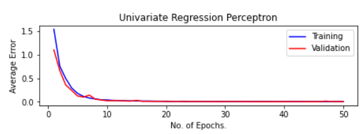
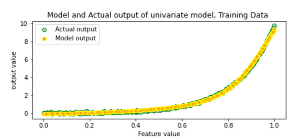
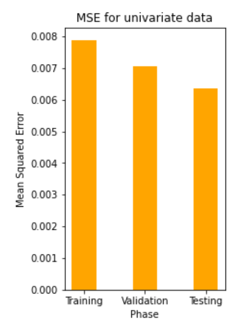
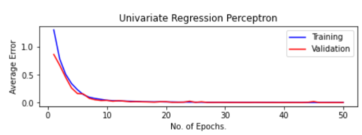
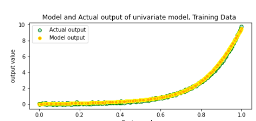
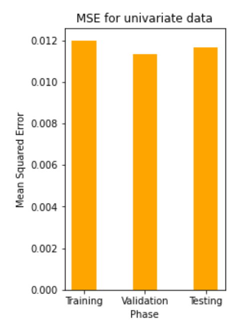


Fig.26. Model output from outer layer neuron 4

2.1.5. Other Architecture :

Hyper-Parameters: Epoch = 50, Learning Rate = 0.05

Architecture	Average Error Plot	Modal vs Actual	MSE	Observation
-Input layer:**1** node -H1 one: **2** node -output layer: **3** node				The plot of average error vs epoch remains almost similar for any number of neurons in the hidden layer and it converges to **zero**. The model is predicting almost the as of the actual values. As we vary the number of neurons in the hidden layer then we can obtain an improved **Mean square error** plot. we can see the variation in MSE plot
-Input layer:**1** node -H1 one: **6** node -output layer: **3** node				
-Input layer:**1** node -H1 one: **10** node -output layer: **3** node				
-Input layer:**1** node -H1 one: **14** node -output layer: **3** node				

2.2. Bivariate:

In Bivariate , we have been provided with the two features and the actual output. We need to predict with multi hidden layer neural architecture and infer the results on the bases of average error, decision region plot, accuracy and neuron output.

2.2.1.Dataset:

In bivariate data we have been provided with the feature and corresponding actual output.

- Bivariate : 10200 data points

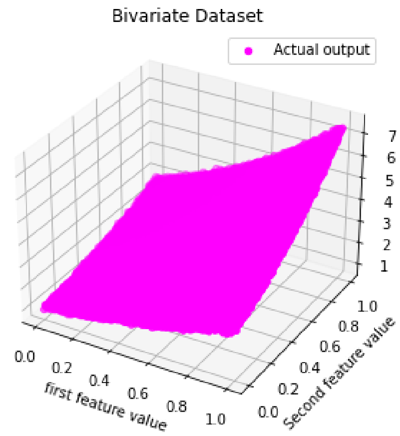


Fig.27. Bivariate dataset

In a bivariate dataset, we have been provided with the 10200 data points in which we have to have taken 60% as the training data and 20% for validation and the remaining 20% is for testing .

