Deep Learning & Applications

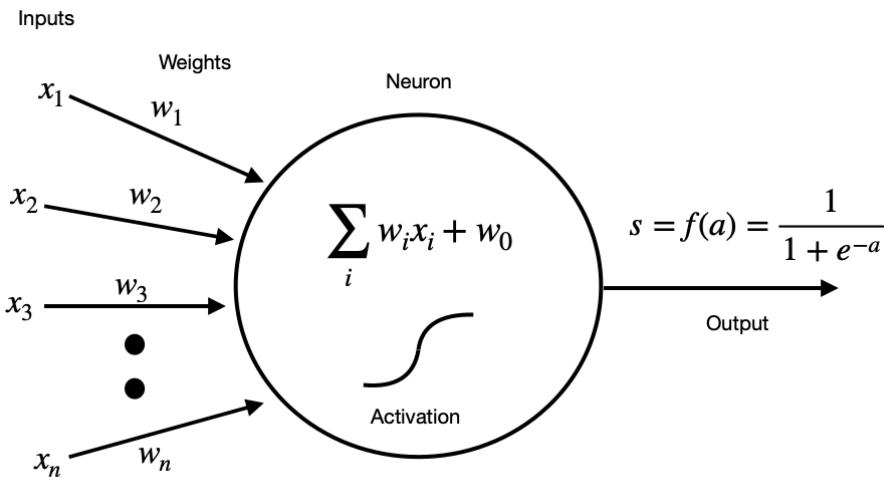
Assignment I

Perceptron & FCNN

CONTENTS

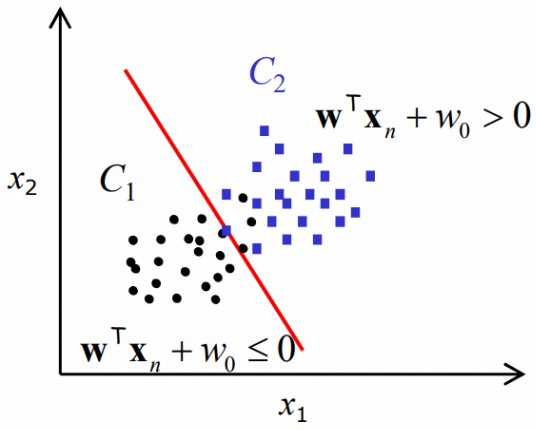
CLASSIFICATION TASKS

PERCEPTRON MODEL



The single neuron (perceptron) model with sigmoidal activation function is shown above. The input vector $\vec{x} = [x_1, x_2, x_3, \ldots . x_n]$ along with its weight $\vec{w} = [w_1, w_2, w_3, \ldots . w_n]$ passed to the neuron. The neuron calculates the dot product $\vec{w}^T . \vec{x}$ and summed with a bias value w_o. The output value is then passed to the sigmoidal activation function, here we used logistic activation function which produces the result in the range of [0, 1]. The details of the activation function used is attached in the appendix at the end of this report.

The neuron can classify two classes, if $\vec{w}^T . \vec{x} + w_0 > 0$, into one and if $\vec{w}^T . \vec{x} + w_0 \leq 0$ into another class as shown in the figure.



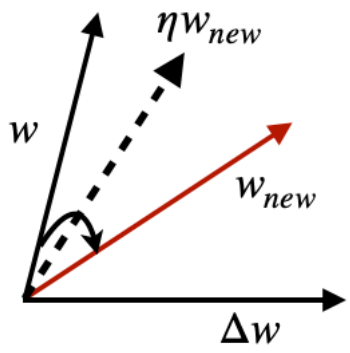
We have to train the hyper plane such that it is placed between the two classes of data so that the error is minimum.

The error in the each sample is calculated by error correction law or loss function (details in the appendix) and the goal is to learn the learning parameter ($\vec{w}$) which is done by minimising the error by gradient descent method.

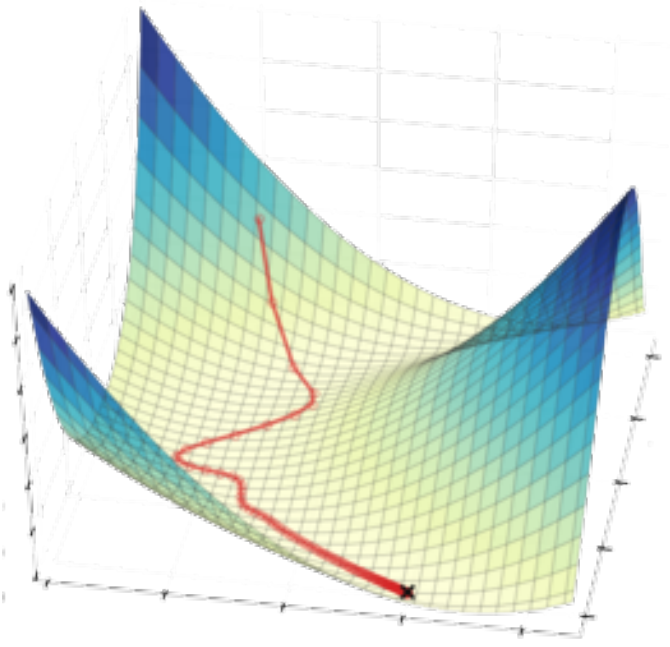
GRADIENT DESCENT ALGORITHM

Gradient descent method allows us to calculate the local minima of a function to estimate the learning parameter faster.

Let the Δw be the change to the direction of convergence. The learning parameter w should move in a direction of Δw creating w_{new} as shown. As we know that we are going to move in Δw direction it is always better to choose small step at a time with the factor η. So we can say that, w has to be updated like:



$$w = w + \eta \Delta w$$



http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html

The Δw should move in the direction opposite to the gradient. Intuitively, it is easy to visualise why it is and to prove mathematically,

$$\Delta w \propto -\frac{\partial E_n}{\partial w}$$

$$\frac{\partial E_n}{\partial w} = \left[\frac{\partial E_n}{\partial w_0} \cdots \frac{\partial E_n}{\partial w_n}\right]^T$$

$$\Delta w = \left[\Delta w_0 \ldots \Delta w_n\right]^T$$

The loss function is detailed in the appendix at the end of the report. The instantaneous error is defined by,

$$E_n = \frac{1}{2}(y_n - s_n)^2$$

$$\frac{\partial E_n}{\partial w} = \frac{1}{2}\frac{\partial}{\partial w}(y_n - s_n)^2$$

$$\frac{\partial E_n}{\partial w} = \frac{1}{2}\frac{\partial}{\partial w}(y_n - s_n)^2 = 2\frac{1}{2}(y_n - s_n)\frac{\partial}{\partial w}(y_n - s_n) = -(y_n - s_n)\frac{\partial s_n}{\partial w}$$

$$= -(y_n - s_n)\frac{\partial f(a_n)}{\partial a_n}\frac{\partial a_n}{\partial w} = -(y_n - s_n)\frac{\partial f(a_n)}{\partial a_n}x_0$$

ALGORITHM (TRAINING):

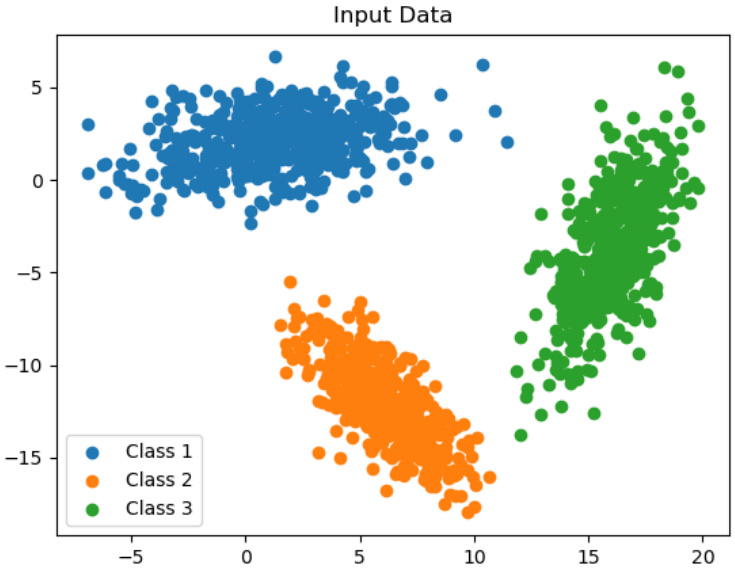
1. *Initialised the learning parameter w with random values*
2. *Chosen single value of x_n from the vector. Here for three class classification $\hat{x}_n = [1, x_n[0], x_n[1]]$ where $x_n \in \mathbb{R}^2$*
3. *Computation of the output of the neuron, $a_n = w^T \hat{x}_n$*
4. *Computation of instantaneous error, $E_n = \frac{1}{2}(y_n - s_n)^2$*
5. *Updation of weights w: $w = w + \eta\Delta w$, where Δw is the change in of the direction opposite to the gradient which is derived above. Thus $\Delta w = -(y_n - s_n)\frac{\partial f(a_n)}{\partial a_n}x_0$. The derivative of the activation function is derived in the appendix attached at the end of the report. Finally the updated weight using logistic sigmoidal function is given by: $\Delta w = \eta(y_n - s_n)s_n(1 - s_n)x_0$ where $0 \leq \eta \leq 1$.*
6. *Repeating the steps 2 to 5 for all the training data set: defined by epoch.*
7. *Calculating the average error, $E_{avg} = \frac{1}{N}\sum_{n=1}^{N} E_n$*
8. *Repeating steps 2 to 7 until the convergence criteria: 100 epoch or Average error falls below 10^{-3}*

ALGORITHM (TESTING):

