CS671 - Deep Learning and Applications

Assignment 1 : Learning by Perceptron and FCNN

ASSIGNMENT 1 REPORT

submitted by

Aditya Sarkar	Aaron T. Joseph	Srishti Ginjala
IIT Mandi	IIT Mandi	IIT Mandi
B19003	B19128	B19084

under the supervision of

Dr. Dileep AD

INDIAN INSTITUTE OF TECHNOLOGY, MANDI

March 2022

Abstract

In this assignment, we have observed the learning algorithms of Perceptron ad Fully Connected Neural Networks (FCNN). Perceptron with sigmoidal activation function was implemented for each of the datasets provided to group 21. We have used one-against-one approach for 3-class classification. We have implemented stochastic gradient descent method for perceptron learning algorithm. Similarly, we have worked on fully connected neural network (FCNN) for each of the above datasets. We also tried FCNN with one hidden layer for dataset 1, and used both one and two hidden layers for dataset 2. Different number of hidden nodes for both datasets were tried and we noted the observations. For FCNN, stochastic gradient descent (SGD) was implemented for backpropagation algorithm. We used squared error as instantaneous loss function.
This report is made in LaTeX

Contents

Chapter 1

Perceptron Classification

1.1 Brief description of the dataset

We were given two datasets :

1. **Dataset 1: Linearly separable classes:** 3 class, 2-dimensional linearly separable data is given. Each class has 500 data points. The scatter plot for the dataset is given below.

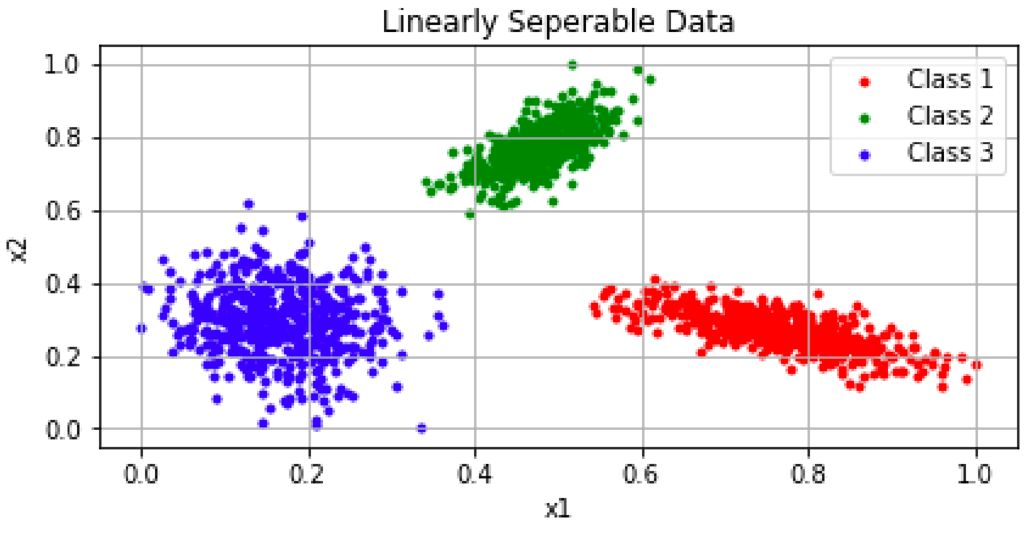


Fig. 1.1: Linearly separable data

2. **Dataset 2: Nonlinearly separable classes:** 2-dimensional data of 2 classes that are non-linearly separable. The number of examples in each class and their order was given at the beginning of each file. The scatter plot for the dataset is given below.

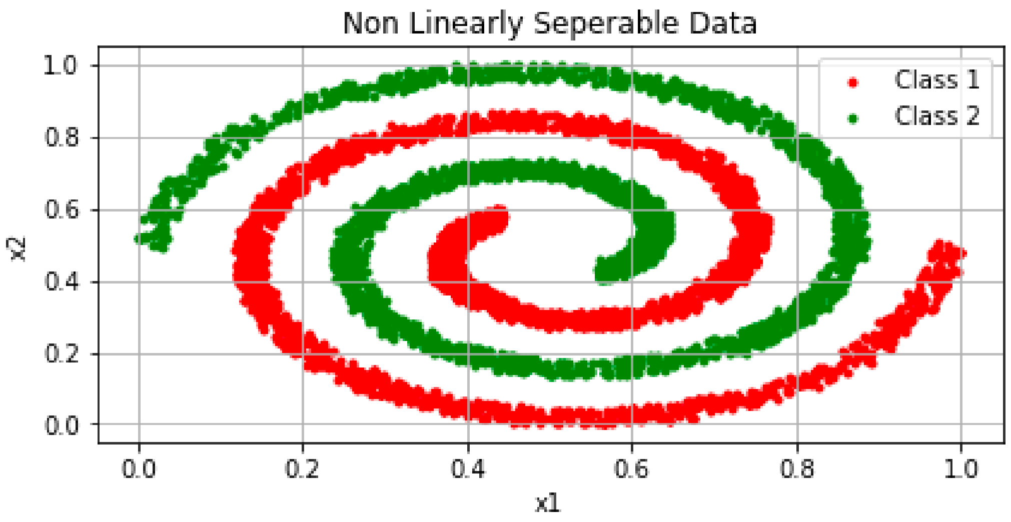


Fig. 1.2: Non-Linearly separable data

1.2 Observations

This section will discuss the observations from our experiments. It will also mention the inferences that one can take from the observations.

1.2.1 Results on Linear dataset

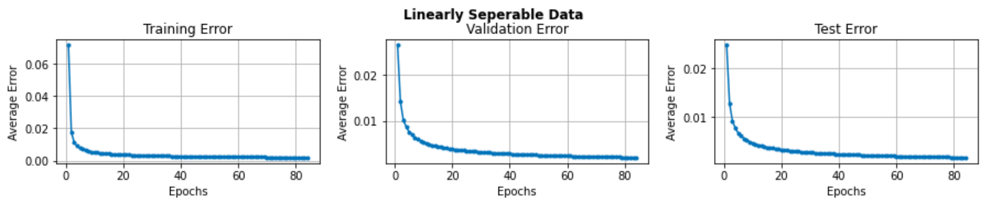


Fig. 1.3: Average error vs epochs (training)

In the above two plots (fig 1.3 and fig 1.4), we can see that the error (for both training and validation) is going down and is tending towards zero. This indicates that the model was able to learn the patterns of the data. Comparing both the plots, we can see that is drop is more steep for training data than for the validation data. This is expected because the model is being trained on the former i.e. it is bound to learn its patterns. While for validation, model is only predicting and not learning anything. From the above plots, one can conclude that the learning was generalized.

1.2.2 Decision region plots

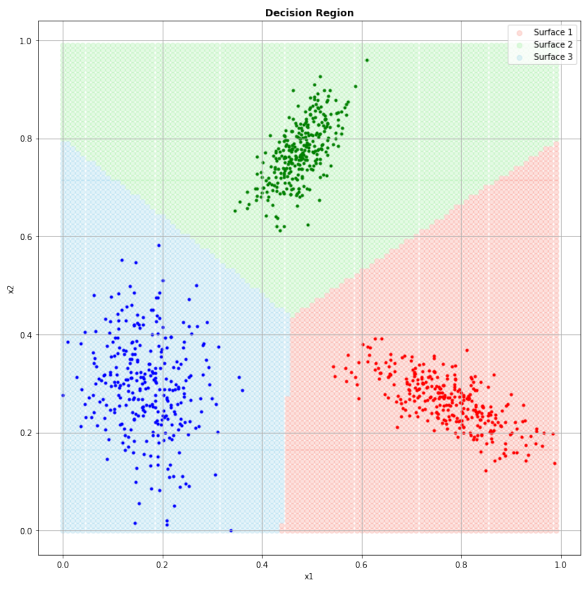


Fig. 1.4: Decision boundaries

.

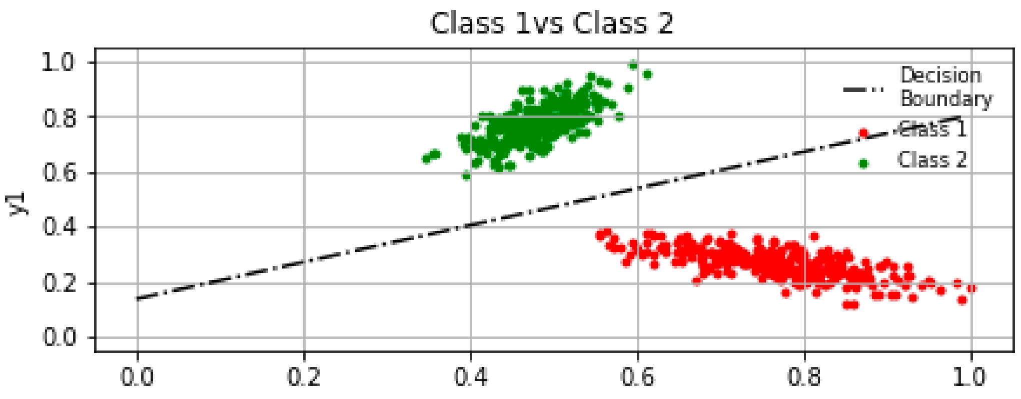


Fig. 1.5: Class 1 vs Class 2

In this report, we have implemented one-to-one approach. For the first plot, we have the decision boundary between class 1 and class 2. We can see that it is a linear boundary. This is expected, as Perceptron learning algorithm targets to create a line between the classes. Thus it is a linear classifier.

This plot shows the decision boundary between class 1 and class 3. Just like the above plot, we can see a line between different classes. This indicates that the Perceptron model is a linear classifier.

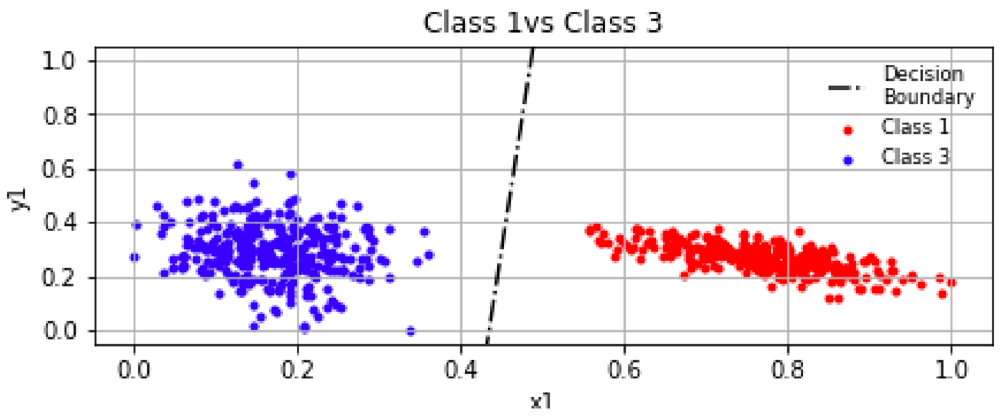


Fig. 1.6: Class 1 vs Class 3

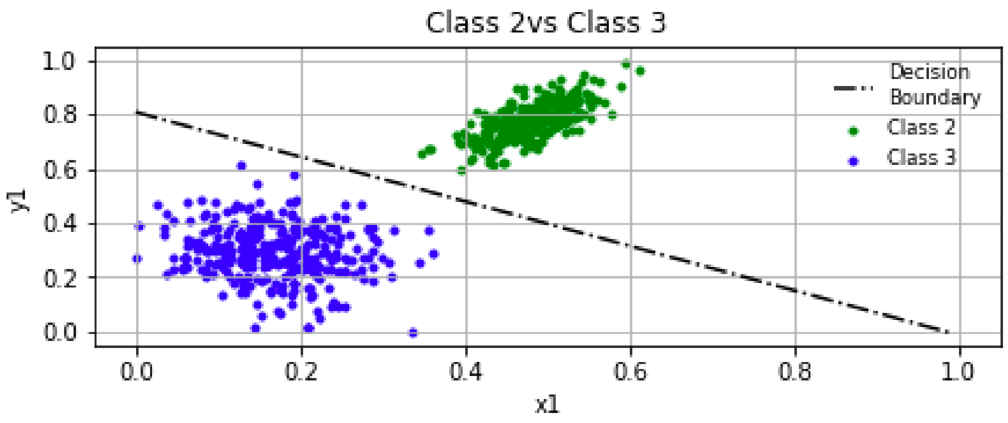
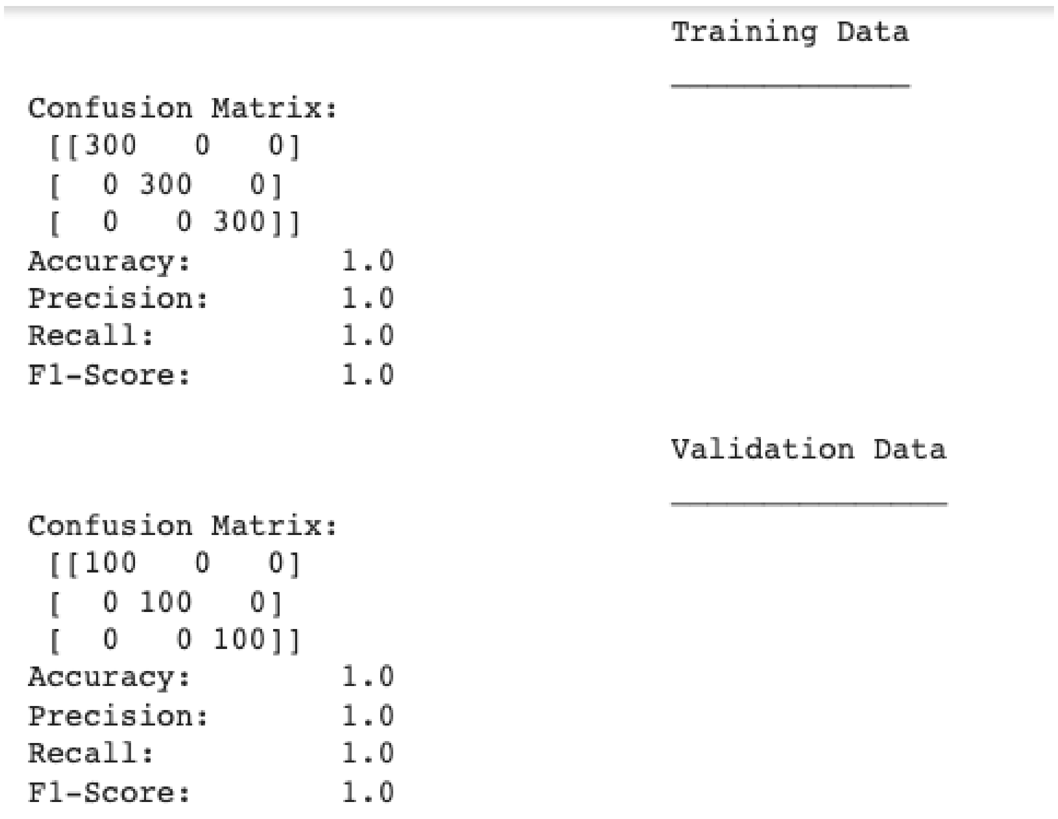


Fig. 1.7: Class 2 vs Class 3

This plot shows the decision boundary between class 2 and class 3. Just like the above plot, we can see a line between the two classes. This because of the Perceptron model which is a linear classifier.

1.2.3 Confusion matrix and classification accuracy

Training and validation confusion matrix : The above two confusion matrices are for training and the testing data. We can see that the diagonal of the matrix has the highest number of samples. This means the predicted class and the actual class is same for most of the data points (Here all the data points lie on diagonal). It means that our accuracy is very high, which is also evident from accuracy our code showed. It is 100% for perceptron on linear dataset.