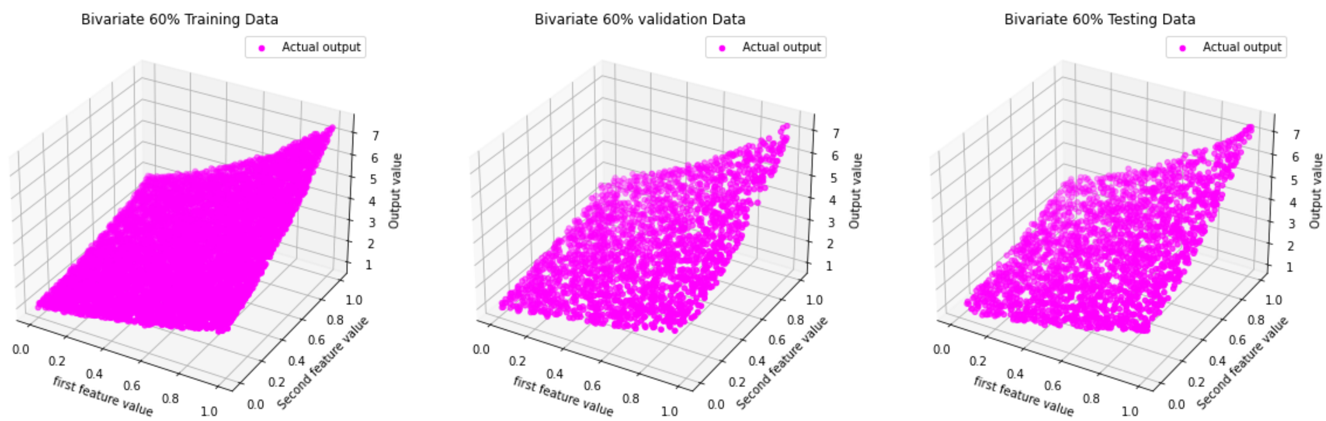


Fig.28 . Bivariate dataset for a) Training 60% b) Validation 20% c) Testing 20%

2.1.2. Neural Architecture:

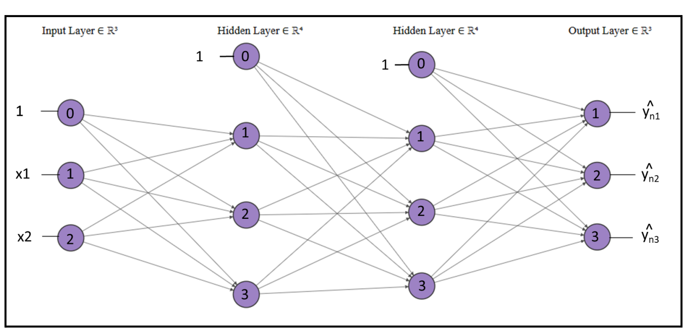


Fig. 29. FCNN with two hidden layers for Bivariate data.

- Given-training data:
- Architecture:
    - Input layer:**2** neurons
    - Hidden layer one: **3** neurons
    - Hidden layer Two: **3** neuron
    - Output layer: **3** neurons

- Goal: Estimate parameters $\mathbf{W_{ij}}^{[h]}$ and $\mathbf{W_{jk}}^{[o]}$ for FCNN.
    - $\mathbf{W_{ij}}^{[h1]}$ is the weight matrix of size **(2 + 1) x 3:**
        - Indicate the weights $W_{ij}^{[h1]}$ in the connections between input and hidden layer 1.
        - $W_{ij}^{[h1]}$ is the weight from i^{th} input neuron to j^{th} neuron in the hidden layer 1.
    - $\mathbf{W_{jl}}^{[h2]}$ is the weight matrix of size **(3 + 1) x 3:**
        - Indicate the weights $W_{jl}^{[h2]}$ in the connections between hidden layer 1 and hidden layer 2.
        - $W_{jl}^{[h2]}$ is the weight from j^{th} hidden layer 1 neurons to l^{th} neurons in the hidden layer 2.

    - $\mathbf{W_{lk}}^{[o]}$ is the weight matrix of size **(3 + 1) x 3:**
        - Indicate the weights $W_{lk}^{[o]}$ in the connections between hidden layer 2 and output layer.
        - $W_{lk}^{[o]}$ is the weight from l^{th} hidden layer 2 neurons to the k^{th} neurons of the output layer.

- Steps followed:
    3. Initially the given data is labeled with the one hot notation i.e, for class1 to [1,0,0], class2 to [0,1,0] and class3 to [0,0,1], this is to represent the actual class. And the $W_{ij}^{[h1]}$, $W_{jl}^{[h2]}$and $W_{lk}^{[o]}$ with random values.
    4. Randomly chosen the training example $\mathbf{x_n}$.
    3. Forward propagation:
       Compute output of all output neuron: $\mathbf{Y^{\wedge}_{nk}}$
       In each forward propagation the activation value and the Logistic function is evaluated.