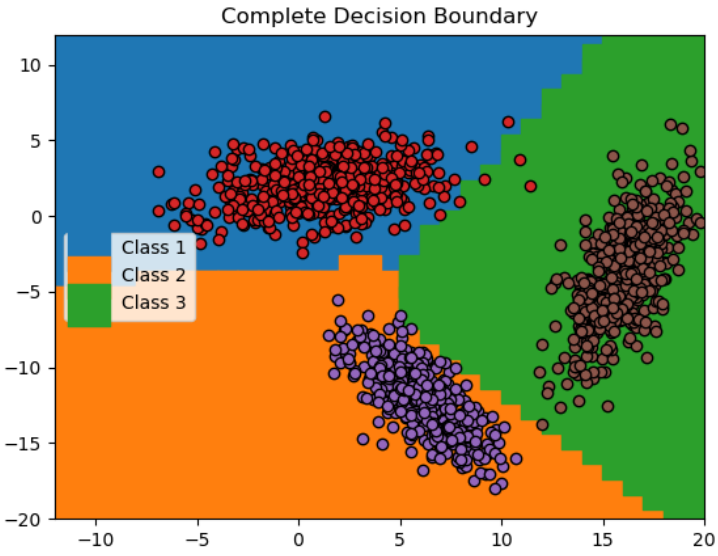
1. *With the computed w, find $a_n = w^T \hat{x}_n$*
2. *Pass the a_n to the logistic sigmoidal activation function*
3. *Then classify to class 1 if the output is less than 0.5 and to class 2 if the output is greater than or equal to 0.5.*
4. *Plotting of the tested data*

LINEAR SEPARABLE DATA RESULTS

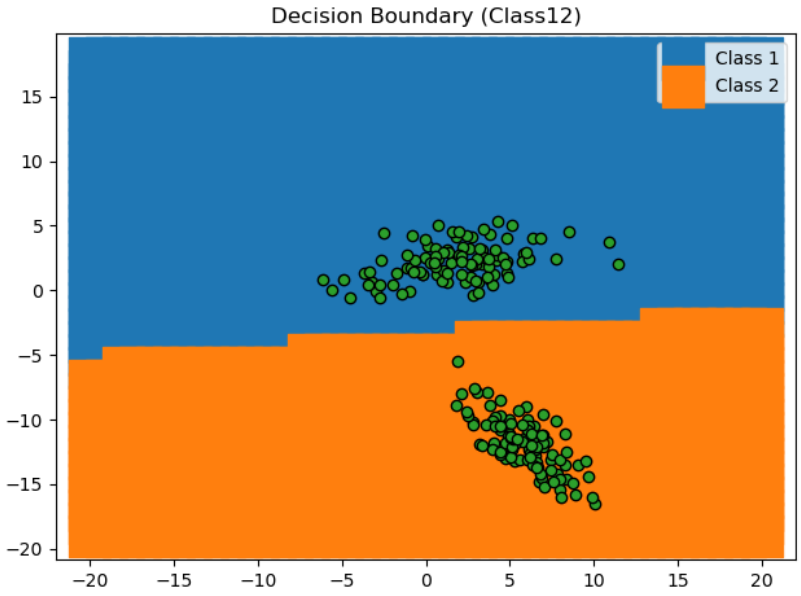
The algorithm is tested for linearly separable data as shown below

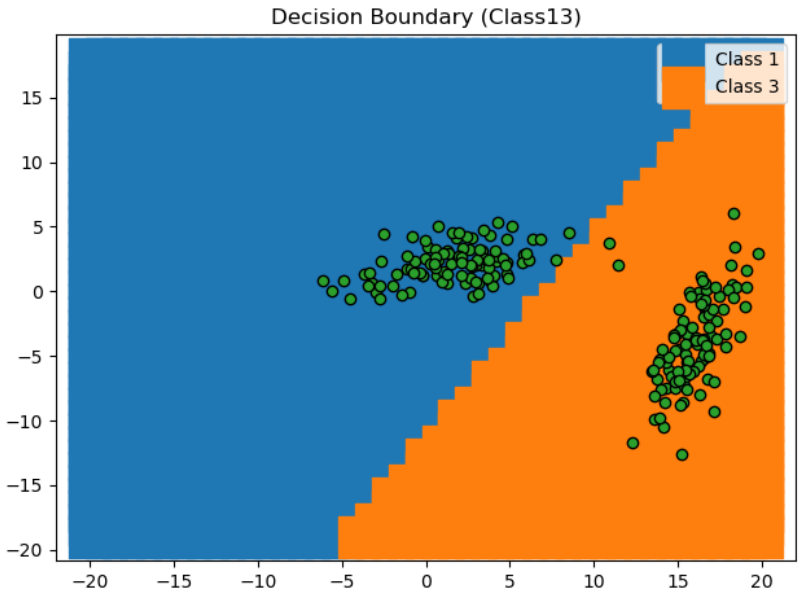


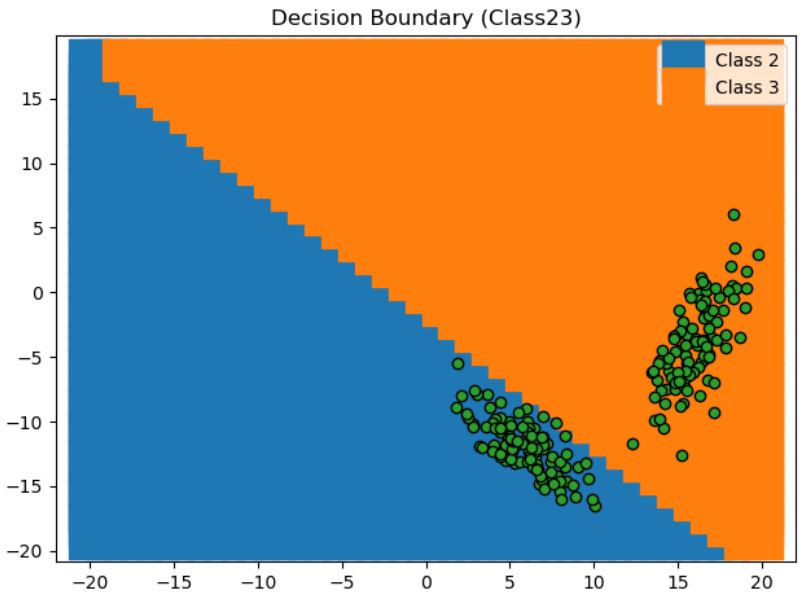
We have classified the data using one to one approach and merged the outputs together. The complete decision boundary is calculated and plotted against the possible inputs and shown.



We have used matplotlib for plotting and coloured the each pixel with size of 500. So some pixel is overlapped and in the border the pixel and representing as misclassified but it is actually not. The confusion matrix and accuracy parameters gives the accurate results obtained.

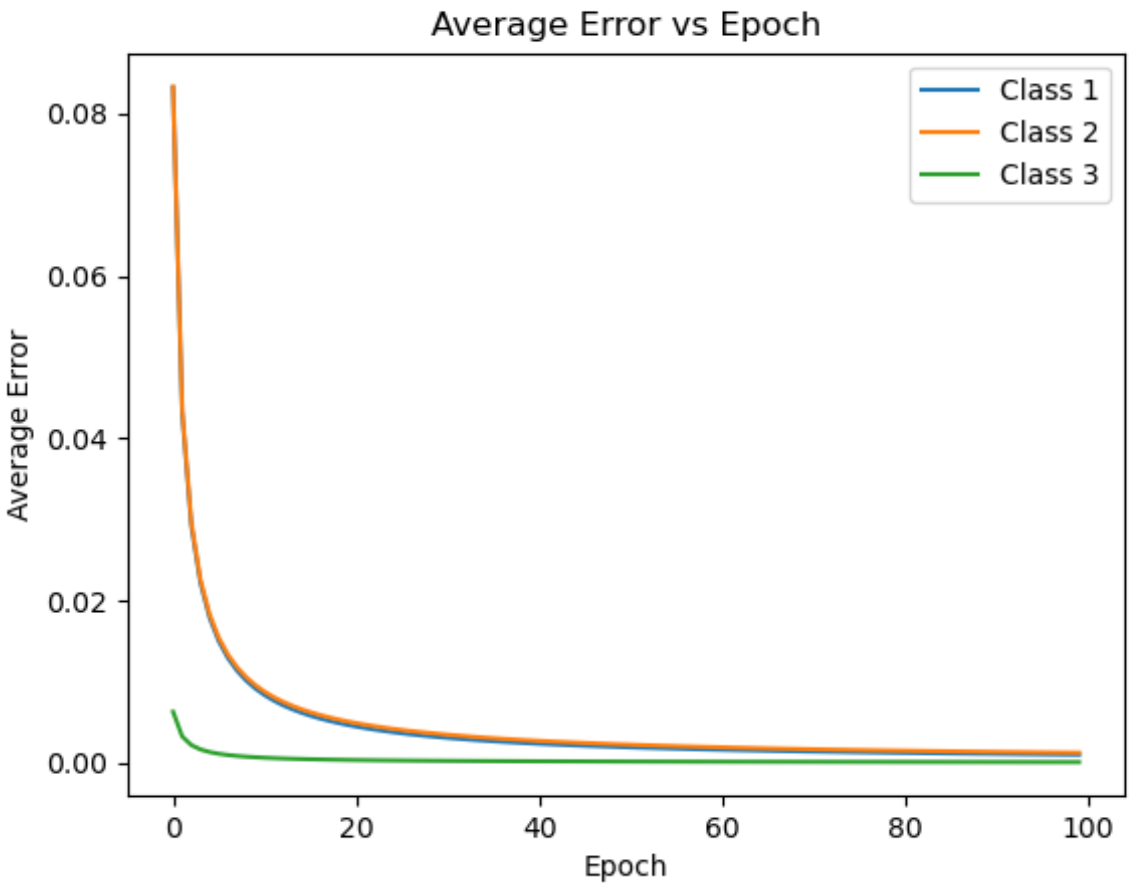






Decision Boundaries of each one to one classes

The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch



ACCURACY PARAMETERS:

1) Confusion Matrix:

```
Confusion Matrix
[[198   0   2]
 [  0 200   0]
 [  0   0 200]]
```

2) Accuracy: 99.6667%

3) Precision: Class1: 98.01% Class2: 100% Class3: 100%
Average: 99.33%

4) Recall: Class1: 98.01% Class2: 100% Class3: 100%
Average: 99.34%

5) F-Measure: Class1: 0.99 Class2: 1.0 Class3: 1.0009
Average: 0.9966

THE CONFUSION MATRIX

ACTUAL CLASS				
Predicted Class		Class 1	Class 2	Class 3
	Class 1	98	0	2
	Class 2	0	100	0
	Class 3	0	0	100

OBSERVATIONS:

- **C11**: 98 test samples predicted as class 1 & actually belongs to class 1
- **C12**: 0 test samples predicted as class 1 & actually belongs to class 2
- **C13**: 2 test samples predicted as class 1 & actually belongs to class 3
- **C21**: 0 test samples predicted as class 2 & actually belongs to class 1
- **C22**: 100 test samples predicted as class 2 & actually belongs to class 2
- **C23**: 0 test samples predicted as class 2 & actually belongs to class 3
- **C31**: 0 test samples predicted as class 3 & actually belongs to class 1
- **C32**: 0 test samples predicted as class 3 & actually belongs to class 2
- **C33**: 100 test samples predicted as class 3 & actually belongs to class 3

ACCURACY:

$$Accuracy = \frac{\text{Number of samples correctly classified (C11+C22+C33)}}{\text{Total number of samples used for testing}} * 100$$

Accuracy % = (98+100+100)/300 * 100 = 99.667%

PRECISION:

"*Number of samples correctly classified as positive class, out of all the examples classified as positive class*"

$$\text{Class 1 prescision} = \frac{\text{TP for Class 1}}{\text{TP for Class 1 + FP for class 1}}$$

$$\text{Class 1 prescision} = \frac{98}{98+2} = 98.01\%$$

$$\text{Class 2 prescision} = \frac{\text{TP for Class 2}}{\text{TP for Class 2 + FP for class 2}}$$

$$\text{Class 2 prescision} = \frac{100}{0+0} = 100\%$$

$$\text{Class 3 prescision} = \frac{\text{TP for Class 3}}{\text{TP for Class 3 + FP for class 3}}$$

$$\text{Class 3 prescision} = \frac{100}{0+0} = 100\%$$

$$\textbf{Mean prescision} = \frac{1}{3}\sum_{i=1}^{3} Presision_i = \frac{98+100+100}{3} = 99.33\%$$

RECALL:

"Number of samples correctly classified as positive class, out of all the examples belonging to positive class"

$$\text{Class 1 Recall} = \frac{\text{TP for Class 1}}{\text{TP for Class 1 + FN for class 1}}$$

$$\text{Class 1 Recall} = \frac{98}{98+2} = 98.01\%$$

$$\text{Class 2 Recall} = \frac{\text{TP for Class 2}}{\text{TP for Class 2 + FP for class 2}}$$

$$\text{Class 2 Recall} = \frac{100}{0+0} = 100\%$$

$$\text{Class 3 Recall} = \frac{\text{TP for Class 3}}{\text{TP for Class 3 + FN for class 3}}$$

$$\text{Class 3 Recall} = \frac{100}{0+0} = 100\%$$

$$\textbf{Mean recall} = \frac{1}{3}\sum_{i=1}^{3} Recall_i = \frac{98 + 100 + 100}{3} = 99.34\%$$

F Measure:

"Harmonic mean of precision and recall"

$$\text{Class 1 F measure} = 2\frac{\text{Precision}_1 * \text{Recall}_1}{\text{Precision}_1 + \text{Recall}_1}$$

$$\text{Class 1 F measure} = 2\frac{0.98 * 0.98}{0.98 + 0.98} = 0.999$$

$$\text{Class 2 F measure} = 2\frac{\text{Precision}_2 * \text{Recall}_2}{\text{Precision}_2 + \text{Recall}_2}$$

$$\text{Class 2 F measure} = 2\frac{1 * 1}{1 + 1} = 1$$

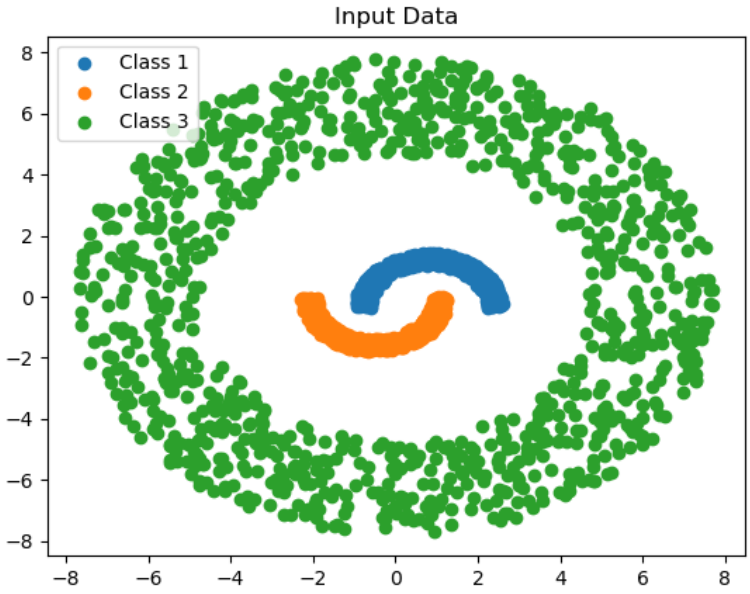
$$\text{Class 3 F measure} = 2\frac{\text{Precision}_3 * \text{Recall}_3}{\text{Precision}_3 + \text{Recall}_3}$$

$$\text{Class 3 F measure} = 2\frac{1 * 1}{1 + 1} = 1$$

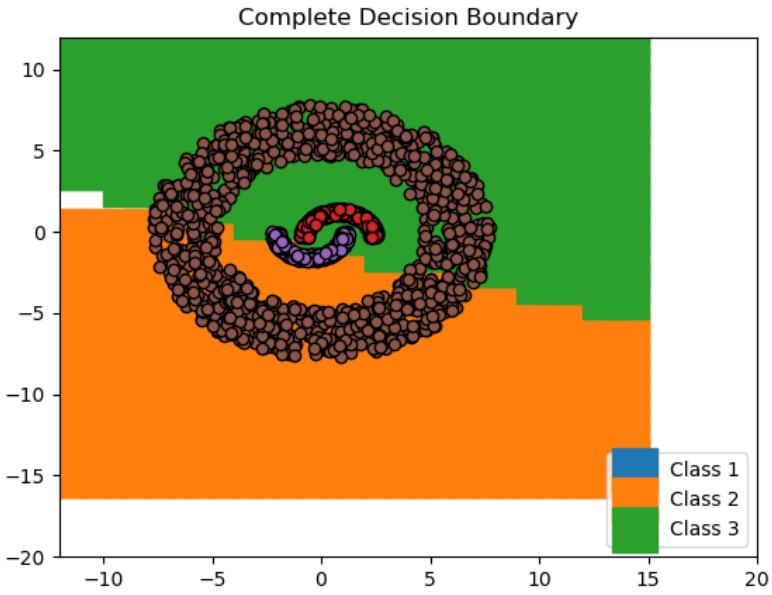
$$\textbf{Mean Fmeasure} = \frac{1}{3}\sum_{i=1}^{3} Fmeasure_i = \frac{0.99 + 1 + 1}{3} = 0.99$$

NON-LINEAR DATA RESULTS

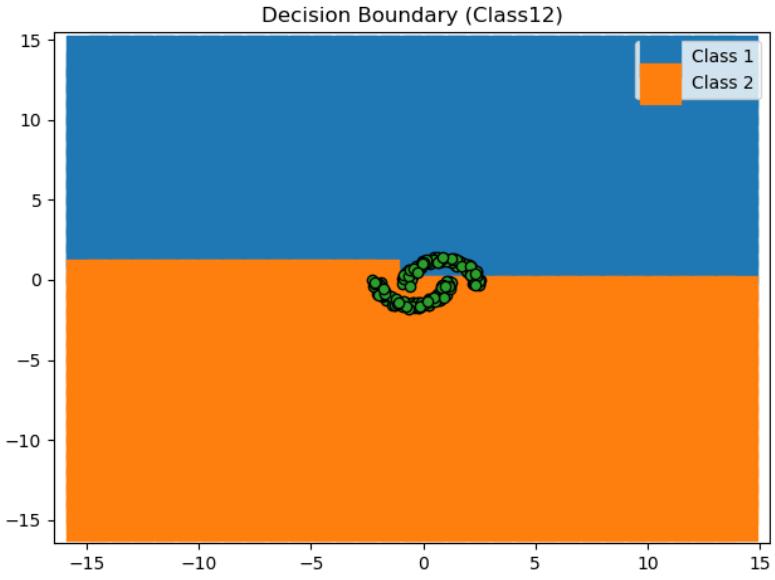
The algorithm is tested for Non linearly separable data as shown below