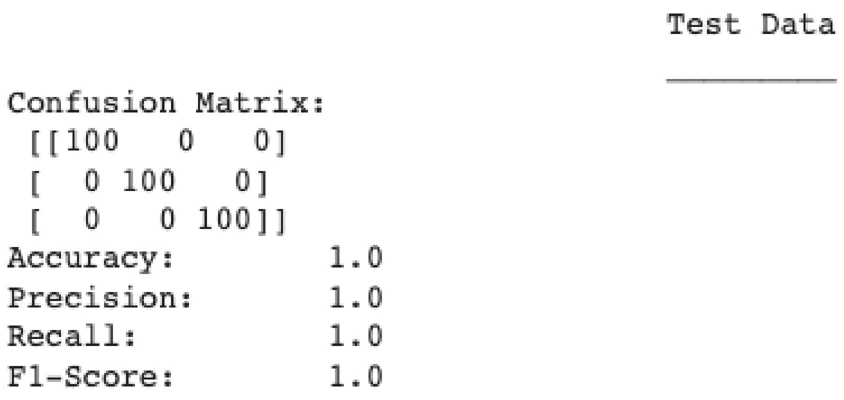


```
                                    Training Data
                                    _____________
Confusion Matrix:
 [[300   0   0]
 [  0 300   0]
 [  0   0 300]]
Accuracy:        1.0
Precision:       1.0
Recall:          1.0
F1-Score:        1.0

                                    Validation Data
                                    _______________
Confusion Matrix:
 [[100   0   0]
 [  0 100   0]
 [  0   0 100]]
Accuracy:        1.0
Precision:       1.0
Recall:          1.0
F1-Score:        1.0
```

Fig. 1.8: Confusion matrix

Testing confusion matrix :if you look carefully at the above two confusion matrices, you will see that those are for training and the validation dataset. Or I would say the dataset which the model has already seen or experienced. We can also see that almost all the data points are in the diagonal, indicating a very high accuracy. But does that mean that my model is generalised or trained well ? Maybe. We actually can't claim it. To measure the true accuracy of model, we allowed it to do predictions on unseen dataset (we call this test dataset here). The figure below is the confusion matrix for the same. You can see that the almost all the points are on the diagonal. Thus, it means that the model has trained well and it is able to perform good on an unseen dataset.



```
                                    Test Data
                                    _________

Confusion Matrix:
 [[100   0   0]
 [  0 100   0]
 [  0   0 100]]
Accuracy:        1.0
Precision:       1.0
Recall:          1.0
F1-Score:        1.0
```

Fig. 1.9: Confusion matrix for test dataset

1.2.4 Results on non-linear dataset

In the above two plots, we can see that the error for training is going down and is tending towards zero. In the other plot where validation training loss vs epochs is given, we can see that it is kind of erratic. Sometimes, it is going down and sometimes, it is going up. It shows that the model was not able to learn the data patterns properly. Interestingly, we can see that the training loss is much lower than the validation loss. This shows that the model is getting overfit. To handle overfitting, we use regularizers or dropout, which is out of scope for this report.

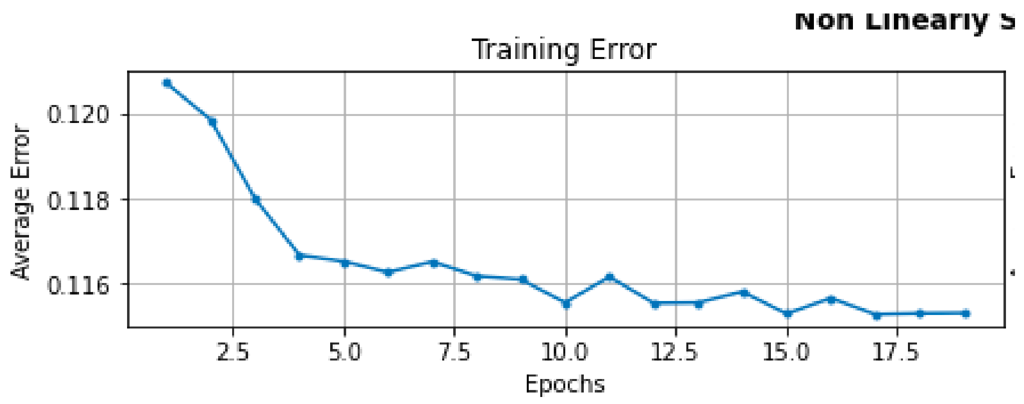


Fig. 1.10: Training loss vs epochs

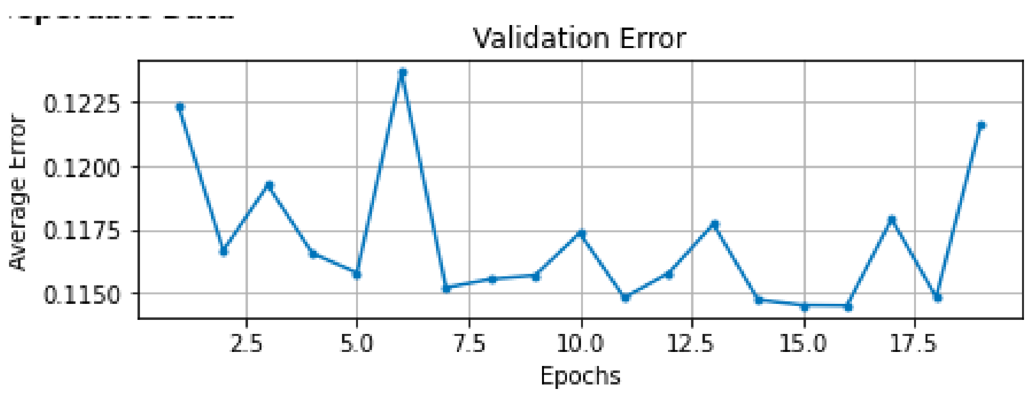


Fig. 1.11: Validation loss vs epochs

1.2.5 Decision boundary for non-linear dataset

As mentioned above, perceptron learning is a linear classifier. So no matter what, it will always create a linear decision boundary. In this case of non-linear dataset, it is trying to make a line between the classes. However, it won't work as the dataset is spiral in nature. Thus it is having a comparatively high loss. This is also evident from the confusion matrix.

1.2.6 Confusion matrix for non-linear dataset

Training, validation and testing confusion matrix : The above two confusion matrices are for training and the validation data. We can see that the diagonal of the matrix does not have the highest number of samples. Many data points are lying on non-diagonal positions. This means the predicted class and the actual class is not the same for many data points, thereby indicating incorrect prediction. It means that our accuracy is low, which is also evident from accuracy our code showed. It is around 56% for perceptron on non-linear training dataset. Since the model hasn't trained well, we can see a similar thing for the testing dataset. The model failed to capture the non-linearity of the dataset.

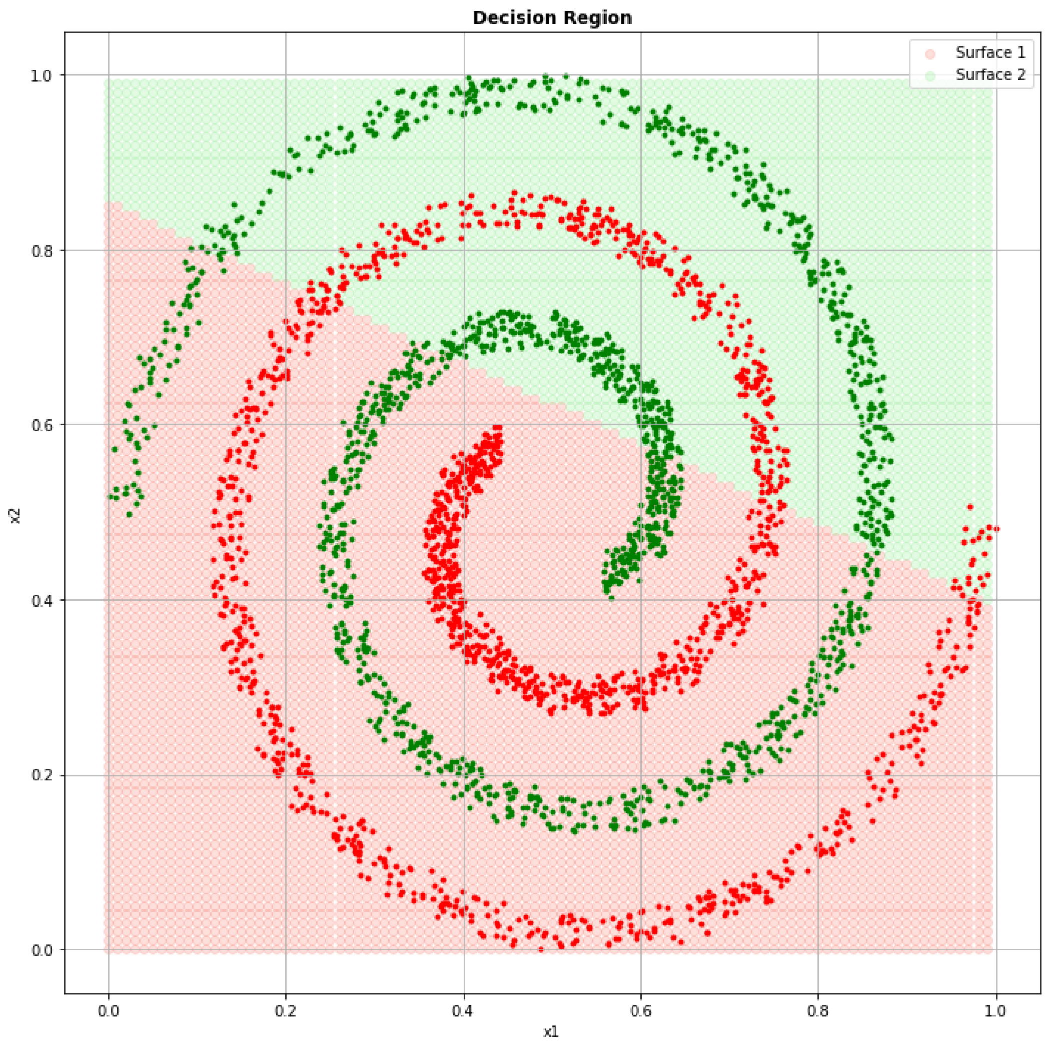


Fig. 1.12: Decision boundary

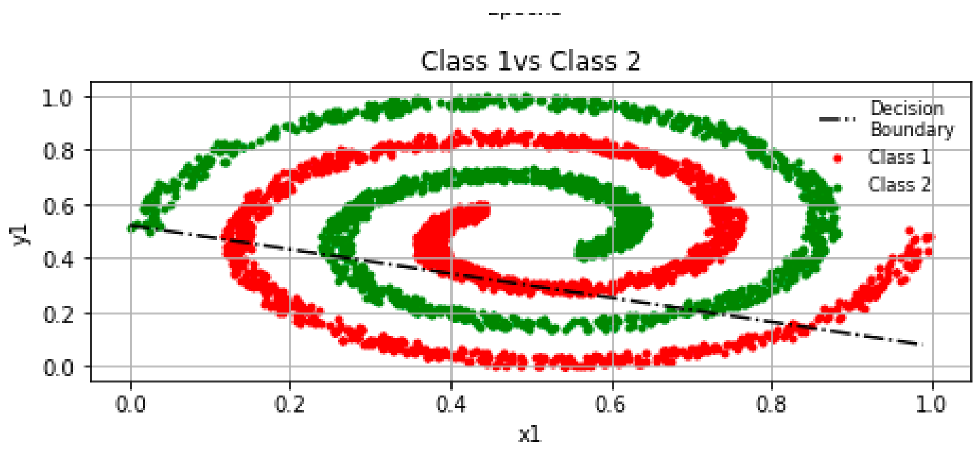
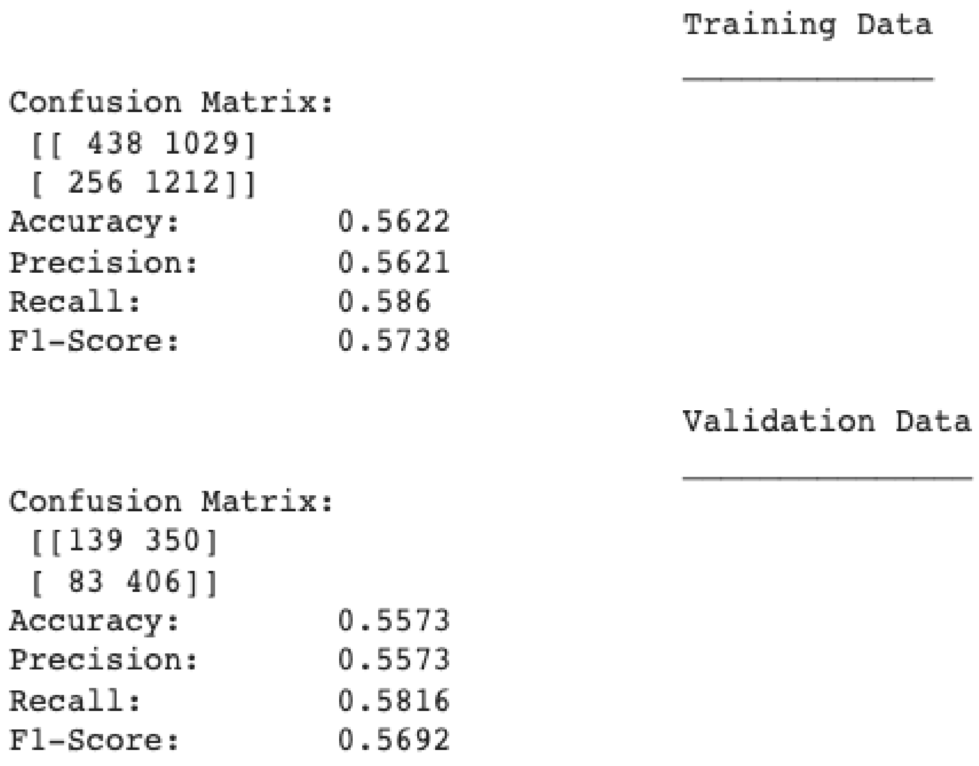


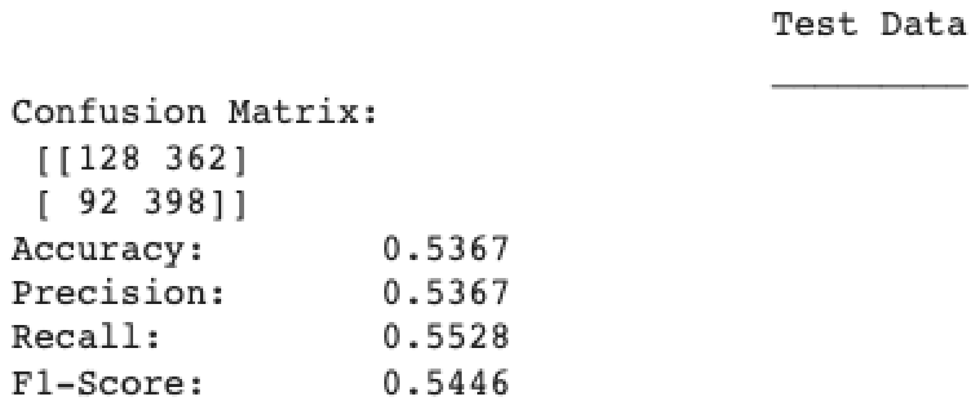
Fig. 1.13: Decision boundary



```
                                       Training Data
                                       _____________
Confusion Matrix:
 [[ 438 1029]
 [ 256 1212]]
Accuracy:        0.5622
Precision:       0.5621
Recall:          0.586
F1-Score:        0.5738

                                       Validation Data
                                       _______________
Confusion Matrix:
 [[139 350]
 [ 83 406]]
Accuracy:        0.5573
Precision:       0.5573
Recall:          0.5816
F1-Score:        0.5692
```

Fig. 1.14: Confusion matrix for training data



```
                                               Test Data
                                               _________
Confusion Matrix:
 [[128 362]
 [ 92 398]]
Accuracy:          0.5367
Precision:         0.5367
Recall:            0.5528
F1-Score:          0.5446
```

Fig. 1.15: Confusion matrix for test data

Looking carefully at the accuracies, we can also see that it around 50%. 50% is the lowest accuracy a machine learning classifier can attain. And it is possible when model hasn't learned anything and is giving same predictions for all the datapoints.