       Activation value = a = $\sum w^T x$ logistic = $f(a) = \dfrac{1}{1+e^{-\beta a}}$

    4. Backward propagation:
       Compute instantaneous error: $E_n = \frac{1}{2}\sum_{k=1}^{K}(y_{nk} - \hat{y}_{nk})^2$

    5. Repeat the steps 2 to 4 till all the training examples are presented once (Epoch)

    6. Compute the average error: $E_{av} = \dfrac{1}{N}\sum_{n=1}^{N} E_n$

    7. Now repeat steps 2 to 6 till the convergence criterion is satisfied.

2.1.3. Training:

In training we have used for each hidden layer neuron a sigmoidal activation function and for outer neuron linear activation function to get the predicted output and thus calculate the instantaneous error which further helps in updating the weights and average error for each epoch and also to backpropagate the updated weights in the network.

Parameters :

- No. of epochs : 50
- learning Factor : 0.05
- Training Data : 60% (6000 data points)
- Validation Data: 20% (2000 data points)

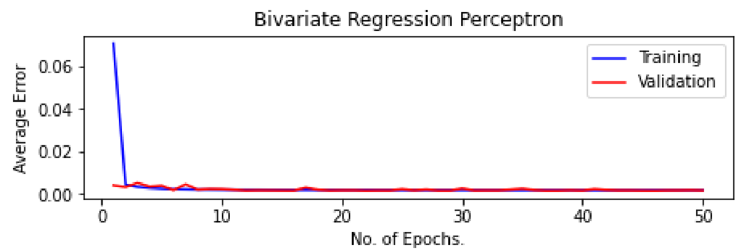


Fig. 30.Average Error vs Epochs for bivariate dataset

Observation: It is clearly seen that the model is converging to get the average error of the model on prediction tends to zero and hence successfully predicts the model output data by updating the optimal weights.

2.1.4. Testing:

- **Mean Squared Error:**
  After the training has been done and get the optimal weights, we calculated mean squared error on the optimal weights for training validation and testing dataset.

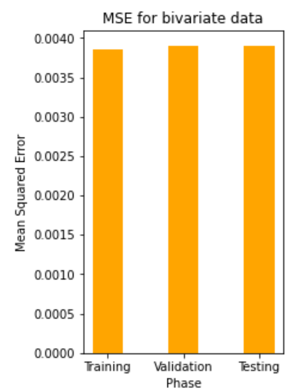
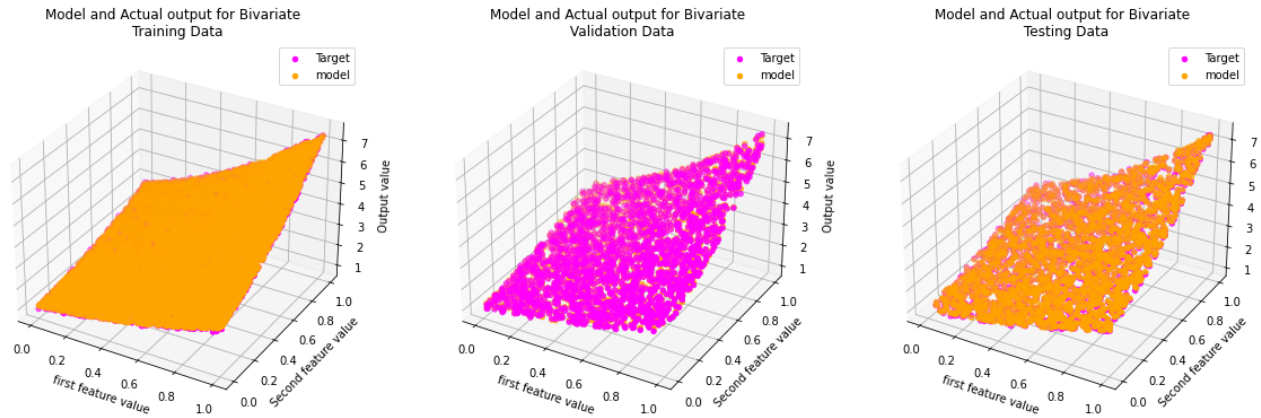


Fig.31. Mean Squared error computed for Training, Validation and Testing Phase

Observation: It is observed that the training and Testing have somewhat equal mean squared error, as testing can be less than or equal to training Mean squared error but not greater than that in any case. Which is easily seen in the above figure.