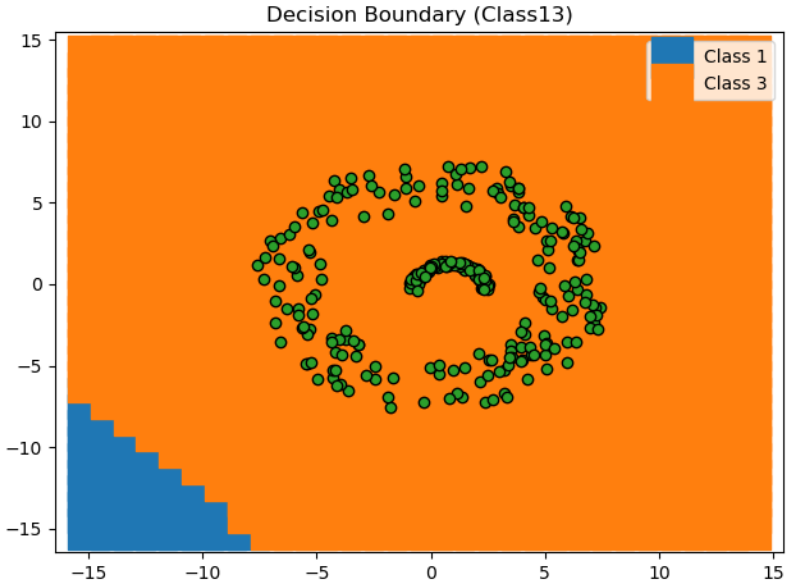


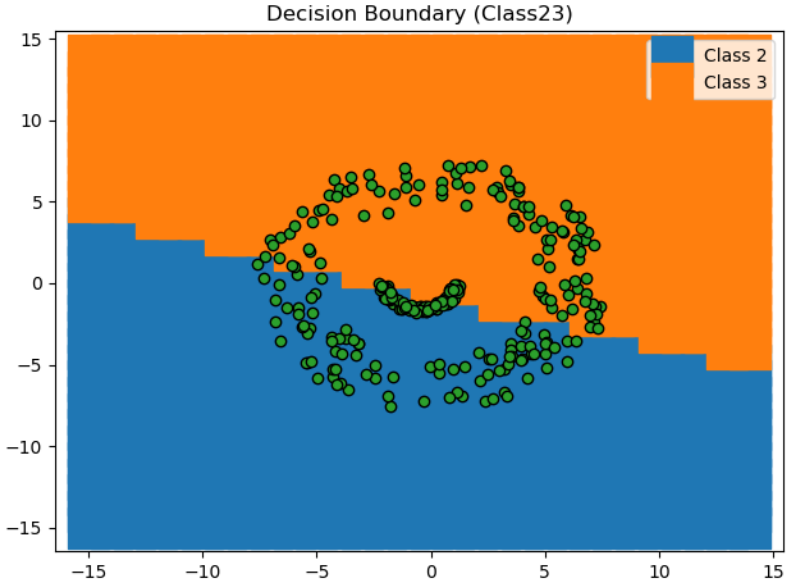
We have classified the data using one to one approach and merged the outputs together. The complete decision boundary is calculated and plotted against the possible inputs and shown.



As we may can observe that the perceptron model works very poor for the non linear separable data.

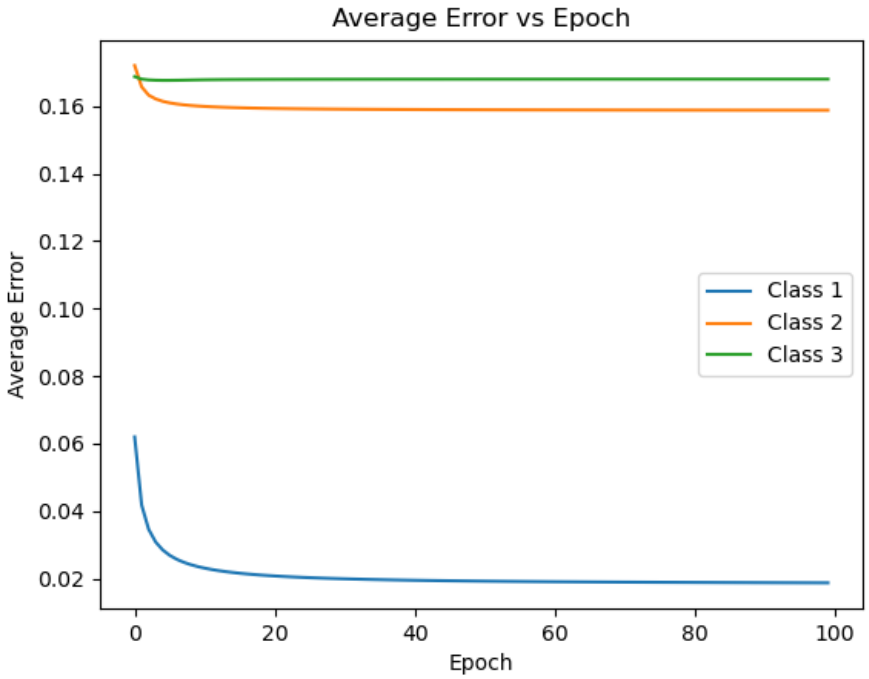






Decision Boundaries of each one to one classes

The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch



ACCURACY PARAMETERS:

1) Confusion Matrix:

```
Confusion Matrix
[[  91    9  100]
 [   1  188   12]
 [-101   -2  304]]
```

2) Accuracy: 42.512%

3) Precision: Class1: 0.294 Class2: 0.8785 Class3: 3.10
Average: 1.425

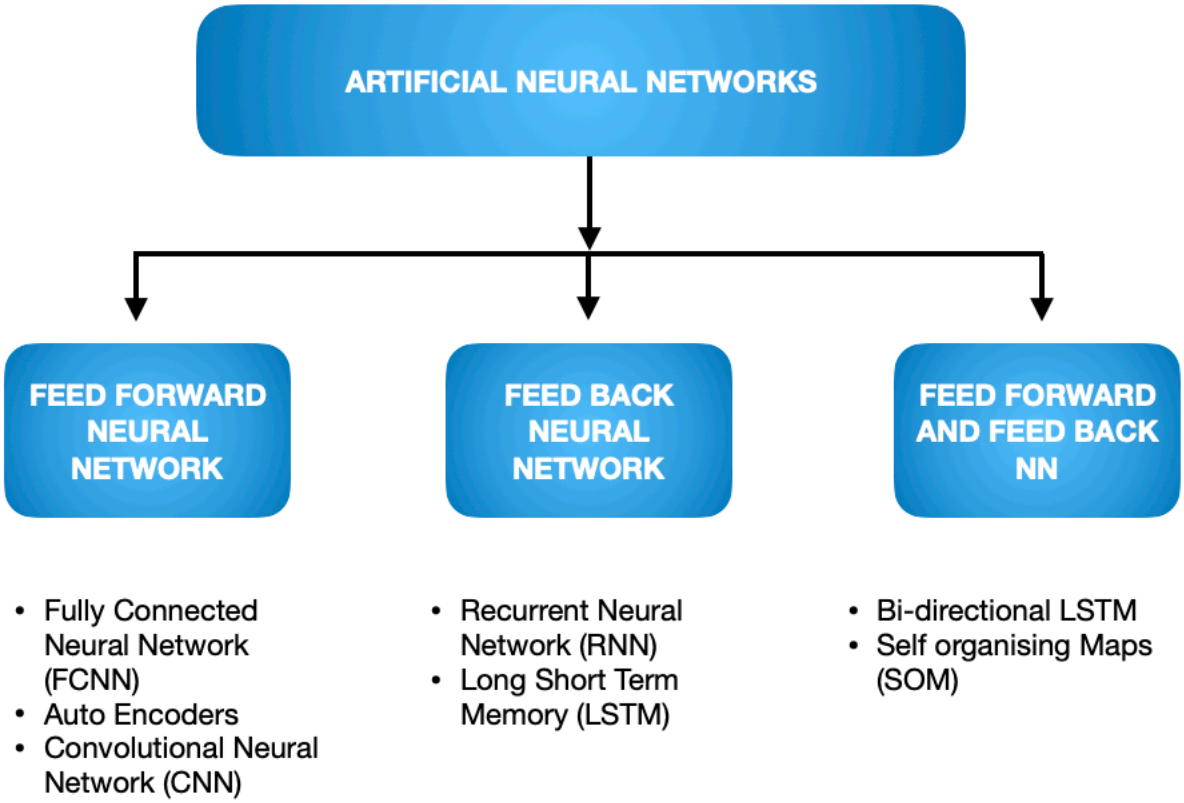
4) Recall: Class1: 0.294 Class2: 0.8785 Class3: 3.10
Average: 0.223

5) F-Measure: Class1: 0.91 Class2: 0.93 Class3: 4.104
Average: 1.972

THE CONFUSION MATRIX

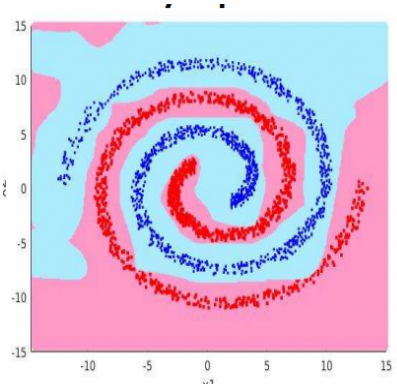
ACTUAL CLASS				
Predicted Class		Class 1	Class 2	Class 3
	Class 1	91	9	100
	Class 2	1	188	12
	Class 3	101	2	304