Chapter 2

Perceptron Regression

2.1 Brief description of the dataset

We were given two datasets :

1. **Dataset 1:** 1-dimensional (Univariate) input data. The scatter plot is shown below.

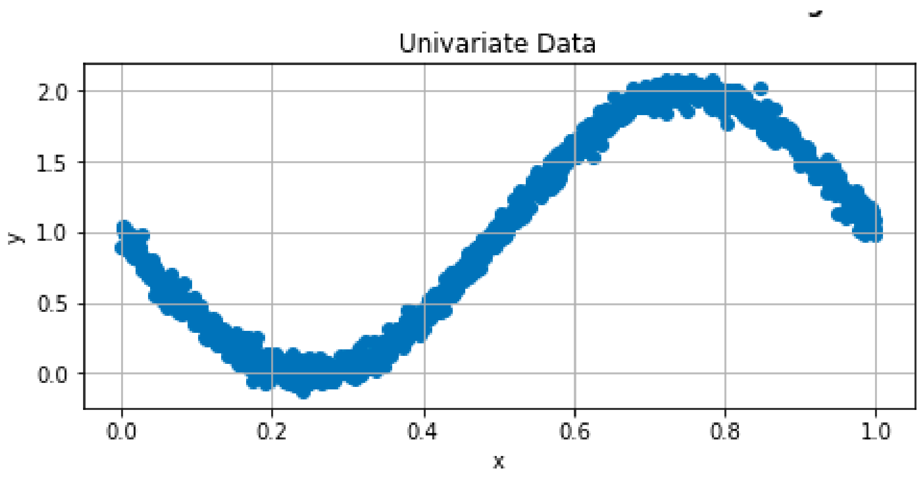


Fig. 2.1: Scatter plot of univariate

2. **Dataset 2:** 2-dimensional (Bivariate) input data. The 3D scatter plot is shown below.

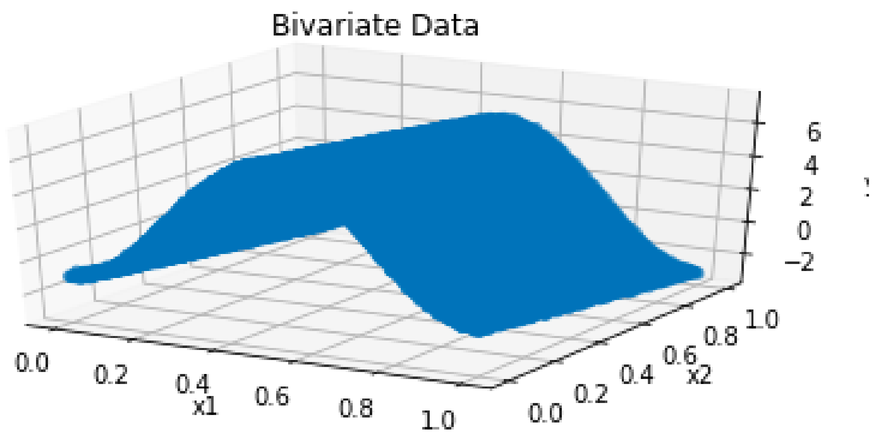


Fig. 2.2: 3D Scatter plot of bivariate

2.2 Observations

This section will discuss the observations from our experiments. It will also mention the inferences that one can take from the observations.

2.2.1 Average error vs epochs

In the two plots (fig 2.3) below, we can see the training loss and validation loss going down. After a particular value, it is not going down or it has become stagnant. This is because the perceptron (a linear classifier) is trying to fit a line of the form `y=mx+c` to a non-linear dataset. Thus it can achieve accuracy up to a particular upper bound, but not more than that. To effectively learn the non-linearity in the data, we need a non-linear regressor such as a FCNN. This can be better shown in below sections where we present how regressor is fitting a linear equation.

2.2.2 MSE vs datasets

We observe that the training and validation error is lower compared to the test error . This is because the model has seen the former two dataset while the latter is unseen. Since it wasn't able to learn the non-linearity in the data, it is showing high error for the testing data.

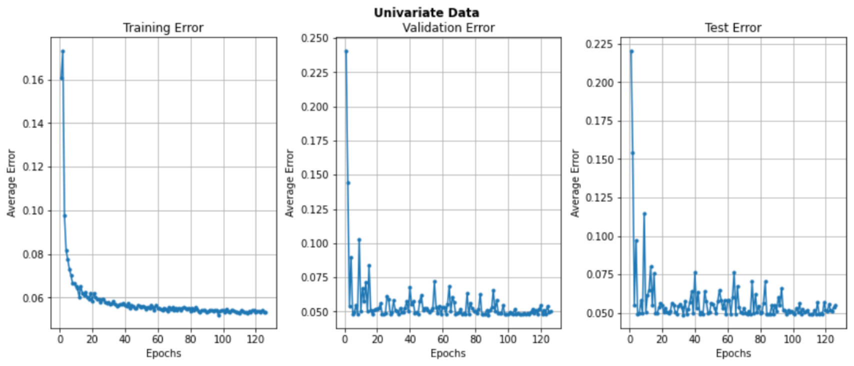


Fig. 2.3: Error vs epochs

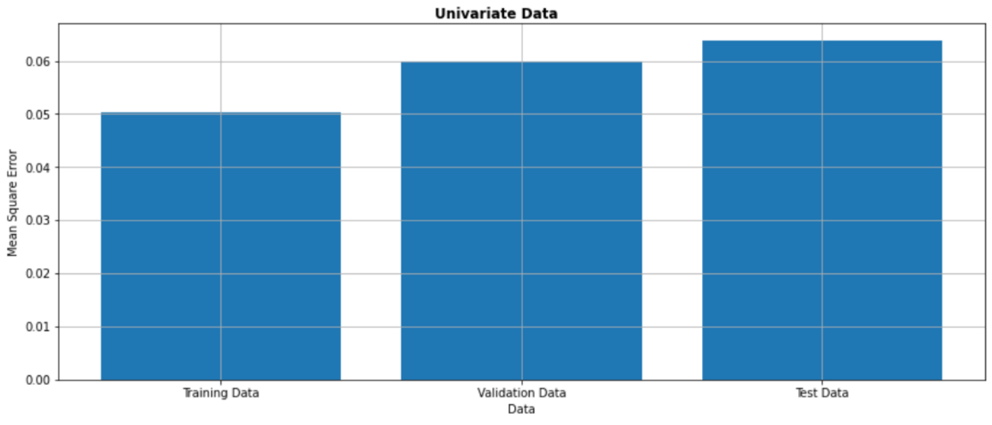


Fig. 2.4: MSE of the datasets by the trained model

2.2.3 Output data vs predicted data

In this plot, we have the x axis as the actual values and y axis as the predicted values. Let us now ponder upon this. Suppose our regression model is 100% accurate, then what is the expected plot ? All the data points will lie on the diagonal of the plot. This is because both the x values and y values are exactly the same. Now, consider our plot (fig 2.4). You can see that a significant portion is not on the diagonal. This means that the model wasn't able to learn some patterns of the data (in this case, non linear patterns).

2.2.4 Output data vs predicted boundary

In this plot, we have shown the predicted surface by the perceptron algorithm. With perceptron, we will always get a linear surface (dimensions can be anything). For univariate data, we can see a line. This was the best fit line the algorithm can produce. The data is

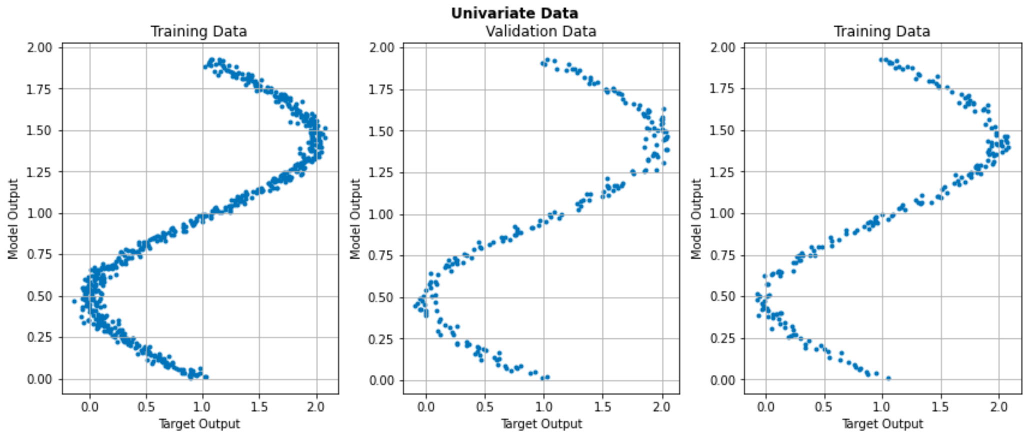


Fig. 2.5: Actual vs predicted for univariate

however non-linear, so it wasn't able to reduce the error to zero or near to zero. A non-linear regressor like polynomialic regressor can achieve this.

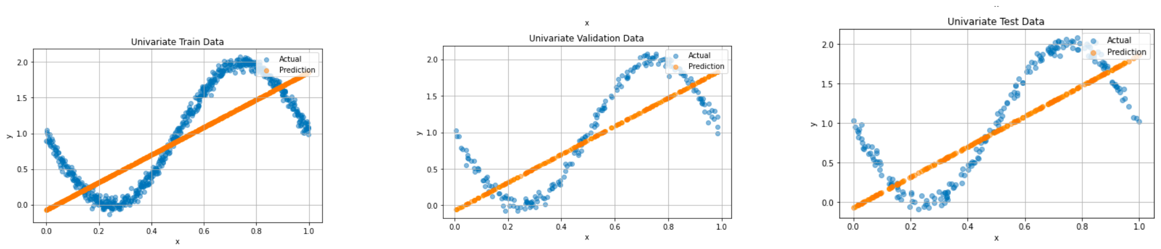


Fig. 2.6: Actual vs predicted surface for univariate

2.2.5 Perceptron results on bivariate dataset

Perceptron works in a similar way as mentioned in the previous sections for this dataset as well.

1. **Error vs Epochs** : We can see that the training and validation losses are decreasing. However, just like in the previous case, it is not able to go below a certain lower bound. This is because the linear surface (here plane) can't capture the non-linearity of the data. This will be more clear when we discuss the surface that is being fit by the model.

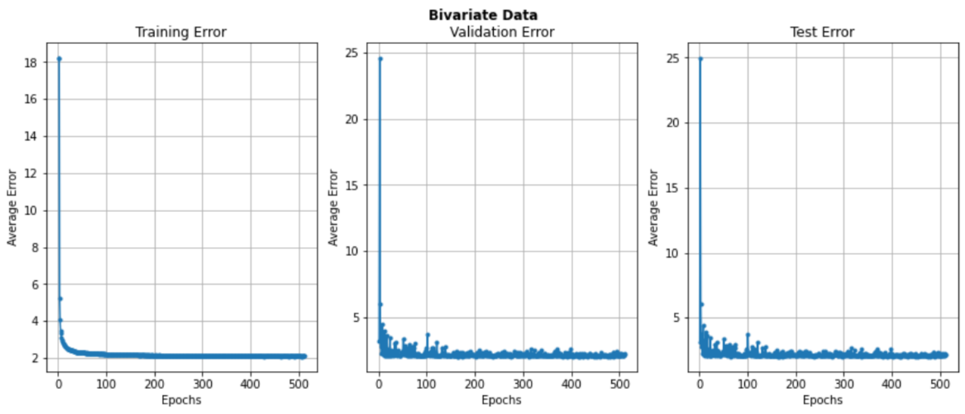


Fig. 2.7: Error v/s epochs

2. **MSE vs datasets** : As in the previous case of univariate data, we got a similar MSE for training and validation, but for testing, it's high. This is because the model has seen the former two dataset while the latter is unseen. Since it wasn't able to learn the non-linearity in the data, it is showing high error for the testing data.

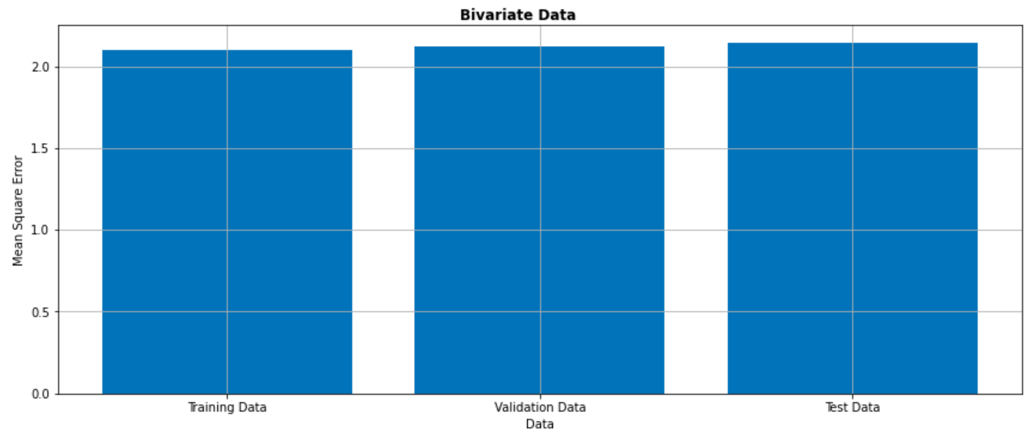


Fig. 2.8: MSE of the datasets by the trained model

3. **Output data vs predicted data** :

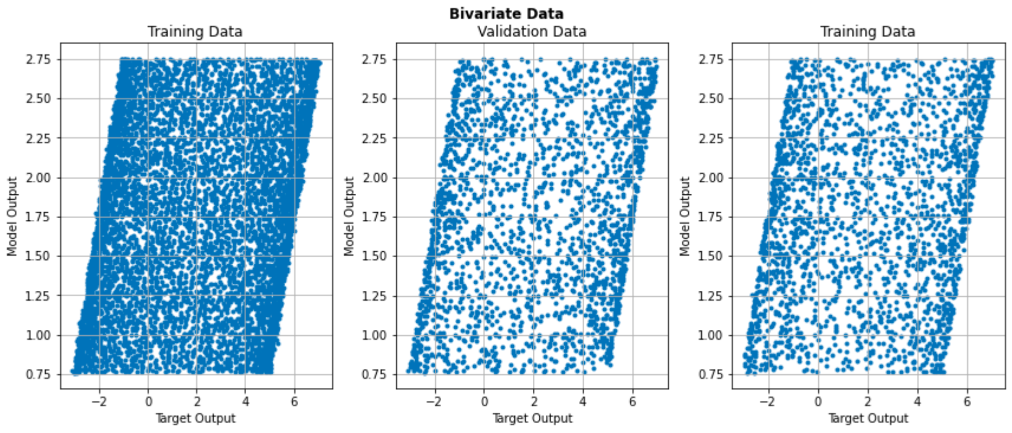


Fig. 2.9: Actual vs predicted for bivariate

4. **Fitted Surface** : Perceptron has fitted a linear surface, a plane to the dataset as we can see in the figure below. Errors will be high because it has yet to capture the non-linearity of the data.

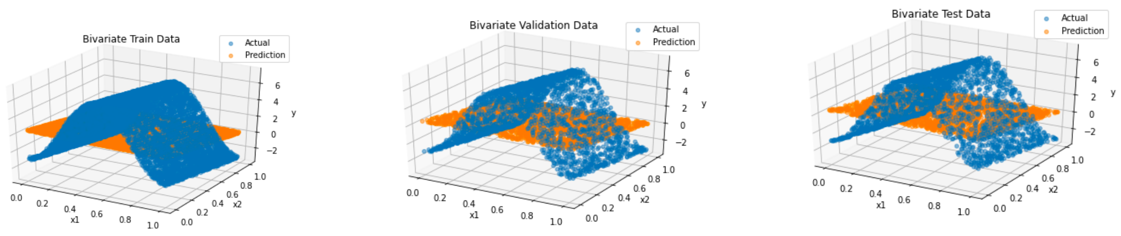


Fig. 2.10: Predicted surface for Perceptron

So the only solution to this problem is using a non-linear classifier which we will be using in the next chapter.