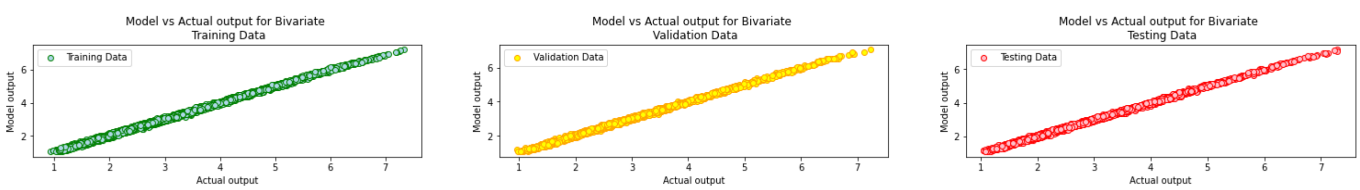
- **Model vs Actual output**
  After the model is trained on the training dataset,we computed the optimal weights and on that weight we predicted the model output for the corresponding training ,validation and testing dataset.



a) b) c)

Fig.32. Features vs output for both model and actual output for a) Training b)Validation c) Testing

Observation: It is clearly seen that the model is accurately predicting the output value to the actual value for the corresponding Training , validation and testing dataset. Hence we can say that our model will nearly predict the actual output.

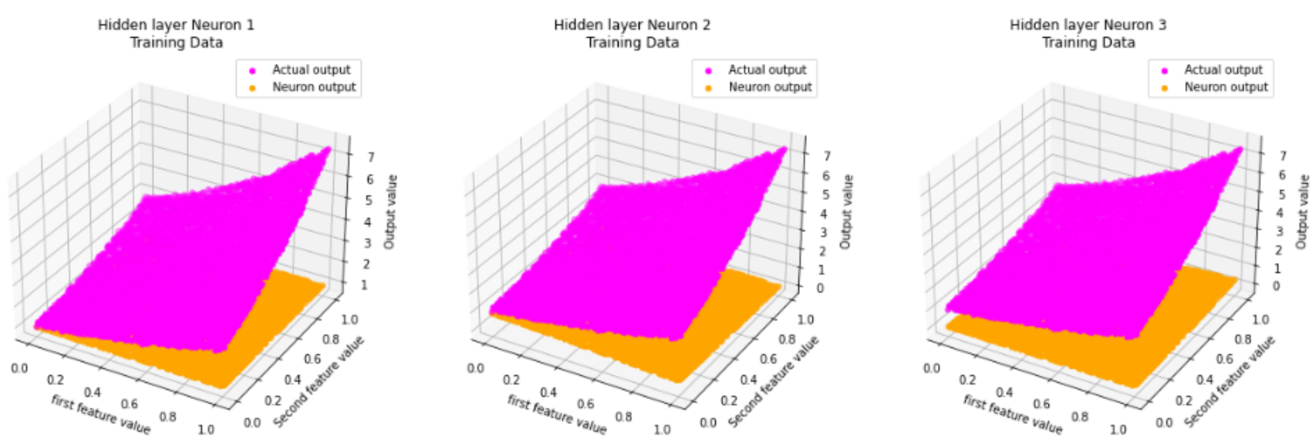


a) b) c)

Fig.33. Model vs Actual output for a) Training b)Validation and c) Testing dataset

Observation: It is seen that model output is nearly equal to the actual output, which states that it is following (y=x) line equation and Hence our model is nearly predicting the actual output.