FULLY CONNECTED NEURAL NETWORK (FCNN) MODEL



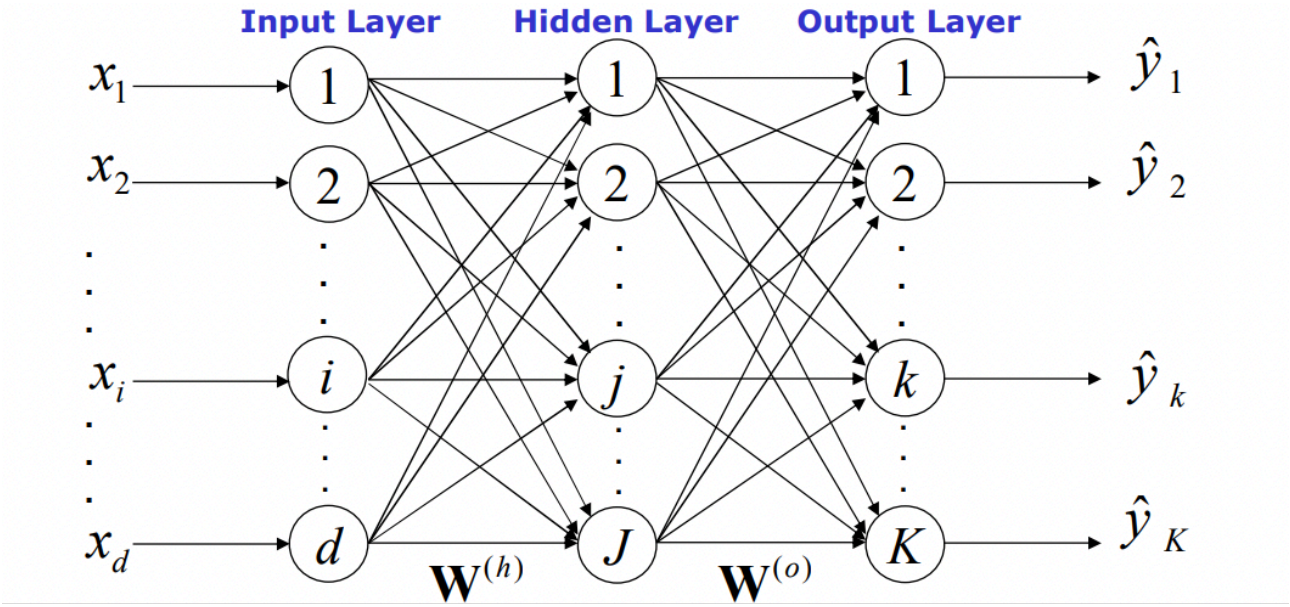
For Nonlinear data classification or regression single neuron model is not sufficient. It becomes very hard to decide the boundaries of the predictions. The spiral data is shown as an example, where the single neuron model fails.

Such hard problems require a series of neuron so that can approximate a nonlinear surface by stitching the boundary/surface from each neuron.



This series of neuron is called as layers. One can use as many as layers to classify non-linearity. The use of two or more layers commonly called hidden layers are called Deep Neural Networks (DNN).
The DNN also requires lot of data to train in order to work properly.

FCNN Architecture with ONE Hidden Layer



- The input layer consists of d (dimension of input vector) neurons which is a linear neuron which simply passes the input as output.
- Hidden layers has the J sigmoidal neurons (like the perceptron).
- Output layer is either linear for regression tasks or sigmoidal neuron for classification tasks.
- Number of layers and neurons in each of the hidden layers are decided experimentally.
- Weights associated with all the connection between the neurons indicate the parameter of the complex nonlinear discriminant function/nonlinear surface that the network is trying to approximate

BACK PROPAGATION ALGORITHM

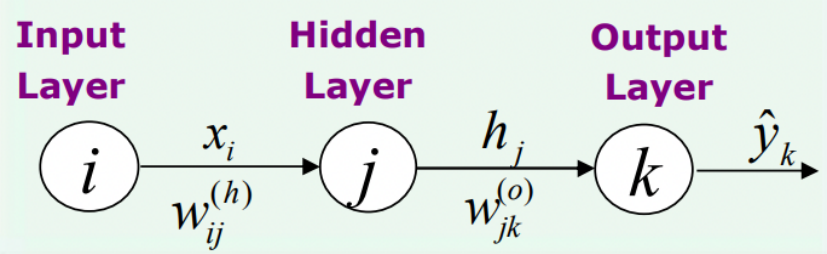
Here the learning parameter $w^{(h)}$ and $w^{(o)}$ can be computed using the Gradient descent method by Minimising the squared error between the output of network and corresponding desired output.

These layer of neurons can be designed by simple matrix $w^{(h)}$ and $w^{(o)}$ as shown and can include as many as hidden layers we want. Many high end linear algebra computations helps us to compute faster.

$$\mathbf{W}^{(h)} = \begin{bmatrix} w_{01}^{(h)} & w_{02}^{(h)} & \dots & w_{0j}^{(h)} & \dots & w_{0J}^{(h)} \\ w_{11}^{(h)} & w_{12}^{(h)} & \dots & w_{1j}^{(h)} & \dots & w_{1J}^{(h)} \\ \hline w_{i1}^{(h)} & w_{i2}^{(h)} & \dots & w_{ij}^{(h)} & \dots & w_{iJ}^{(h)} \\ \hline w_{d1}^{(h)} & w_{d2}^{(h)} & \dots & w_{dj}^{(h)} & \dots & w_{dJ}^{(h)} \end{bmatrix}^T \in \mathbb{R}^{(d+1)\times J} \quad \mathbf{W}^{(o)} = \begin{bmatrix} w_{01}^{(o)} & w_{02}^{(o)} & \dots & w_{0k}^{(o)} & \dots & w_{0K}^{(o)} \\ w_{11}^{(o)} & w_{12}^{(o)} & \dots & w_{1j}^{(o)} & \dots & w_{1K}^{(o)} \\ \hline w_{j1}^{(o)} & w_{j2}^{(o)} & \dots & w_{jk}^{(o)} & \dots & w_{jK}^{(o)} \\ \hline w_{J1}^{(o)} & w_{J2}^{(o)} & \dots & w_{Jk}^{(o)} & \dots & w_{JK}^{(o)} \end{bmatrix}^T \in \mathbb{R}^{(J+1)\times K}$$

STOCHASTIC GRADIENT DESCENT ALGORITHM (TRAINING):

1. *Initialised the learning parameter $w^{(h)}$ and $w^{(o)}$ with random values*
2. *Chosen single value of x_n from the vector. Here for three class classification $\hat{x}_n = [1, x_n[0], x_n[1]]$ where $x_n \in \mathbb{R}^2$*
3. ***Forward Computation**: Output of all the neuron, $a_n = w^T \hat{x}_n$*

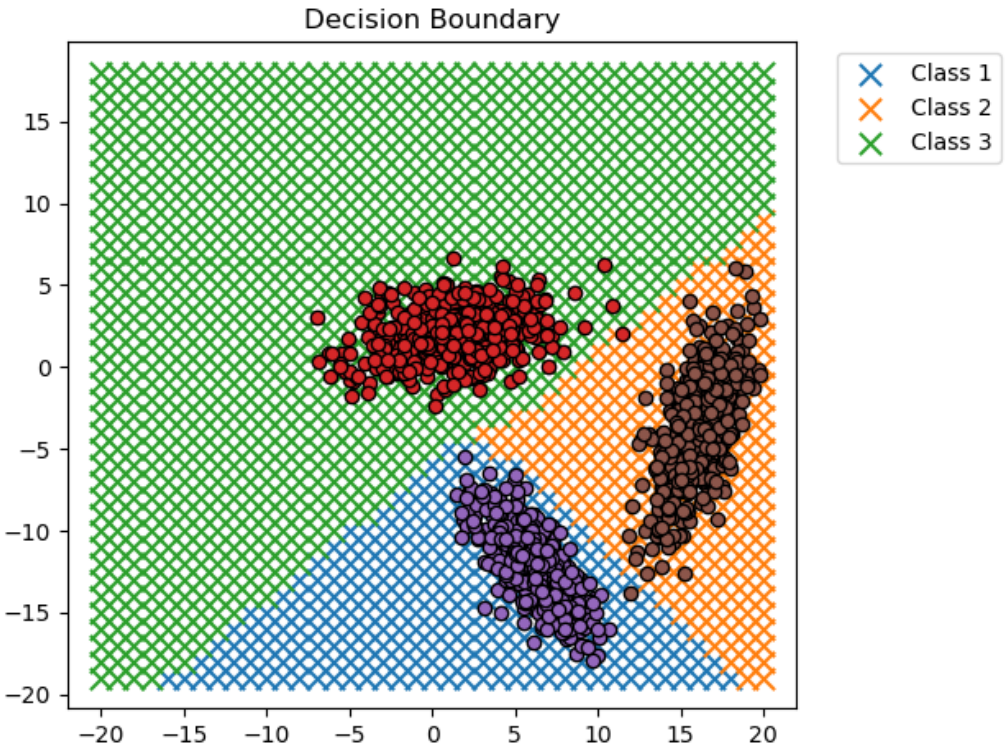


4. ***Backward Computation:** Computation of instantaneous error, $E_n = \frac{1}{2}(y_n - s_n)^2$*
5. *Updation of weights $w^{(h)}$ and $w^{(o)}$: $w_{jk}^{(o)} = w_{jk}^{(o)} + \eta \Delta w_{jk}^{(o)}$ and $w_{ij}^{(h)} = w_{ij}^{(h)} + \eta \Delta w_{ij}^{(h)}$, where $\Delta w_{jk}^{(o)} = \eta(y_{nk} - \hat{y}_{nk})\hat{y}_{nk}(1 - \hat{y}_{nk})h_{nj}$ and $\Delta w_{ij}^{(h)} = \eta \sum_{k=1}^{K} \left((y_{nk} - \hat{y}_{nk})\hat{y}_{nk}(1 - \hat{y}_{nk})\right) w_{jk}^{(o)} g(a_{nj})(1 - g(a_{nj}))x_i$ and $0 \le \eta \le 1$.*
6. *Repeating the steps 2 to 5 for all the training data set: defined by epoch.*
7. *Calculating the average error, $E_{avg} = \frac{1}{N}\sum_{n=1}^{N} E_n$*
8. *Repeating steps 2 to 7 until the convergence criteria: 100 epoch or Average error falls below 10^{-3}*