Chapter 3

FCNN Classification

3.1 Brief description of the dataset

We were given two datasets :

1. **Dataset 1: Linearly separable classes:** 3 class, 2-dimensional linearly separable data is given. Each class has 500 data points. The scatter plot for the dataset is given below.

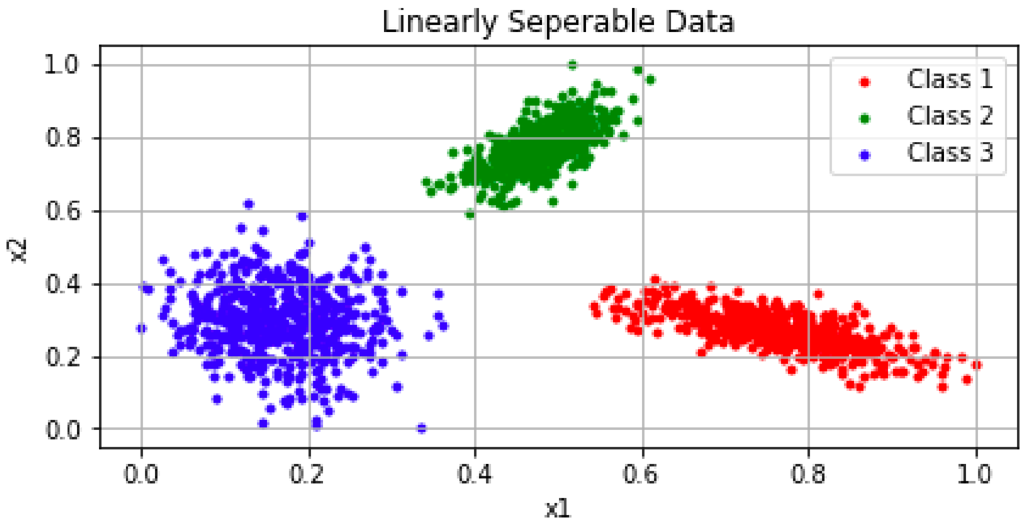


Fig. 3.1: Linearly separable data

2. **Dataset 2: Nonlinearly separable classes:** 2-dimensional data of 2 classes that are non-linearly separable. The number of examples in each class and their order was given at the beginning of each file. The scatter plot for the dataset is given below.

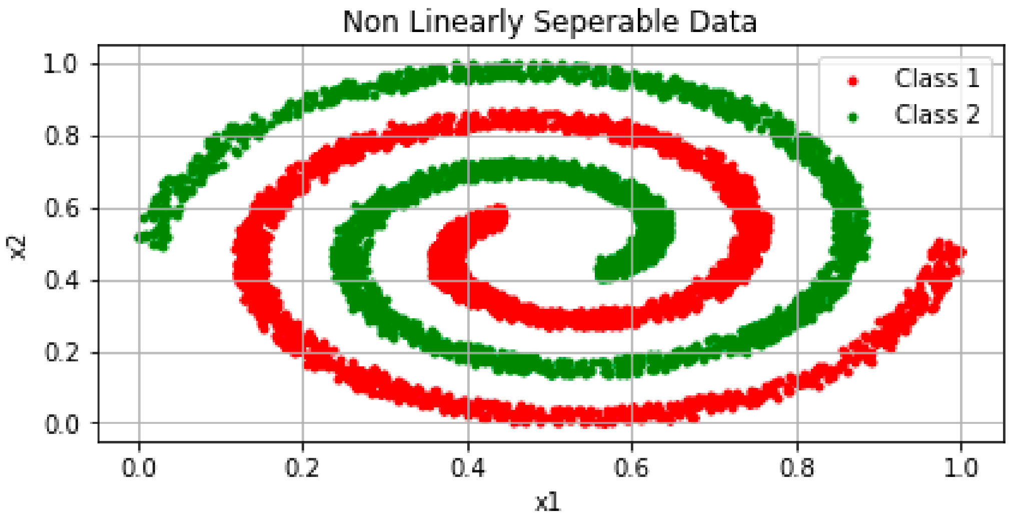


Fig. 3.2: Non-Linearly separable data

3.2 Observations for linear dataset

Here we will not be focusing much on linear dataset, as perceptron, a linear classifier, was able to handle it at a very high accuracy. In this section, we will dive deep into the non linear dataset, which was not solved by perceptron in the previous sections.

3.2.1 Average error vs epochs

In the above two plots (fig 3.3 and fig 3.4), we can see that the error (for both training and validation) is going down and is tending towards zero. This indicates that the model was able to learn the patterns of the data. Comparing both the plots, we can see that is drop is more steep for training data than for the validation data. This is expected because the model. This is exactly similar to the plots we have seen in the perceptron learning algorithm. However, one thing to notice here is, the accuracy is more erratic at the beginning than at the end. This is because of the stochasticity of the optimizer which is trying to find a minima of the loss curve.

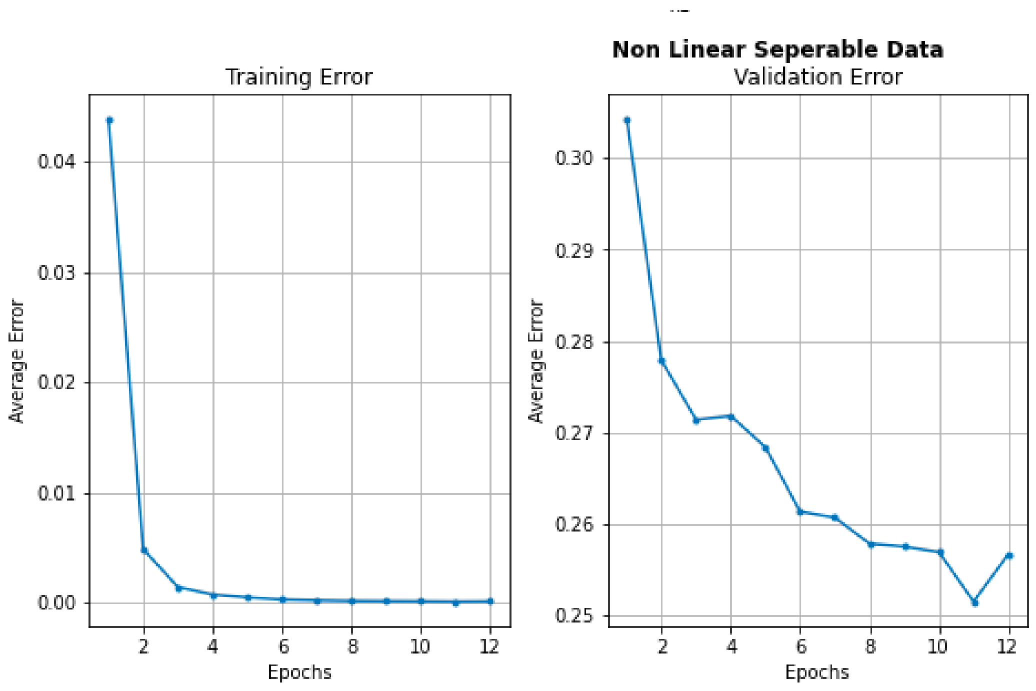


Fig. 3.3: Linearly separable data

3.2.2 Decision region plots

In this report, we have implemented one-to-one approach. For the first plot, we have the decision boundary between class 1 and class 2. We can see that it is a non-linear boundary. This is expected, as FCNN algorithm targets to create a contour between the classes. It can be of any decision boundary of any degree, as it is a non-linear classifier.

The figure below (fig3.5) shows the outputs of the hidden nodes. One can see how they are trying to segregate the data into three classes.

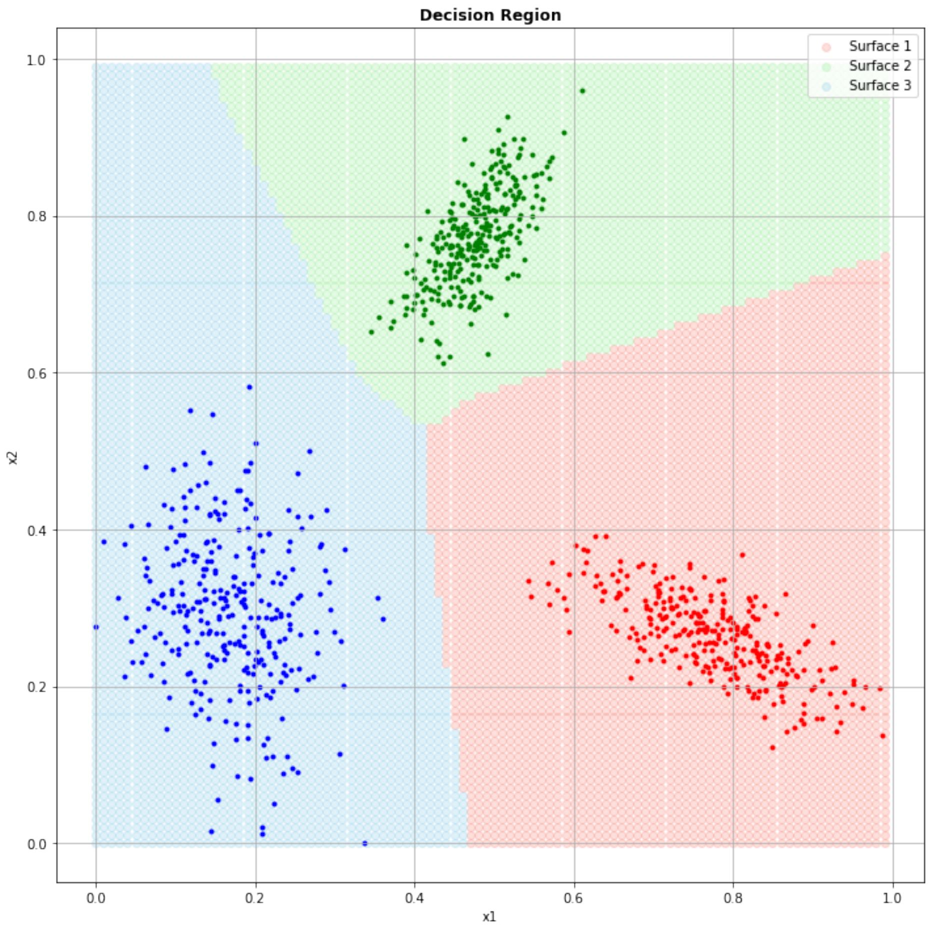


Fig. 3.4: Linearly separable data by 1 hidden layer

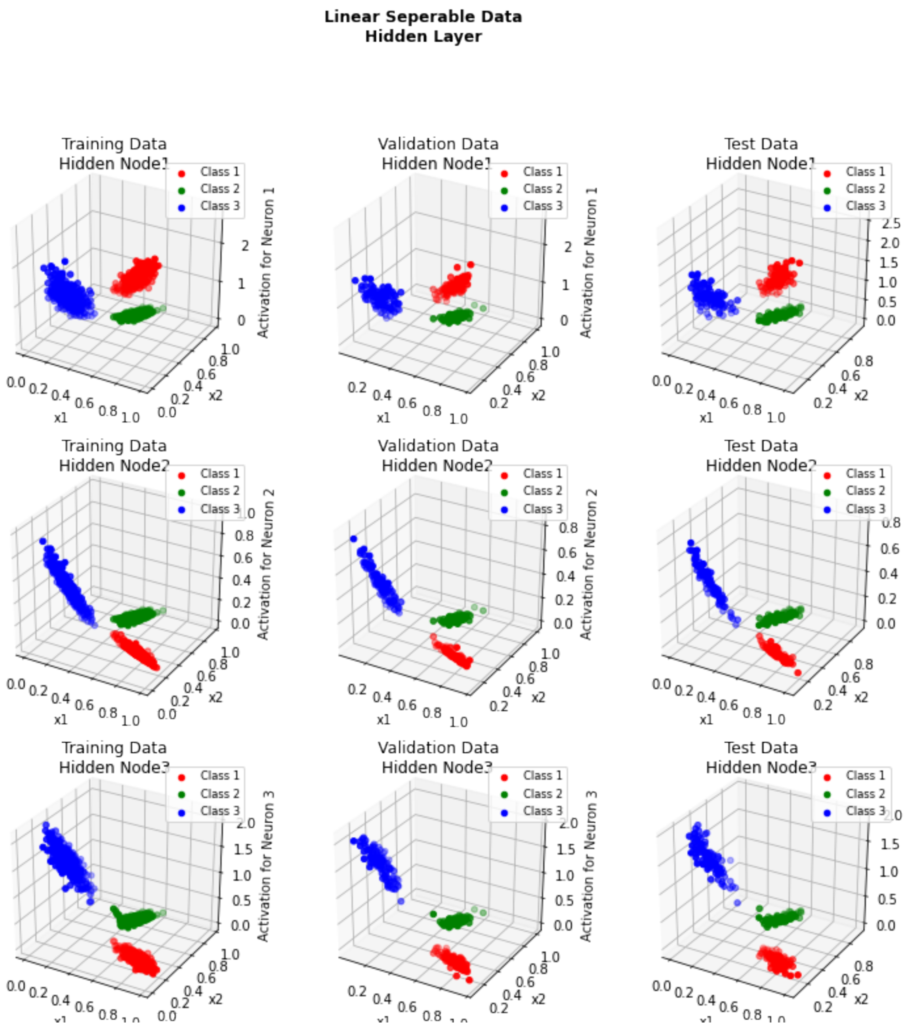


Fig. 3.5: Output of different nodes

3.2.3 Confusion matrix and classification accuracy

As discussed in the previous sections, to measure the true accuracy of any model, we need to look at the test dataset as it is unseen by the model. The following figures (fig 3.7 and 3.8) show the confusion matrix.