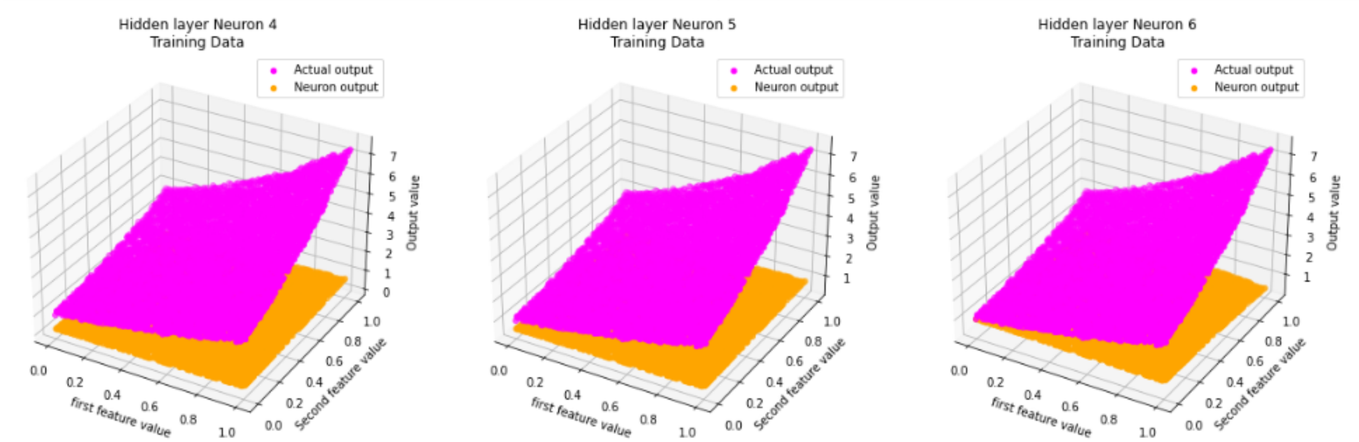
- **Hidden layer 1 Neuron:**
  We have taken 3 neurons in the hidden layer 1 for the prediction of the model output. Output is shown for each neuron provided the sample is from the trained dataset.



a) b) c)

Fig.34. Model output from first hidden layer of a) Neuron 1 b) Neuron 2 c) Neuron 3

- **Hidden layer 2 Neuron:**
  We have taken 3 neurons in the hidden layer 2 for the prediction of the model output. Output is shown for each neuron provided the sample is from the trained dataset.



a) b) c)

Fig.35. Model output from second hidden layer of a) Neuron 4 b) Neuron 5 c) Neuron 6

- **Output layer Neuron:**
  We have taken single neurons in the outer layer of the network architecture for the prediction of the model output . Output is shown for each neuron provided the sample is from the trained dataset.

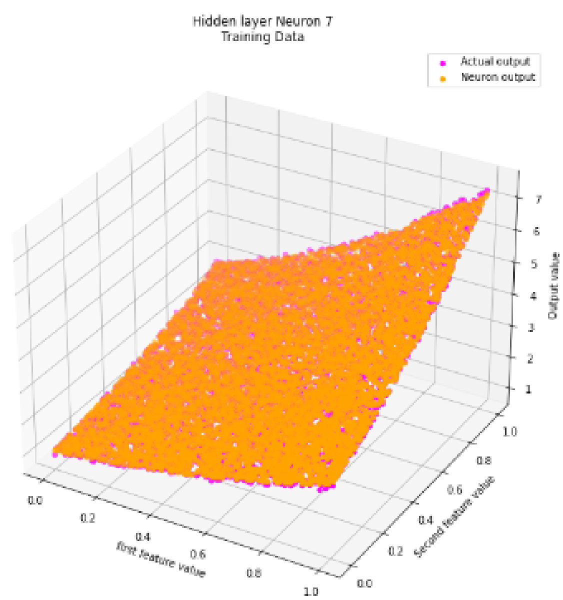
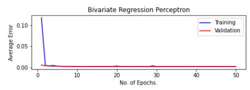
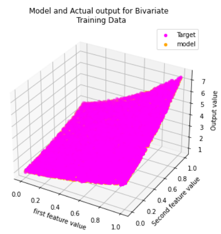
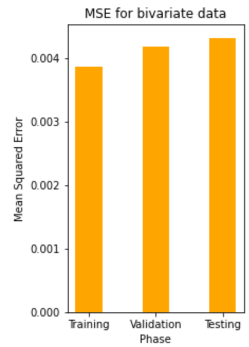