ALGORITHM (TESTING):

1. *For a test example x compute the output of each of the neurons in output layer ($\hat{y}_k$, k=1,2,… K) using the weights obtained by training the model*
2. $\hat{y}_k = f(a_k^o) = f\left(\sum_{j=0}^{J} w_{jk}^h h_j\right)$, *class label for* $x = argmax\hat{y}_k$

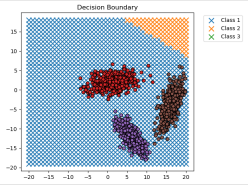
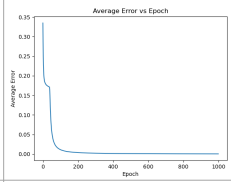
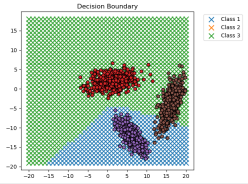
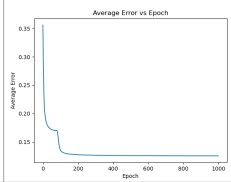
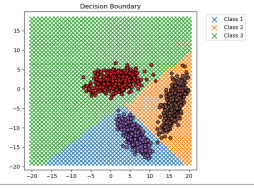
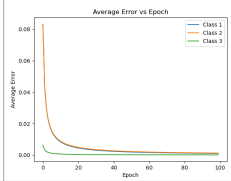
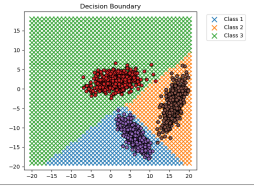
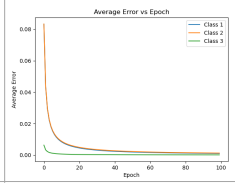
LINEAR SEPARABLE DATA RESULTS



The same linear data is used and the results obtained with one hidden layer FCNN gives 100% accuracy.

The model used 6 neurons in the hidden layer.

VARIOUS MODEL COMPARISON

No of Neurons	Decision Boundary	Error Vs Epoch	Confusion Matrix	Accuracy	Precision	Recall	F-Measure
1			$\begin{bmatrix} 0 & 100 & 0 \\ 0 & 100 & 0 \\ 0 & 100 & 0 \end{bmatrix}$	33.33%	0.333	NaN	0.32
2			$\begin{bmatrix} 99 & 0 & 1 \\ 0 & 44 & 56 \\ 0 & 1 & 99 \end{bmatrix}$	80.66%	0.747	0.807	0.68
5			$\begin{bmatrix} 99 & 0 & 1 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}$	99.67%	0.993	0.9934	0.99
8			$\begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}$	100%	1.00	1	1

ACCURACY PARAMETER (BEST MODEL):

1) Confusion Matrix: $\begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}$

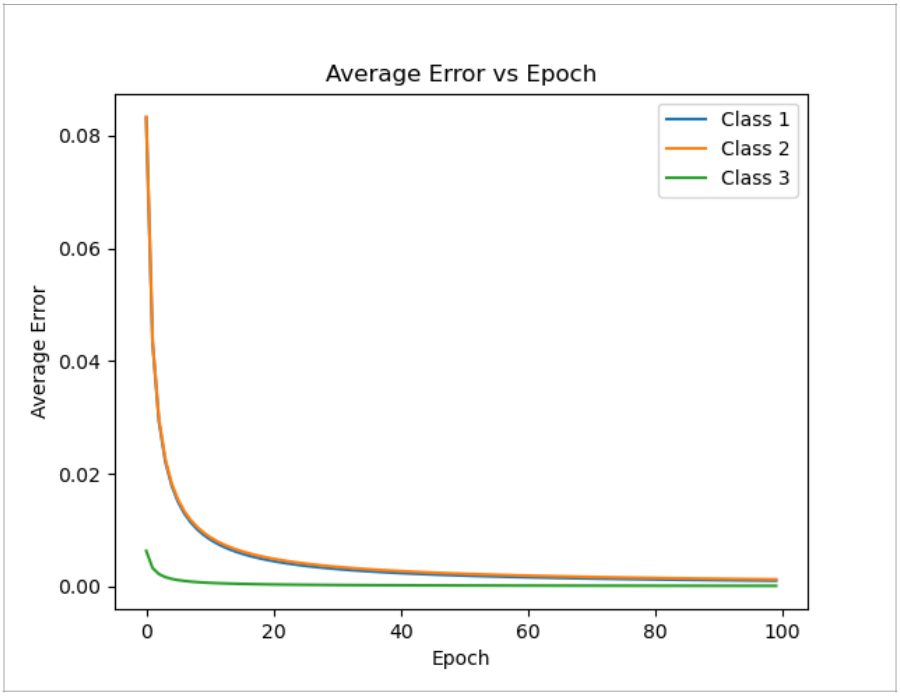
2) Accuracy: 100%

3) Precision: Class1: 100% Class2: 100% Class3: 100%
Average: 100%

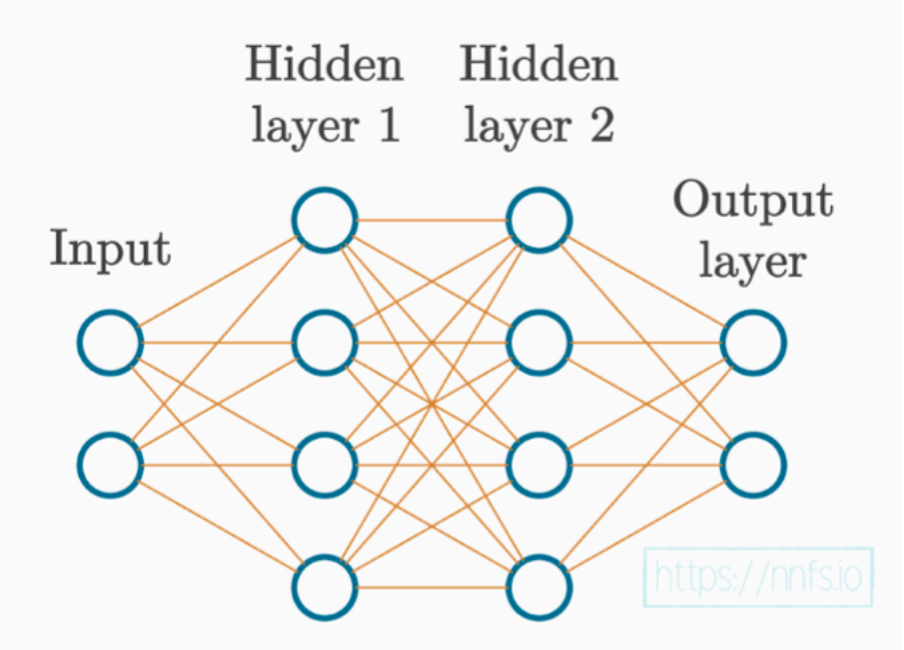
4) Recall: Class1: 100% Class2: 100% Class3: 100%
Average: 100%

5) F-Measure: Class1: 1.0 Class2: 1.0 Class3: 1.0
Average: 1

The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch

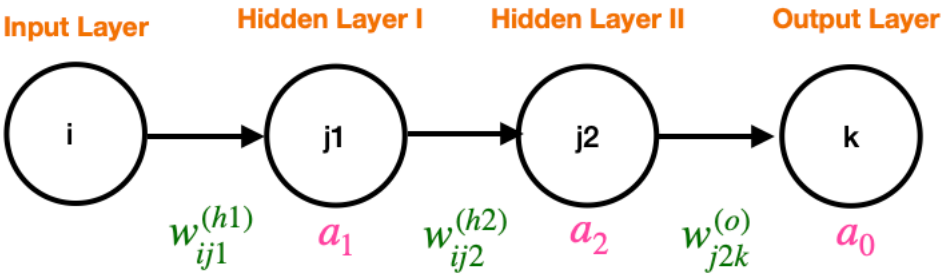


FCNN Architecture with TWO Hidden Layer



https://nnfsio

Increasing the no of layers helps to classify the non linear data more effectively and accurately. So classifying the non linear set, we attempted to increase a one more hidden layer.



All the notations and the algorithm remains same except for the additional layer we have included in between the hidden and output layer.

a_1, a_2 and a_0 are the output of the layers and w's are the learning parameters. Used the same back propagation algorithm with some minor changes in the hidden layer II which is described below

STOCHASTIC GRADIENT DESCENT ALGORITHM (TRAINING):

1. *Initialised the learning parameter* $w^{(h1)}$, $w^{(h2)}$ *and* $w^{(o)}$ *with random values*

2. *Chosen single value of* x_n *from the vector. Here for three class classification* $\hat{x}_n = [1, x_n[0], x_n[1]]$ *where* $x_n \in \mathbb{R}^2$

3. ***Forward Computation****: Output of all the neuron,* $a_n = w^T \hat{x}_n$

4. ***Backward Computation:*** *Computation of instantaneous error,* $E_n = \frac{1}{2}(y_n - s_n)^2$

5. *Updation of weights* $w^{(h1)}$, $w^{(h2)}$ *and* $w^{(o)}$*:*

$w^{(o)} = w^{(o)} + \eta\delta^{(o)}a_2$ *where* $\delta^{(o)} = (y - a_0)a_0(1 - a_0)$

$w^{(h2)} = w^{(h2)} + \eta\delta^{(h2)}a_1$ *where* $\delta^{(h2)} = \left(\sum w^{(o)}\delta^{(o)}\right) a_2(1 - a_2)$

$w^{(h1)} = w^{(h1)} + \eta\delta^{(h1)}x_i$ *where* $\delta^{(h1)} = \left(\sum w^{(h2)}\delta^{(h2)}\right) a_1(1 - a_1)$