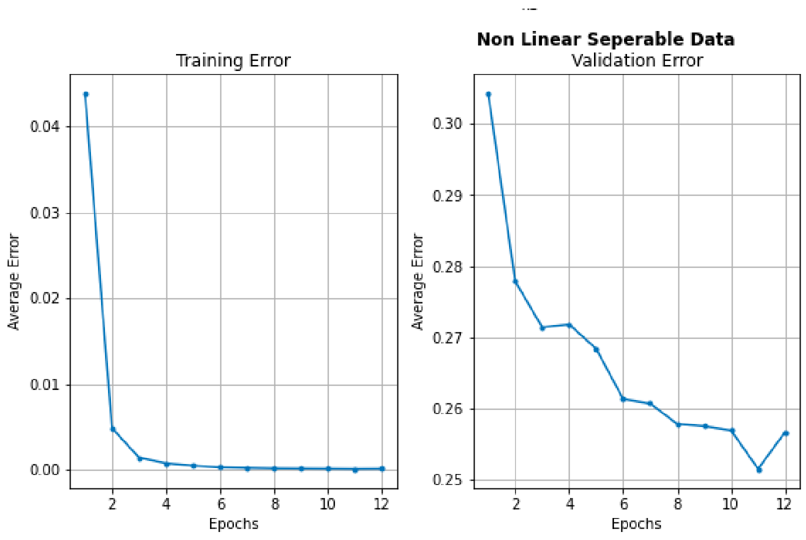
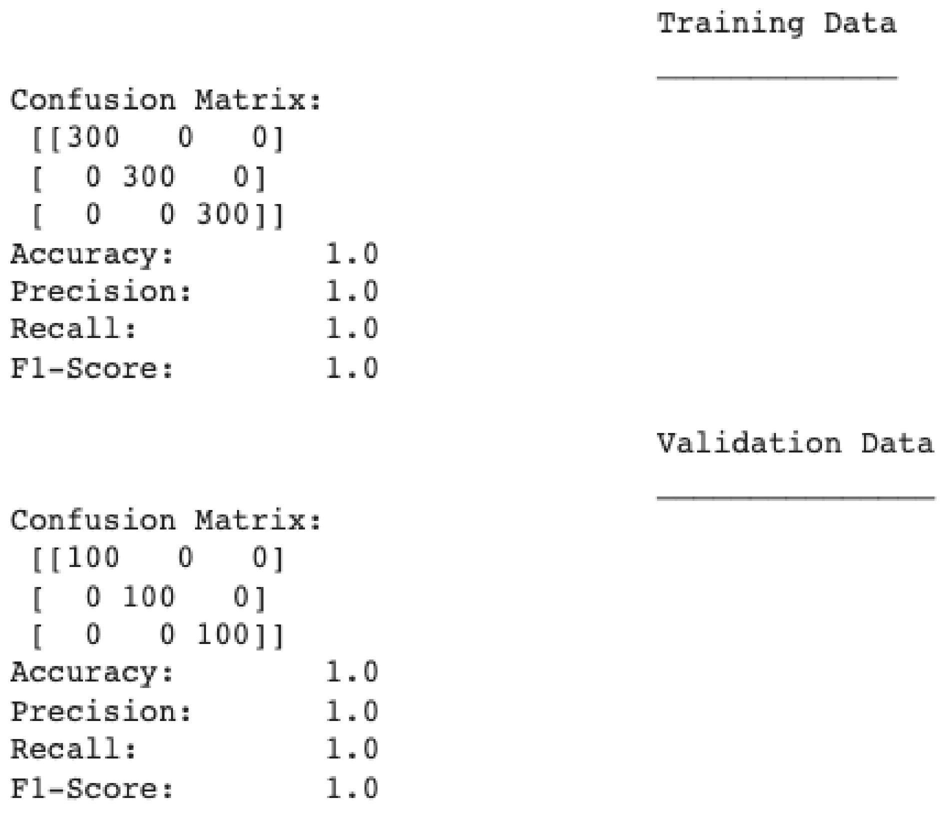


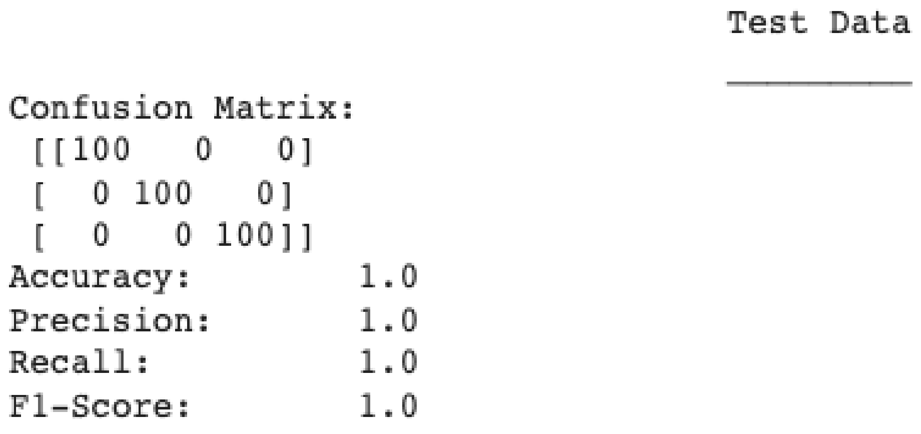
Fig. 3.6: Error plots for Linearly separable data



```
                                    Training Data
                                    _____________
Confusion Matrix:
 [[300   0   0]
 [  0 300   0]
 [  0   0 300]]
Accuracy:          1.0
Precision:         1.0
Recall:            1.0
F1-Score:          1.0

                                    Validation Data
                                    _______________
Confusion Matrix:
 [[100   0   0]
 [  0 100   0]
 [  0   0 100]]
Accuracy:          1.0
Precision:         1.0
Recall:            1.0
F1-Score:          1.0
```

Fig. 3.7: CF for Linearly separable data



```
                                    Test Data
                                    __________
Confusion Matrix:
 [[100   0   0]
 [  0 100   0]
 [  0   0 100]]
Accuracy:          1.0
Precision:         1.0
Recall:            1.0
F1-Score:          1.0
```

Fig. 3.8: CF for Linearly separable data

3.3 Observations for Non-linear dataset

We will be discussing this topic in details.

3.3.1 Effect of number of nodes in hidden layer

Number of nodes and layers is an interesting feature of FCNN. In learning theory, we call them width and depth respectively. Lowering the number of nodes is bringing down the accuracy of the model. However, increasing the number of nodes boosts it up. As we know, number of nodes is a hyperparameter for running neural networks model. So there is no general rule for finding the optimal number of nodes. In the theory of learning, our target to learn the function f which can predict with a good efficiency when a data X is passed though it. A neural network with one node can learn a very complex function given a high number of nodes attached to it. This however brings in two problems :

1. More technical resources are required.
2. We don't know an optimal value of nodes that are required.

However the learning capacity is high with more nodes. We will see an alternate solution to this in the next section where we keep less number of nodes but more layers.

3.3.2 Effect of number of hidden layers

It is known that the learning capacity of any model with increased number of nodes is same as that of more number of layer (but less nodes per layer). However more technical resources are required in the first case. To tackle that, we can have more hidden layers which can also learn any type of mapping. But having a significant number of layers can introduce a problem called vanishing gradients. What is vanishing gradient now? It is actually the gradients getting diminished because of which the weights aren't updated much. This topic is out of scope so we won't be discussing it much in this report. The universal approximation theorem suggests that one/two hidden layers are sufficient to approximate any complex continuous functions. Thus 1-2 layers are sufficient.

3.3.3 Average error vs epochs

In the plots, we can see that it is more erratic in the beginning. This is because of the stochastcity in selecting a sample from the training dataset. We also tried to make it deterministic by simply iterating over the dataset, and we found that the accuracy was stuck at 50% (lowest accuracy a model can achieve). Thus, stochasticity is the major reason for improving the learning of the model.

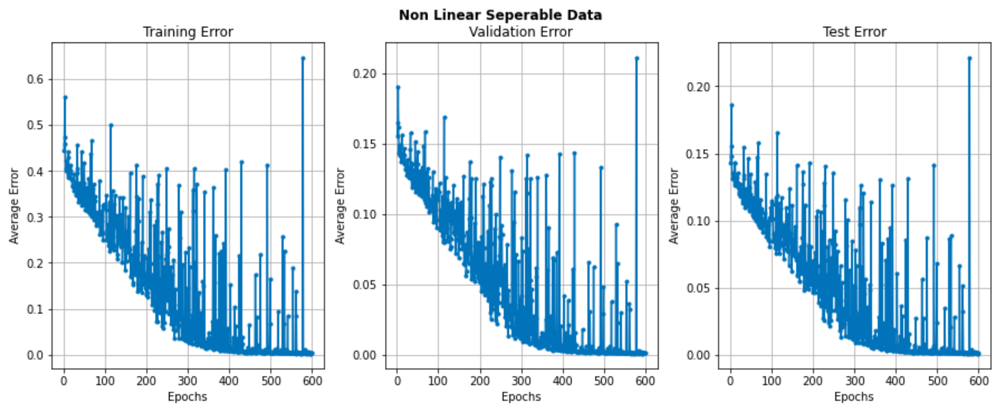


Fig. 3.9: Error plots for Linearly separable data

3.3.4 Decision region plots

Since it is a non-linear classifier, we can see a non-linear decision boundary for neural networks with both one and two layers. The model was capture the non-linearity of the data.

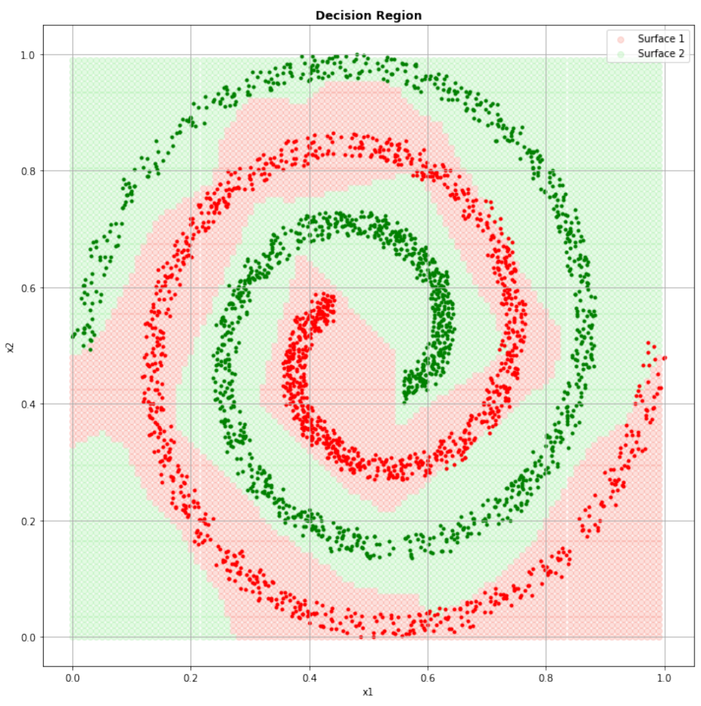


Fig. 3.10: Decision boundary (NN 1 hidden) for Non-Linearly separable data

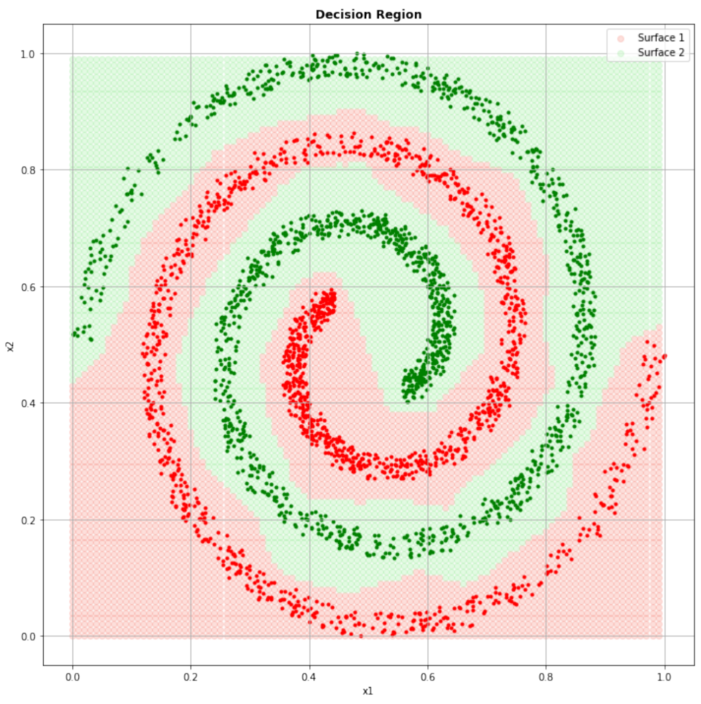
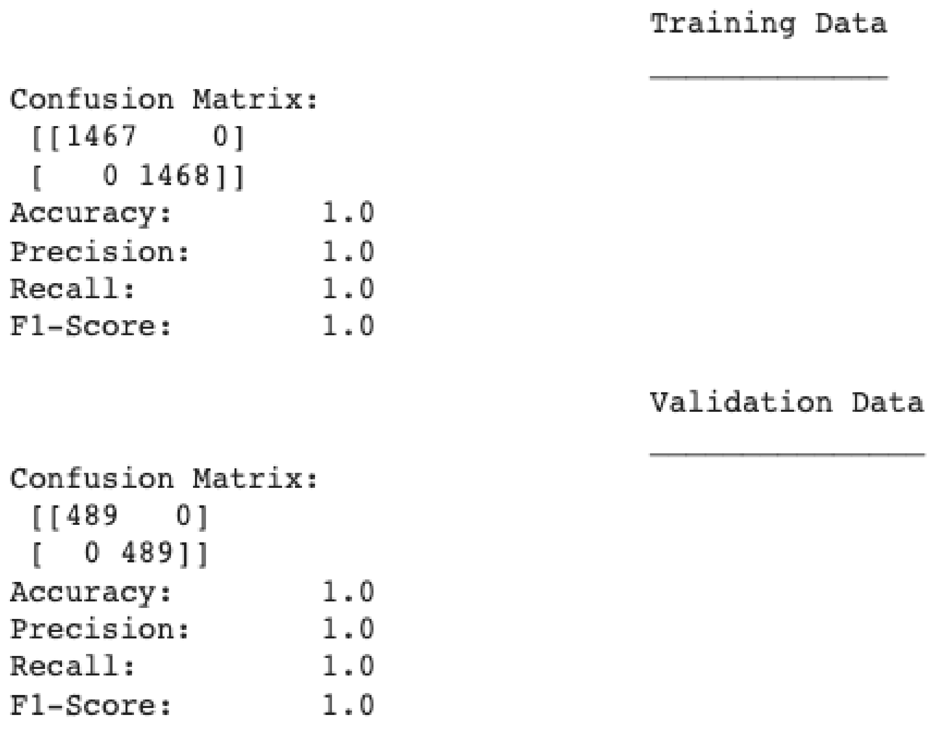


Fig. 3.11: Decision boundary (NN 2 hidden) for Non-Linearly separable data

The outputs of the hidden nodes are also provided below for neural networks. One can see how they are trying to segregate the data into three classes. Same diagram for 2 hidden layer neural networks is in the notebook.

3.3.5 Confusion matrix and classification accuracy

Since most of the samples are on the diagonal of the matrix, the model was able to learn the patterns of the data.



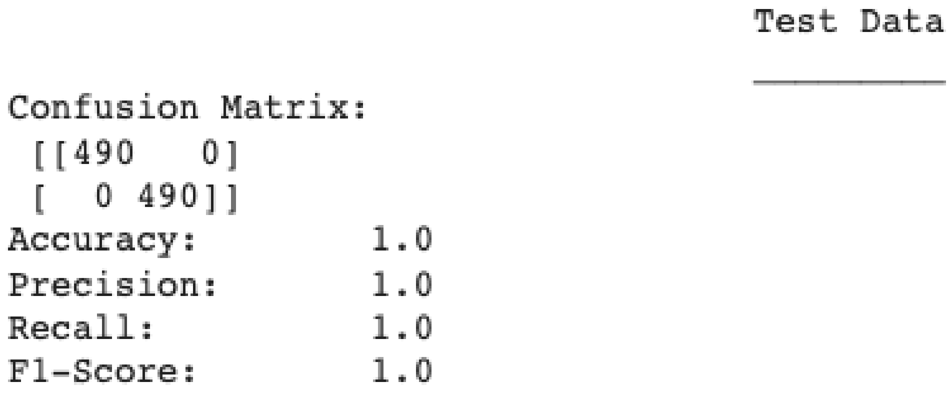
```
                                    Training Data
                                    _____________
Confusion Matrix:
 [[1467    0]
 [   0 1468]]
Accuracy:        1.0
Precision:       1.0
Recall:          1.0
F1-Score:        1.0

                                    Validation Data
                                    _______________
Confusion Matrix:
 [[489   0]
 [  0 489]]
Accuracy:        1.0
Precision:       1.0
Recall:          1.0
F1-Score:        1.0
```

Fig. 3.12: CF for Non-Linearly separable data (1 hidden)