Fig.36. Model output from Outer layer of Neuron 7

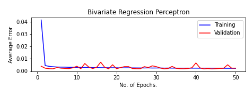
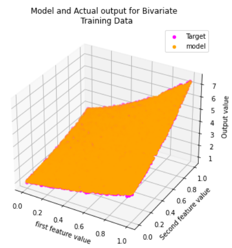
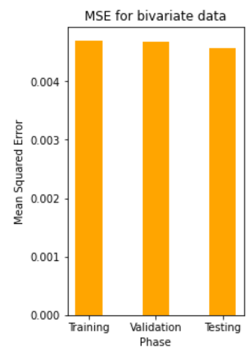
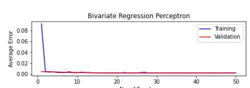
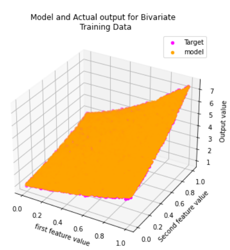
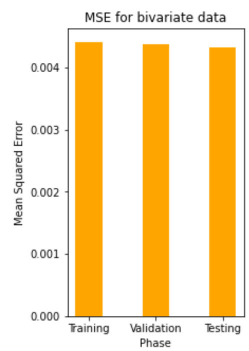
1.2.5. Other Architecture :

Hyper-Parameters: Epoch = 50, Learning Rate = 0.05

<table>
<tr><th>Architecture</th><th>Average Error Plot</th><th>Modal vs Actual</th><th>MSE</th><th>Observation</th></tr>
<tr><td>-Input layer:2 node
-H1 one: 3 node
-H2 one: 6 node
-output layer: 1 node</td><td>
</td><td>
</td><td>
</td><td rowspan="2">Considered increasing number of neurons in respective hidden layers.

As the number of neurons increases in the hidden layer, in less number of epochs the average converges to zero.

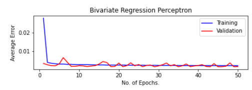
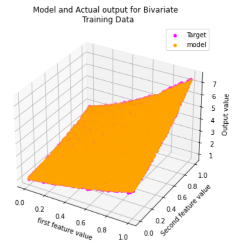
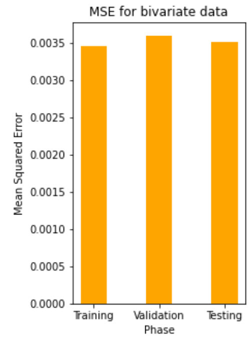
The model is predicting almost the as of the actual values.

As we vary the number of neurons in the hidden layer then we can obtain an improved Mean square error plot. we can see the variation in MSE plot</td></tr>
<tr><td>-Input layer:2 node
-H1 one: 25 node
-H2 one: 35 node
-output layer: 1 node</td><td>
</td><td>
</td><td>
</td></tr>
<tr><td>-Input layer:2 node
-H1 one: 6 node
-H2 one: 3 node
-output layer: 1 node</td><td>
</td><td>
</td><td>
</td><td>Considered a decreasing number of neurons in respective hidden layers.

As the number of neurons decreases in the hidden layer, in less number of epochs the average converges to zero.

The model is predicting almost the as of the actual values.

As we vary the number of neurons in the hidden layer then we can obtain</td></tr>
</table>

-Input layer:**2** node -H1 one: **35** node -H2 one: **25** node -output layer: **1** node				an improved **Mean square error** plot. we can see the variation in MSE plot