6. *Repeating the steps 2 to 5 for all the training data set: defined by epoch.*

7. *Calculating the average error,* $E_{avg} = \frac{1}{N}\sum_{n=1}^{N} E_n$

8. *Repeating steps 2 to 7 until the convergence criteria: 100 epoch or Average error falls below* 10^{-3}

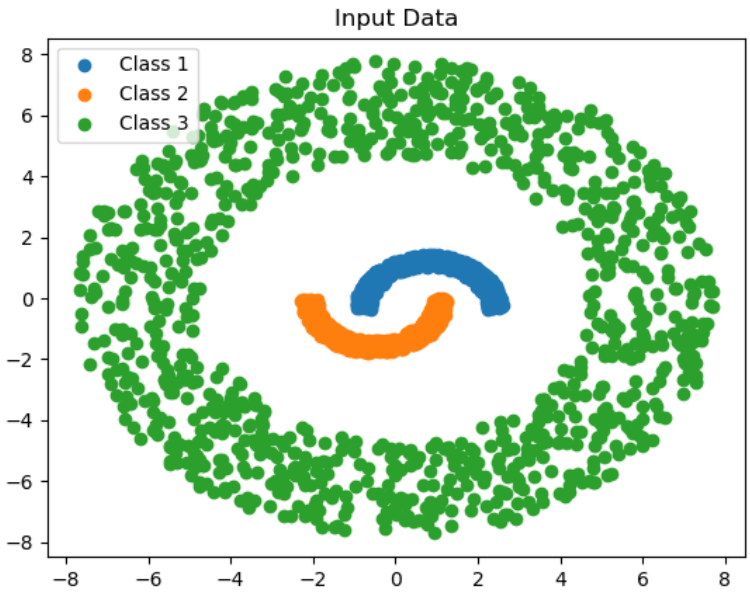
ALGORITHM (TESTING):

1. *For a test example x compute the output of each of the neurons in output layer (* $\hat{y}_k$*, k=1,2,… K) using the weights obtained by training the model*

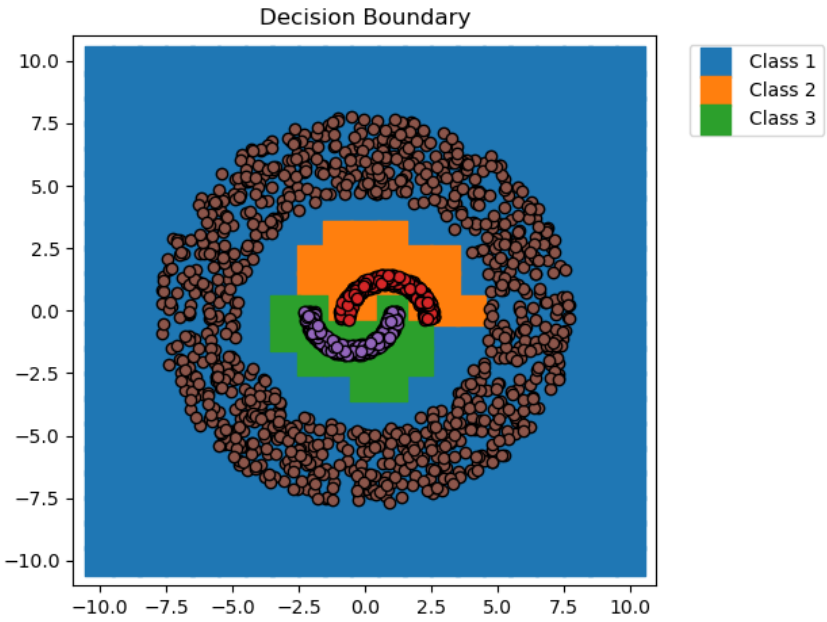
2. $\hat{y}_k = f(a_k^o) = f\left(\sum_{j=0}^{J} w_{jk}^h h_j\right)$*, class label for* $x = argmax\hat{y}_k$

NON LINEAR SEPARABLE DATA RESULTS

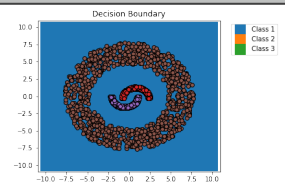
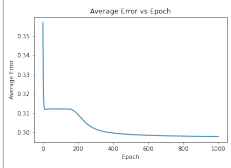
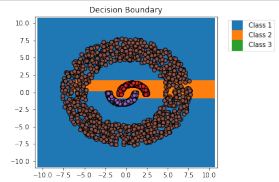
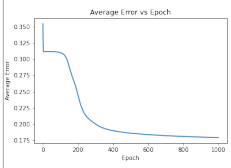
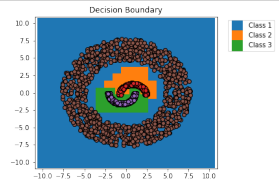
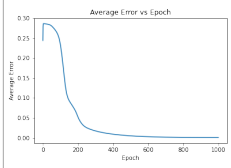
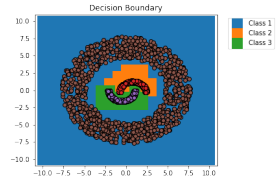
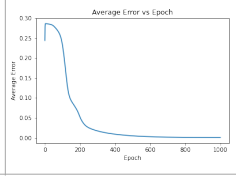
The algorithm is tested for Non linearly separable data as shown below



The data is classified using two hidden nodes with each of 16 neurons and the result is obtained.



VARIOUS MODEL COMPARISON

No of Neurons	Decision Boundary	Error Vs Epoch	Confusion Matrix	Accuracy	Precision	Recall	F-Measure
1			$\begin{bmatrix} 0 & 0 & 100 \\ 0 & 0 & 101 \\ 0 & 0 & 201 \end{bmatrix}$	50%	0.333	NaN	0.32
2			$\begin{bmatrix} 93 & 0 & 7 \\ 3 & 0 & 98 \\ 12 & 9 & 180 \end{bmatrix}$	67.91%	0.559	0.405	0.68
5			$\begin{bmatrix} 94 & 5 & 1 \\ 3 & 98 & 0 \\ 0 & 0 & 201 \end{bmatrix}$	97.76%	0.943	0.886	0.96
16			$\begin{bmatrix} 100 & 0 & 0 \\ 0 & 101 & 0 \\ 0 & 0 & 201 \end{bmatrix}$	100%	1.00	1	1

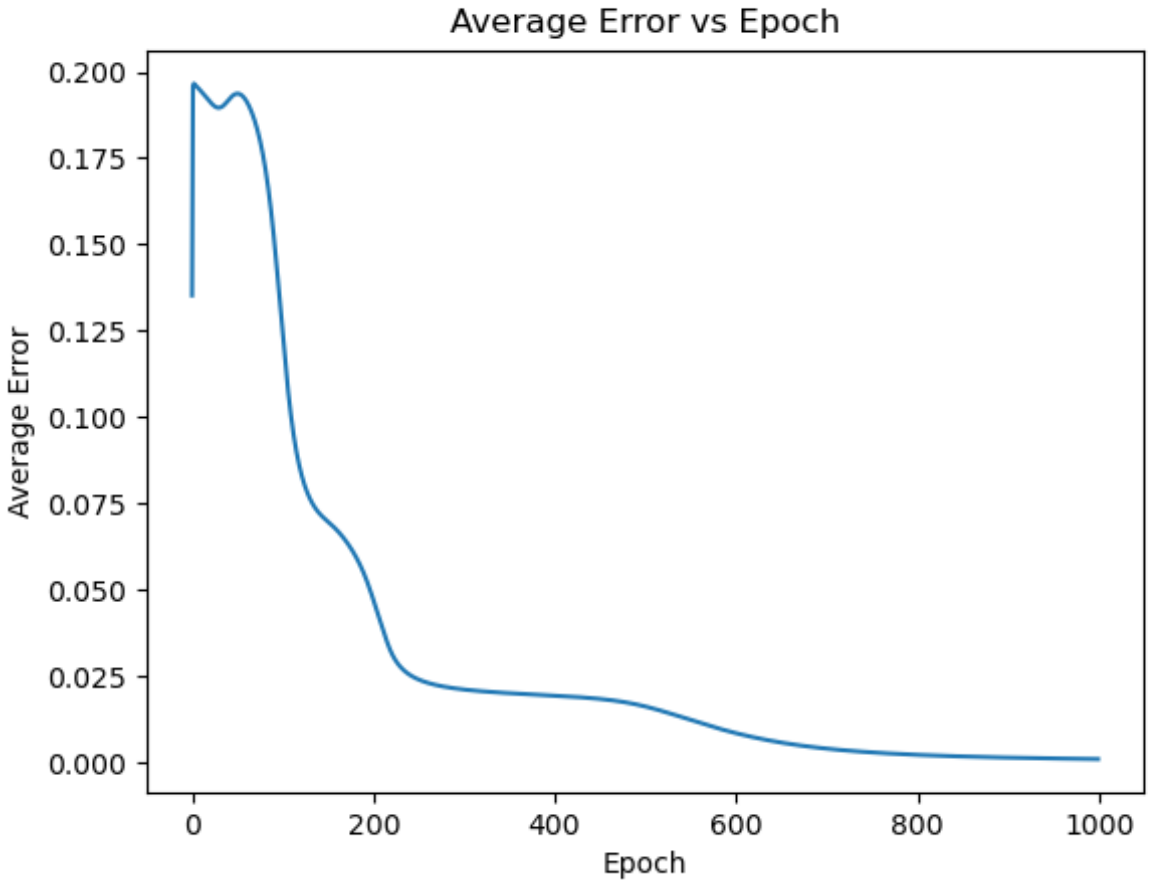
BEST OF ALL (16 Neurons)