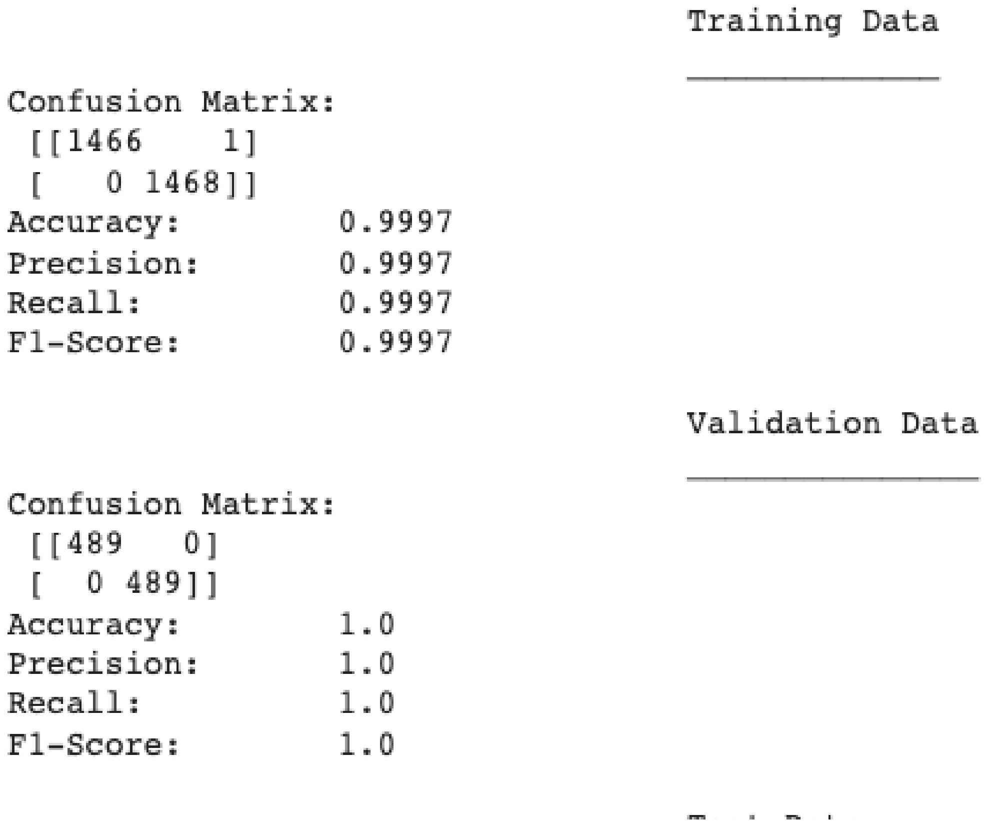
3.3.6 Output of output nodes

The output nodes are trying to segregate the data into two classes as can be seen from the figure below (fig 3.16).



```
                                            Test Data
                                            _________
 Confusion Matrix:
  [[490   0]
  [  0 490]]
 Accuracy:          1.0
 Precision:         1.0
 Recall:            1.0
 F1-Score:          1.0
```

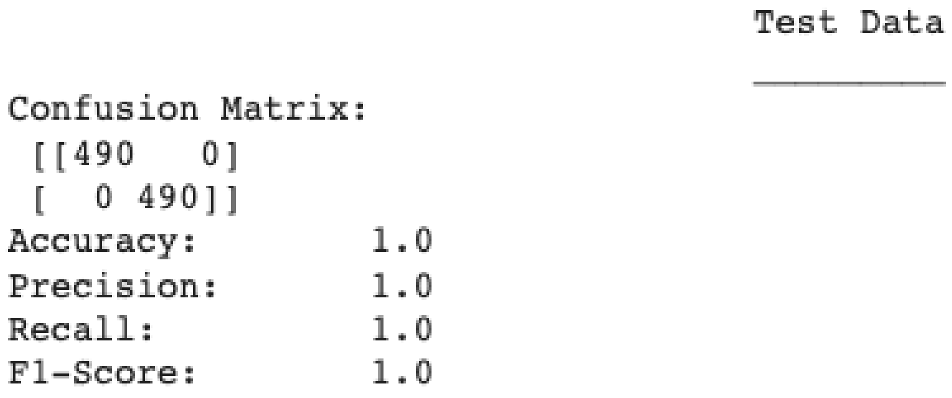
Fig. 3.13: CF for Non-Linearly separable data (1 hidden)



```
                                            Training Data
                                            _____________
Confusion Matrix:
 [[1466    1]
 [   0 1468]]
Accuracy:          0.9997
Precision:         0.9997
Recall:            0.9997
F1-Score:          0.9997

                                            Validation Data
                                            _______________
Confusion Matrix:
 [[489   0]
 [  0 489]]
Accuracy:          1.0
Precision:         1.0
Recall:            1.0
F1-Score:          1.0
```

Fig. 3.14: CF for Non-Linearly separable data (2 hidden)



```
                                        Test Data
                                        _________
Confusion Matrix:
 [[490   0]
 [  0 490]]
Accuracy:        1.0
Precision:       1.0
Recall:          1.0
F1-Score:        1.0
```

Fig. 3.15: CF for Non-Linearly separable data (2 hidden)

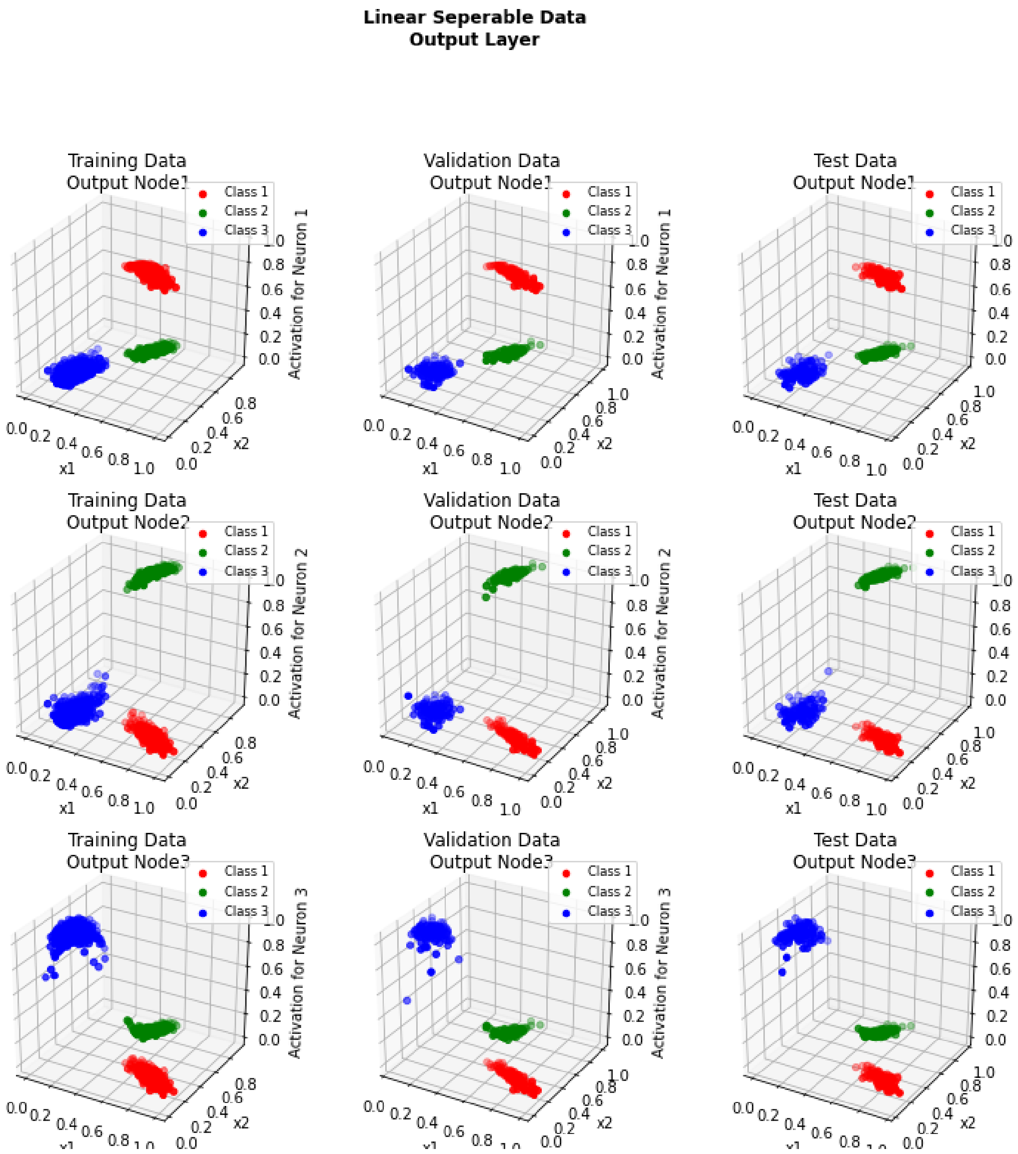


Fig. 3.16: Output of output nodes (1 hidden)

Chapter 4

FCNN Regression

4.1 Brief description of the dataset

We were given two datasets :

1. **Dataset 1:** 1-dimensional (Univariate) input data. The scatter plot is shown below.

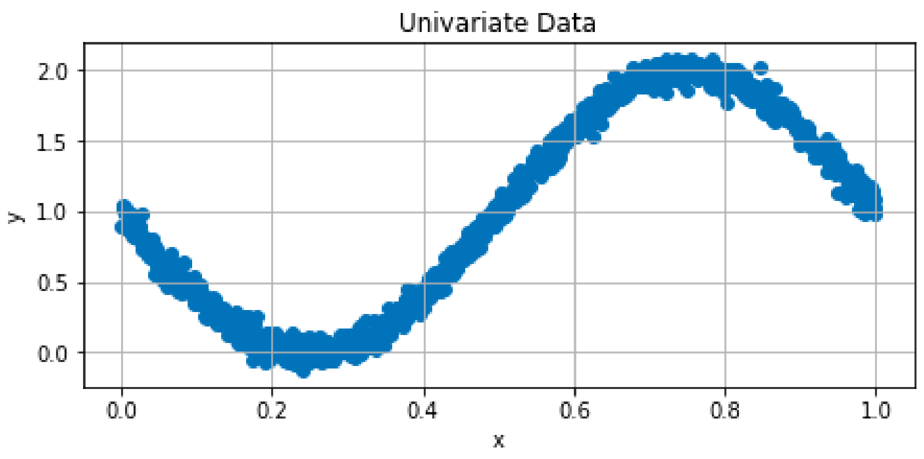


Fig. 4.1: Scatter plot of univariate

2. **Dataset 2:** 2-dimensional (Bivariate) input data. The 3D scatter plot is shown below.

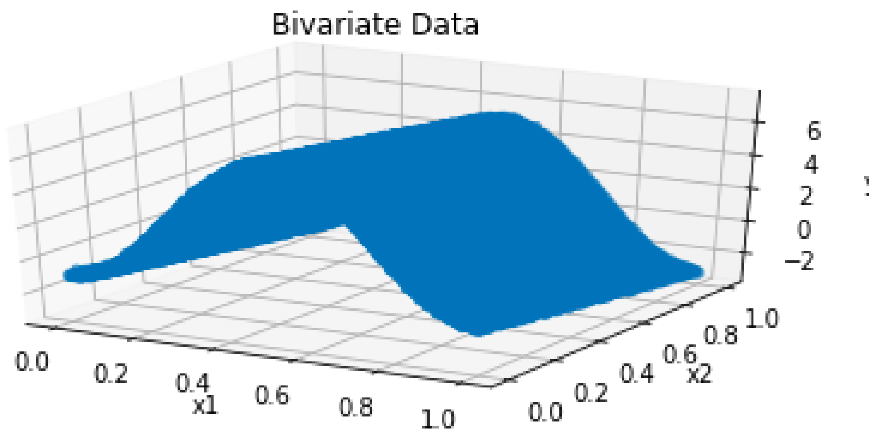


Fig. 4.2: 3D Scatter plot of bivariate

4.2 Observations

This section will discuss the observations from our experiments. It will also mention the inferences that one can take from the observations.

4.2.1 Average error vs epochs

In the two plots (fig 2.3) below, we can see the training loss and validation loss is going down. Since FCNN is a non linear regressor, it can better predict the values of dependent variable when the independent and dependent variables are non linearly related (like our case) . This becomes evident in below sections where we show how our regressor is fitting a non linear equation.

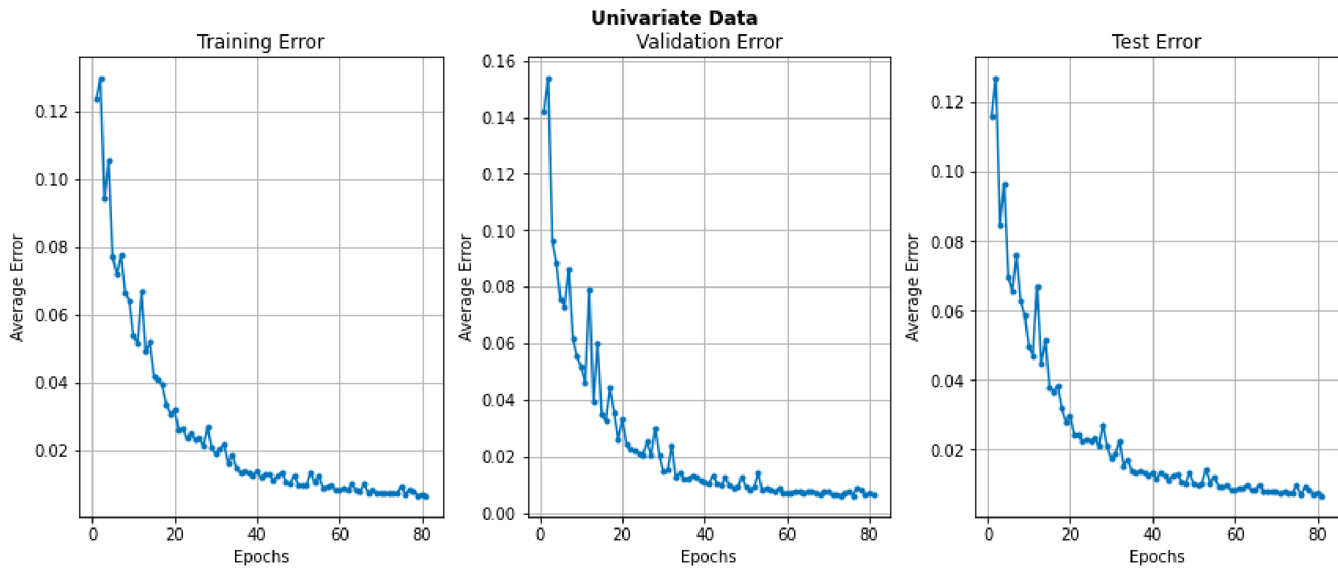


Fig. 4.3: Error vs epochs (1 hidden)

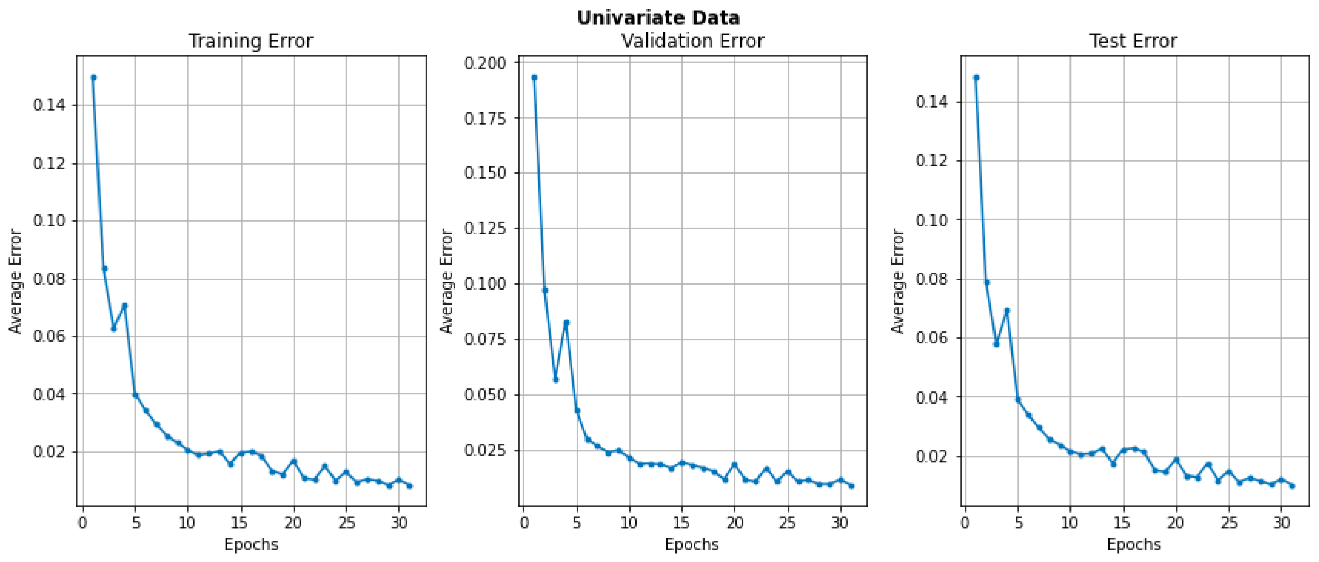


Fig. 4.4: Error vs epochs (2 hidden)

4.2.2 MSE vs datasets

Similar to Perceptron,we observe that the training and validation error is lower compared to the test error . This is because the latter dataset is unknown. However, since the FCNN can better approximate the non linear surface compared to Perceptron, we notice that the MSE for all three datasets is significantly less.

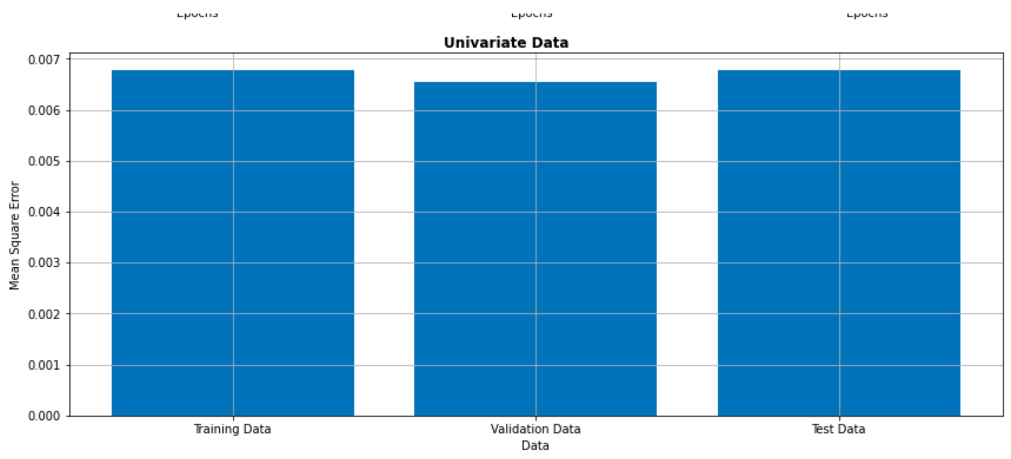


Fig. 4.5: MSE of the datasets by the trained model

4.2.3 Output data vs predicted data

In this plot, we have the x axis as the actual values and y axis as the predicted values. If our regression model is 100% accurate, then all the data points will lie on the diagonal of the plot because both the predicted values and actual values are very close to each other. Now, consider our plot (fig 4.6). You can see that a significant portion of the curve falls across the diagonal. This means that the model can learn all the non linear patterns in the data and predict successfully. (Fig 4.6)

Figure 4.7 shows the output of the output node. Please see how it is trying to map the function provided.

4.2.4 Output data vs predicted boundary

In this plot, we have shown the predicted surface by the fully connected connected neural net. With FCNN, it is possible to estimate any kind of non linear boundary. Hence our

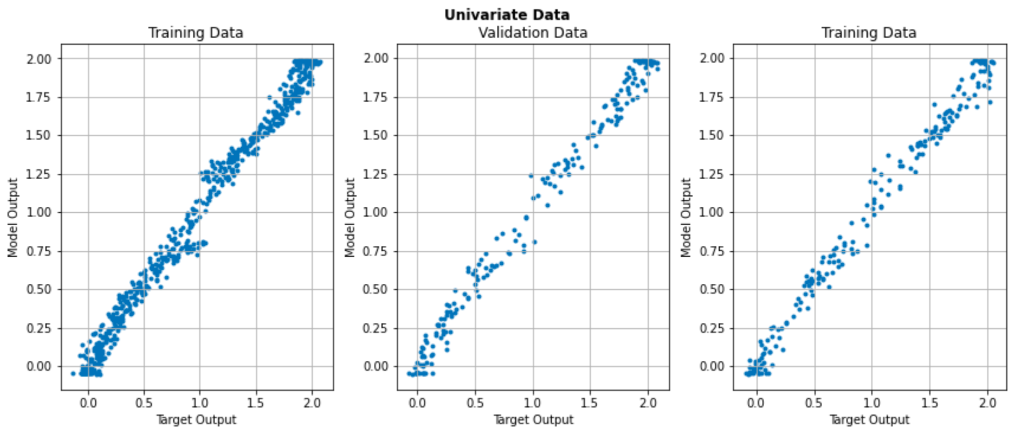


Fig. 4.6: Actual vs predicted for univariate (1 hidden)

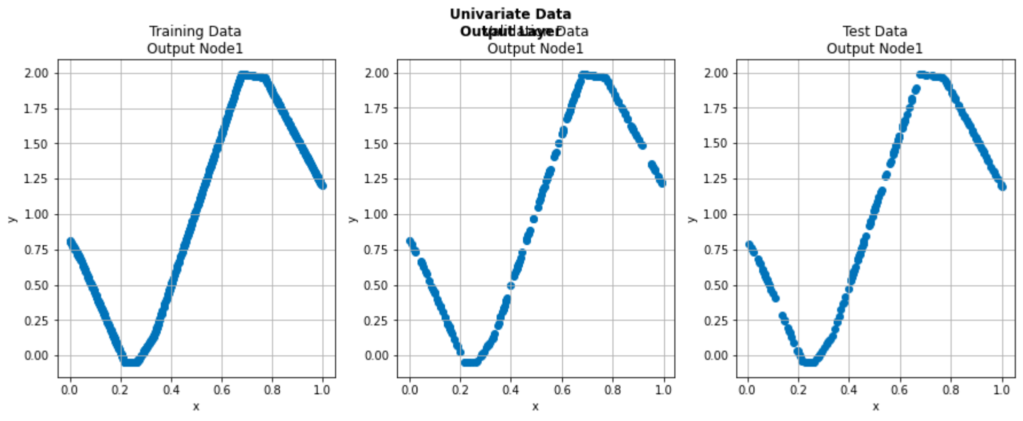


Fig. 4.7: Output node's output (1 hidden)

predicted decision boundary almost traces the output data reducing the error to almost zero. This shows the superiority of neural networks over Perceptron which can produce only linear decision surface.

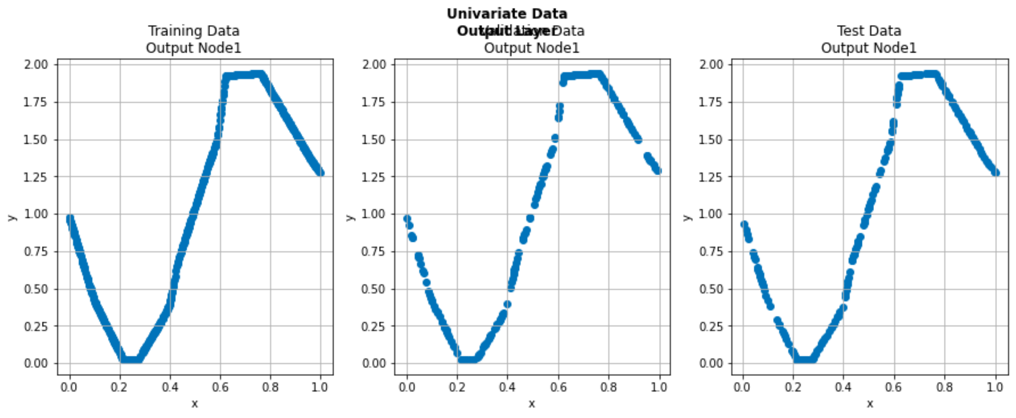


Fig. 4.8: Output node's output (2 hidden)

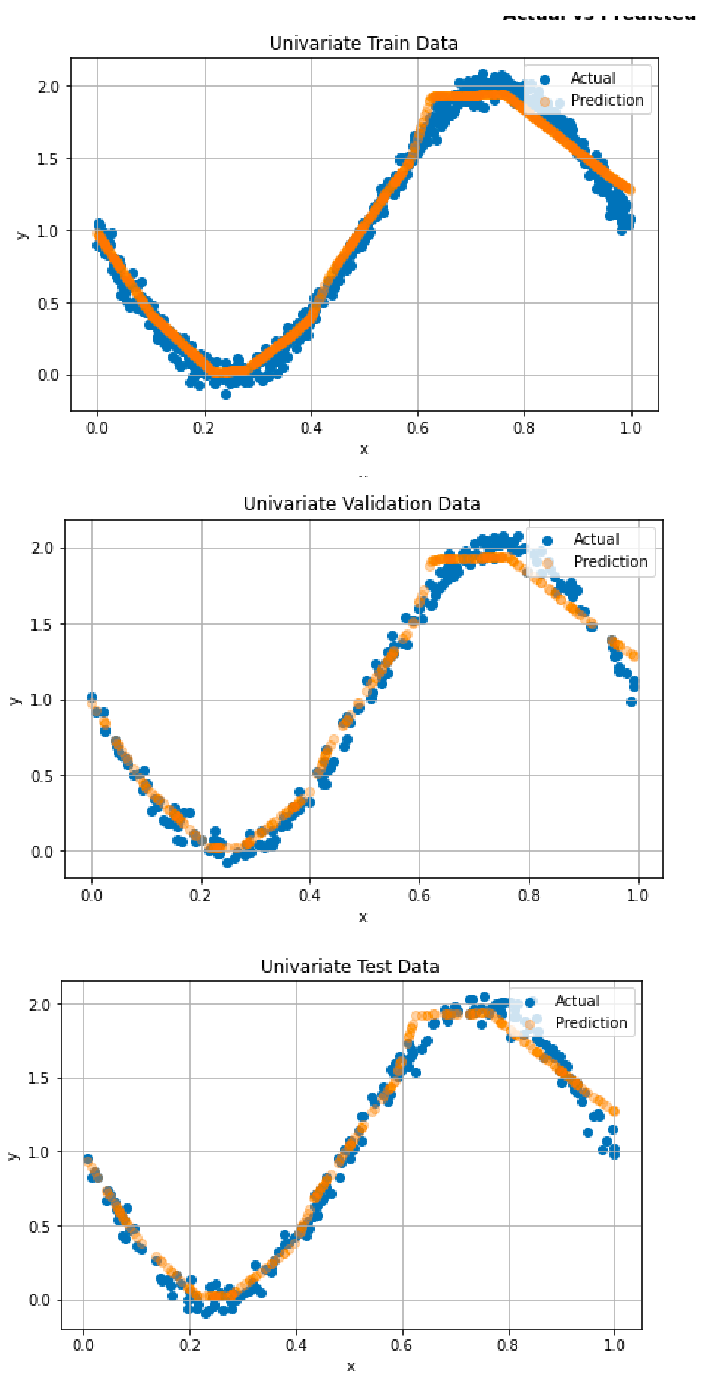


Fig. 4.9: Actual vs predicted surface for univariate

4.2.5 FCNN results on bivariate dataset

FCNN works in a similar way as mentioned in the previous sections for this dataset as well.

1. **Error vs Epochs** : We can see that the training and validation losses are decreasing. Just like in the previous case of univariate data, the error significantly lowers over the range of epochs. This is because the non linear surface can closely capture the non-linearity of the data. This will be more clear when we discuss the surface that is being fit by the model.

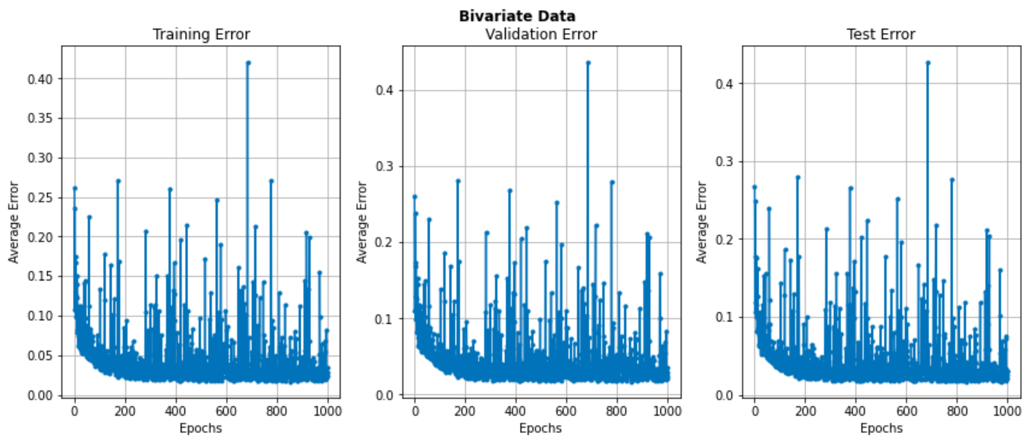


Fig. 4.10: Error v/s epochs (1 hidden)

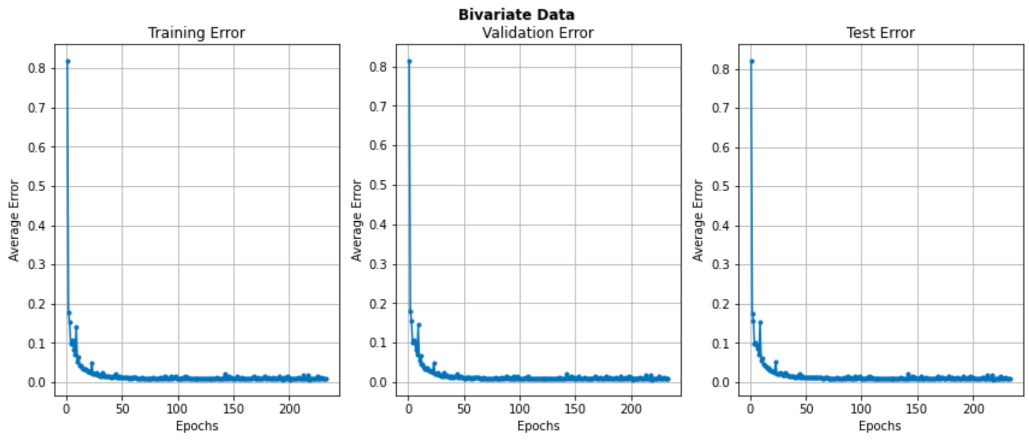


Fig. 4.11: Error v/s epochs (2 hidden)

2. **MSE vs datasets** : As in the previous case of univariate data, we got a similar MSE for training and validation, but for testing, it's high. This is because the model has

seen the former two dataset while the latter is unseen. Since the FCNN is non linear regressor, it is better able to capture the surface of training data and thus is giving lower values of mean squared error than the Perceptron.

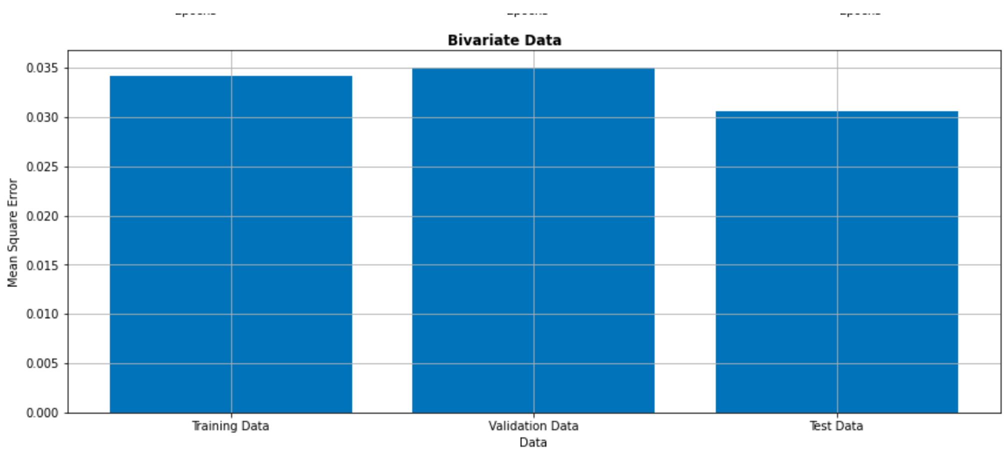


Fig. 4.12: MSE of the datasets by the trained model (1 hidden)

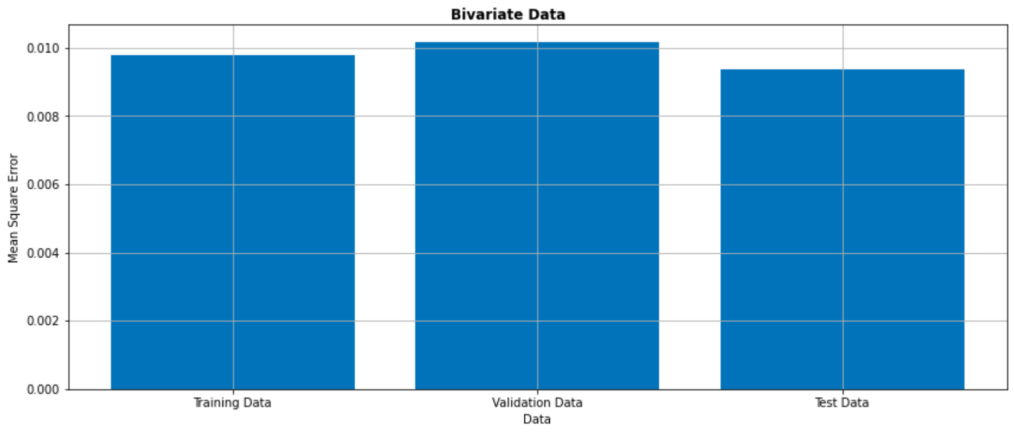


Fig. 4.13: MSE of the datasets by the trained model (2 hidden)

3. **Output data vs predicted data** :

   Since the output and the input are correctly predicted, the plot will be a diagonal line. This is because both y and x are almost same.

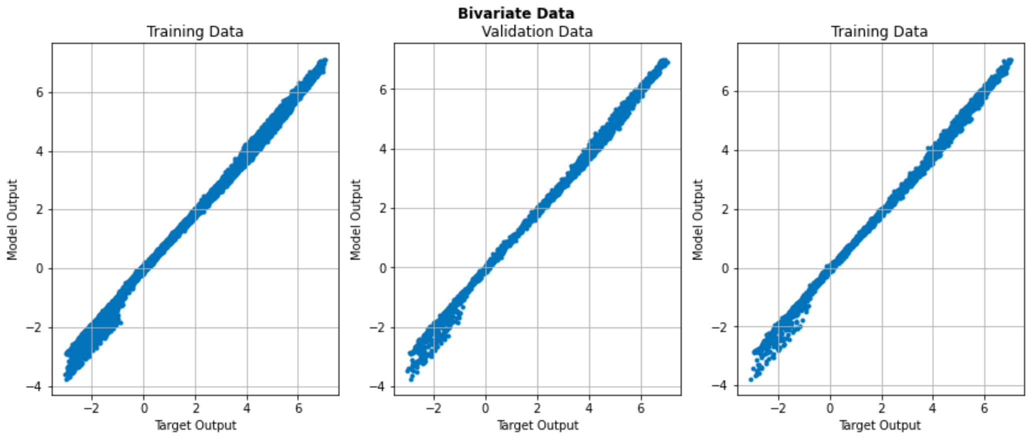


Fig. 4.14: Actual vs predicted for bivariate (1 hidden)

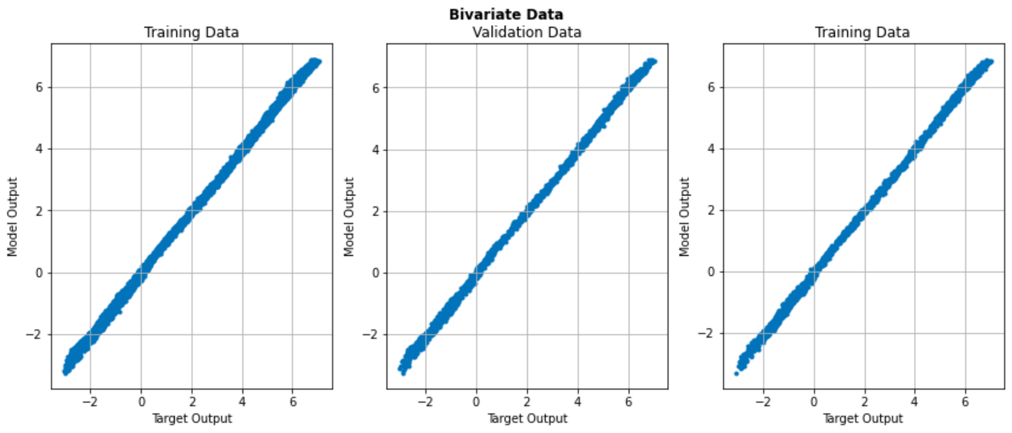


Fig. 4.15: Actual vs predicted for bivariate (2 hidden)

4. **Fitted Surface** : We can observe that FCNN could accuratley estimate the surface of bivariate data. The non linearity of the FCNN is because of the activation functions(ReLU in our case). So the predicted boundary largely coincides with the output actual boundary.

4.2.6 Output Nodes

We can see how the output nodes is able to predict the values.

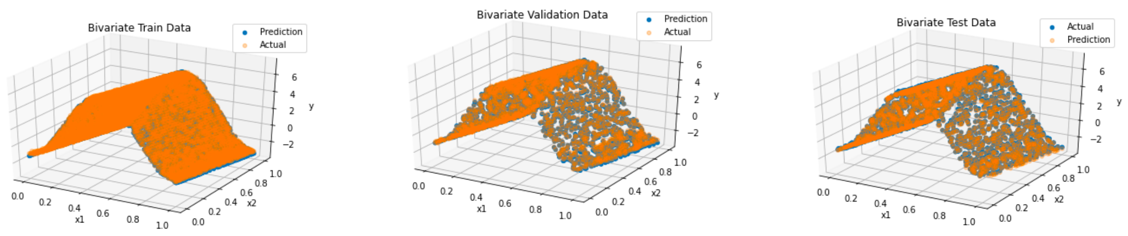


Fig. 4.16: Predicted surface for FCNN (1 hidden)

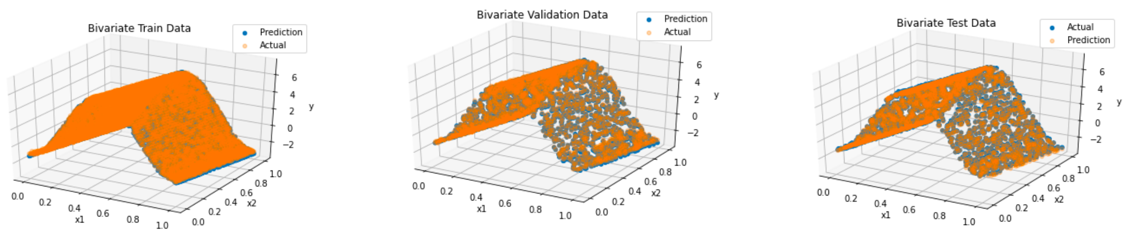


Fig. 4.17: Predicted surface for FCNN (2 hidden)

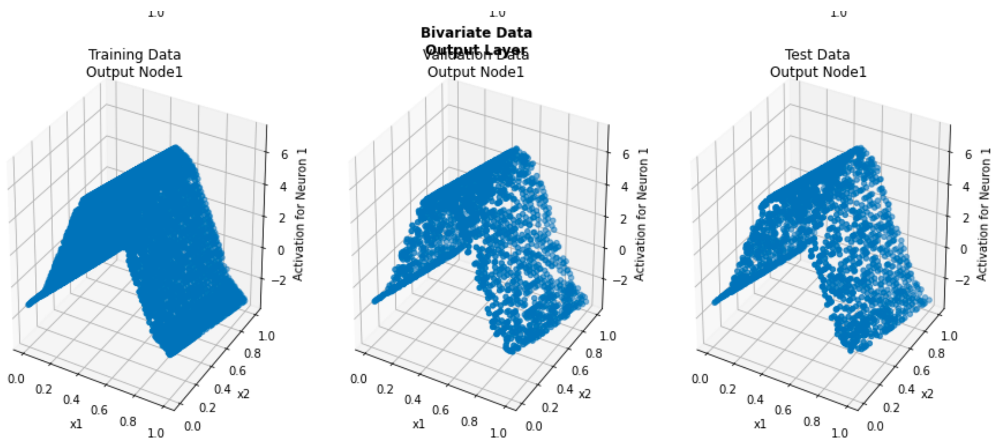


Fig. 4.18: Output node's output

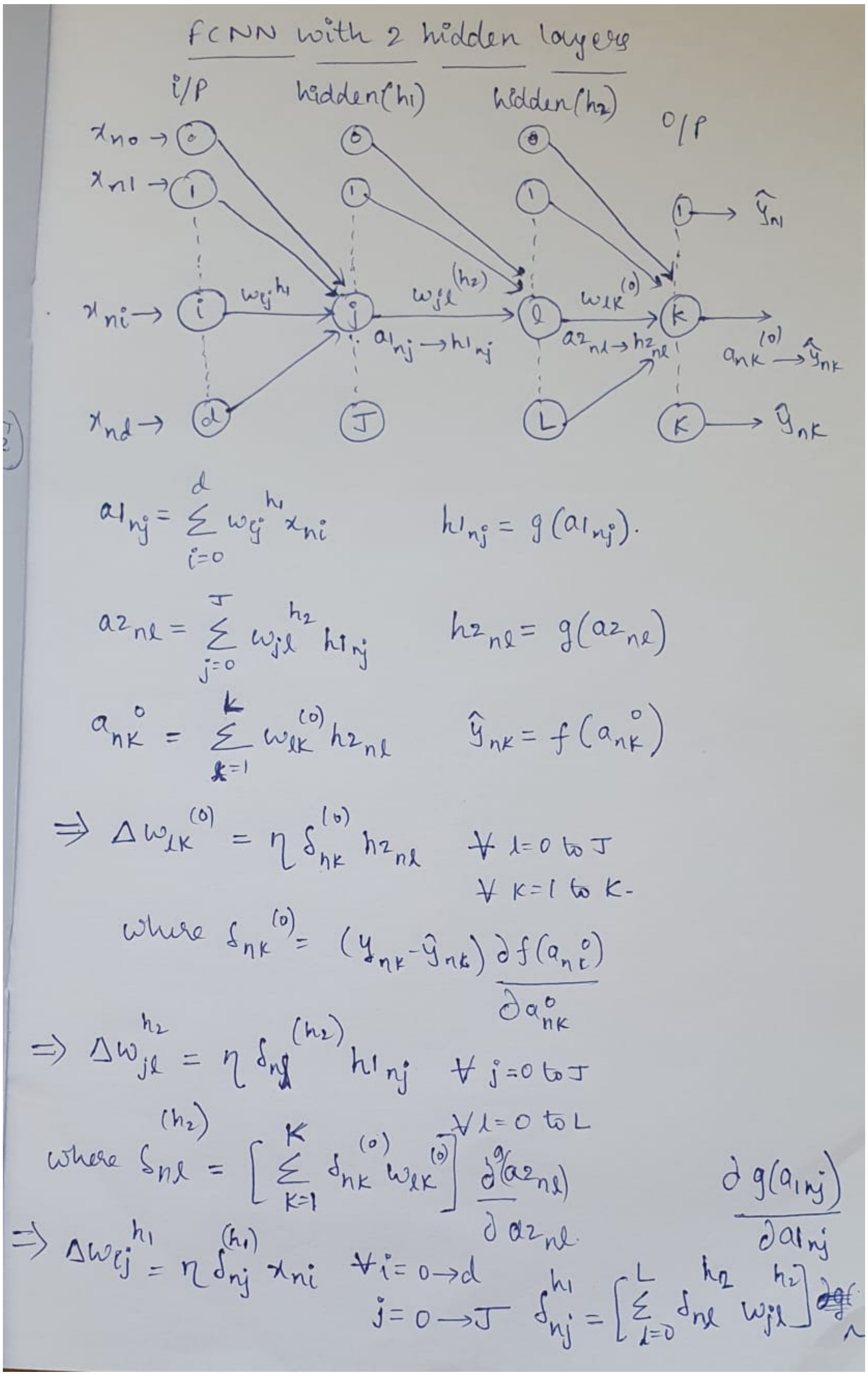
4.3 Comparison

1. Comparison for Classification: We can observe that the FCNN model and FCNN(both 1 and 2 hidden layers) have done equally good with an accuracy of 1 for linearly separable data(for training as well as testing). Coming to non linearly separable data, FCNN has poor performance (accuracy only around 0.6), because of its linear decision surface. On the other hand, FCNN with one and two hidden layers has performed considerably well because of its non linear decision boundary with an accuracy of almost 1.

2. Comparison for Regression: For univariate as well as bivariate data, FCNN failed to predict the output variable because it can produce only linear classifiers which is evident from the line in univariate and plane in bivariate.Hence it has caused many deviations from the output. However, FCNN is better able to predict the output for both univariate and bivariate because of the non linearity introduced by the activation functions. Hence we observe that mean squared error is very less compared to percetron.

Chapter 5

Discussion


FCNN with 2 hidden layers
i/p
hidden(h_1)
hidden(h_2)
o/p
$x_{n0} \rightarrow$
$x_{n1} \rightarrow$
$x_{ni} \rightarrow$
$x_{nd} \rightarrow$
$w_{ij}^{h_1}$
$w_{jl}^{(h_2)}$
$w_{lk}^{(o)}$
$a1_{nj} \rightarrow h1_{nj}$
$a2_{nl} \rightarrow h2_{nl}$
$a_{nk}^{(o)} \rightarrow \hat{y}_{nk}$
$\hat{y}_{n1}$
$\hat{y}_{nK}$
0
1
i
d
j
J
l
L
k
K

$a1_{nj} = \sum_{i=0}^{d} w_{ij}^{h_1} x_{ni}$ $\qquad h1_{nj} = g(a1_{nj})$.

$a2_{nl} = \sum_{j=0}^{J} w_{jl}^{h_2} h1_{nj}$ $\qquad h2_{nl} = g(a2_{nl})$

$a_{nk}^{o} = \sum_{k=1}^{K} w_{lk}^{(o)} h2_{nl}$ $\qquad \hat{y}_{nk} = f(a_{nk}^{o})$

$\Rightarrow \Delta w_{lk}^{(o)} = \eta \, \delta_{nk}^{(o)} h2_{nl}$ $\quad \forall\, l = 0$ to J

$\forall\, k = 1$ to K.

where $\delta_{nk}^{(o)} = (y_{nk} - \hat{y}_{nk}) \dfrac{\partial f(a_{nk}^{o})}{\partial a_{nk}^{o}}$

$\Rightarrow \Delta w_{jl}^{h_2} = \eta \, \delta_{nl}^{(h_2)} h1_{nj}$ $\quad \forall\, j = 0$ to J

$\forall\, l = 0$ to L

where $\delta_{nl}^{(h_2)} = \left[\sum_{k=1}^{K} \delta_{nk}^{(o)} w_{lk}^{(o)} \right] \dfrac{\partial g(a2_{nl})}{\partial a2_{nl}}$

$\Rightarrow \Delta w_{ij}^{h_1} = \eta \, \delta_{nj}^{(h_1)} x_{ni}$ $\quad \forall\, i = 0 \rightarrow d$

$j = 0 \rightarrow J$ $\quad \delta_{nj}^{h_1} = \left[\sum_{l=0}^{L} \delta_{nl}^{h_2} w_{jl}^{h_2} \right]$ ~~∂g~~ $\dfrac{\partial g(a1_{nj})}{\partial a1_{nj}}$

Fig. 5.1: Weights updation

References