THE CONFUSION MATRIX-1

ACTUAL CLASS				
Predicted Class		Class 1	Class 2	Class 3
	Class 1	100	0	0
	Class 2	0	101	0
	Class 3	0	0	201

2) Accuracy: 100%
3) Precision: 1.0
4) Recall: 1.0
5) F-Measure: 1.0

The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch

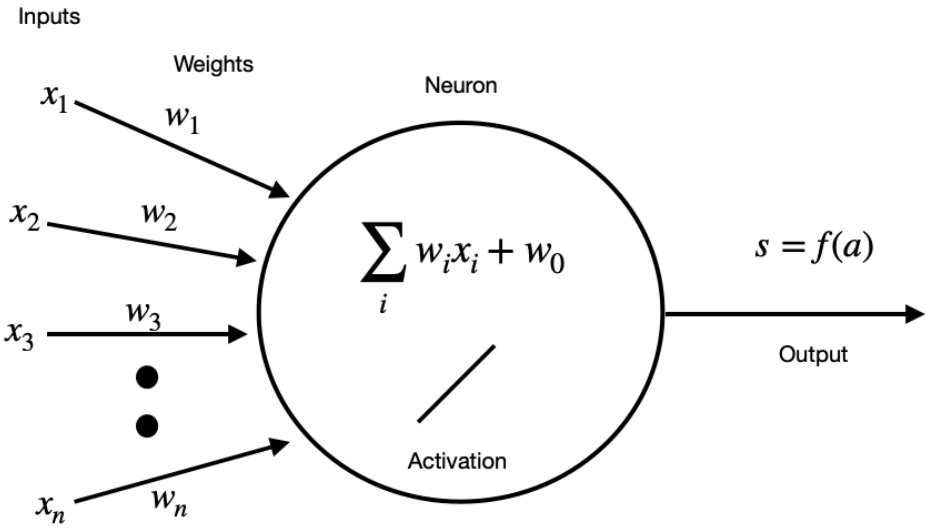


REGRESSION TASKS

Regression analysis is a method that helps us to analyze and understand the relationship between two or more variables of interest.
Task of predicting continuous (or ordered) values for given input. That is fitting a curve or fitting a surface.

Regression is a two step process:
Step1: Building a regression model by learning from data (Training phase)
Step2: Using regression model for prediction (Testing phase) and Predicting output variable

PERCEPTRON MODEL



The Regression task is very similar to the classification task except we use the linear activation function at the output instead of the logistic sigmoidal function.

ALGORITHM (TRAINING):

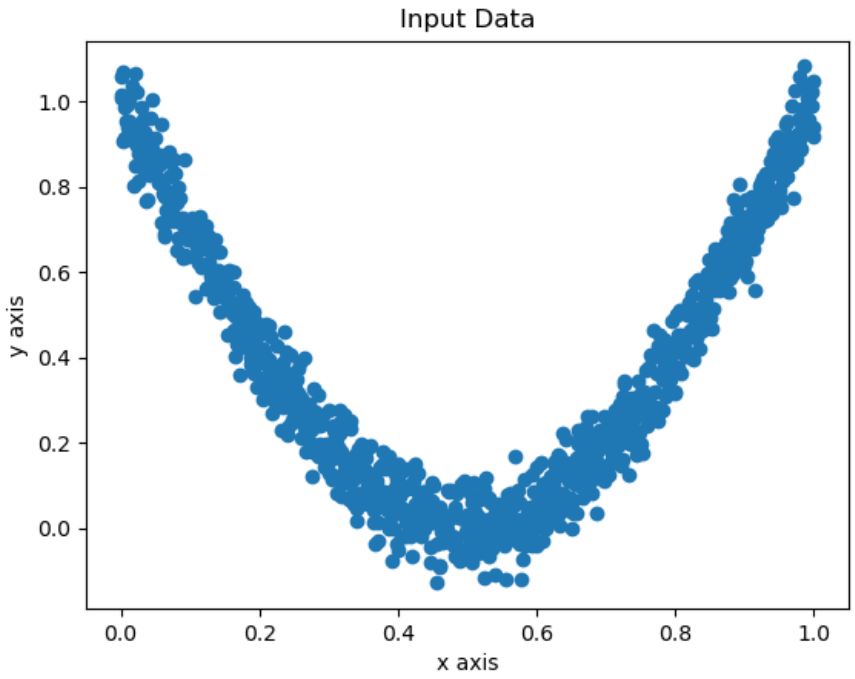
1. *Initialised the learning parameter w with random values*
2. *Chosen single value of x_n from the vector. Example for the univariate regression $\hat{x}_n = [1, x_n[0], x_n[1]]$ where $x_n \in \mathbb{R}^2$*
3. *Computation of the output of the neuron, $a_n = w^T \hat{x}_n$*
4. *Computation of instantaneous error, $E_n = \frac{1}{2}(y_n - s_n)^2$*
5. *Updation of weights w: $w = w + \eta \Delta w$, where Δw is the change in of the direction opposite to the gradient which is derived earlier. Thus $\Delta w = -(y_n - s_n)\frac{\partial f(a_n)}{\partial a_n} x_0$. The derivative of the linear function is 1. Finally the updated weight using linear activation function is given by: $\Delta w = \eta(y_n - s_n)x_0$ where $0 \leq \eta \leq 1$.*
6. *Repeating the steps 2 to 5 for all the training data set: defined by epoch.*
7. *Calculating the average error, $E_{avg} = \frac{1}{2N}\sum_{n=1}^{N} E_n$*
8. *Repeating steps 2 to 7 until the convergence criteria: 100 epoch or Average error falls below 10^{-3}*

ALGORITHM (TESTING):

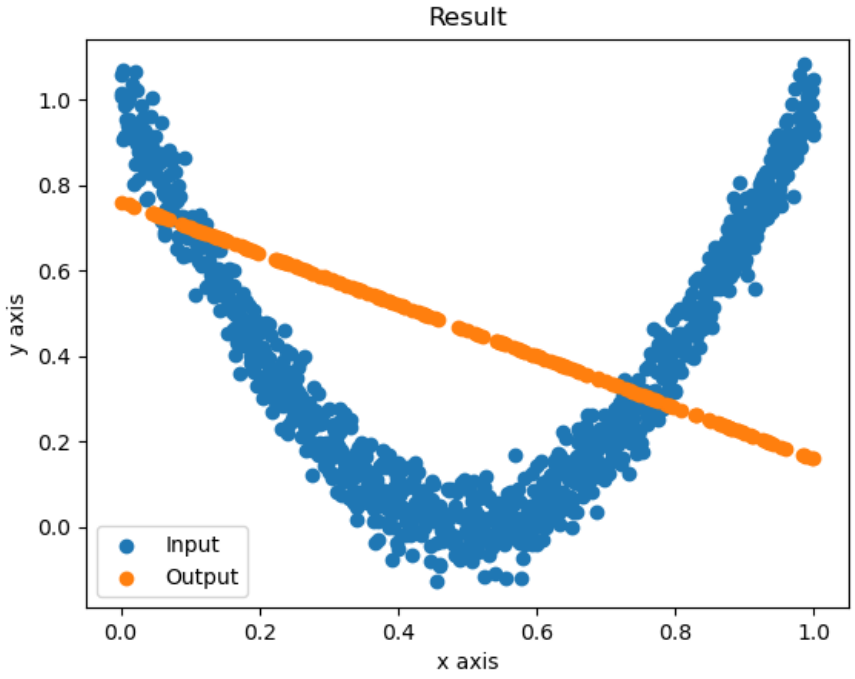
1. *With the computed w, find $a_n = w^T \hat{x}_n$*
2. *Compute the prediction accuracy measured in terms of squared error: $E = (\hat{y} - y)^2$*
3. *The prediction accuracy of regression model is measured in terms of root mean squared error: $E_{RMSE} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - y_n)^2}$*

UNIVARIATE DATA RESULT USING PERCEPTRON

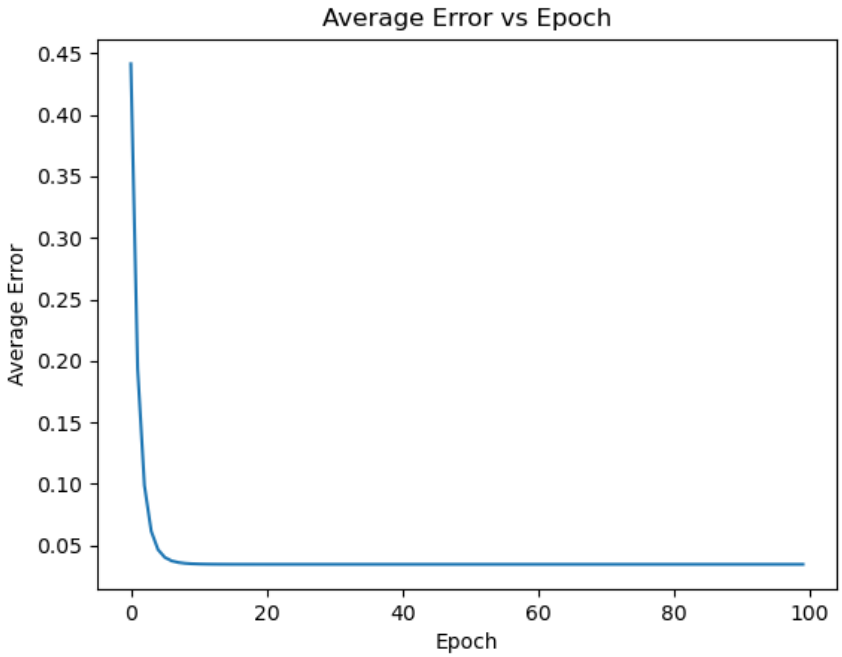
The perceptron model is tested using the univariate data shown below



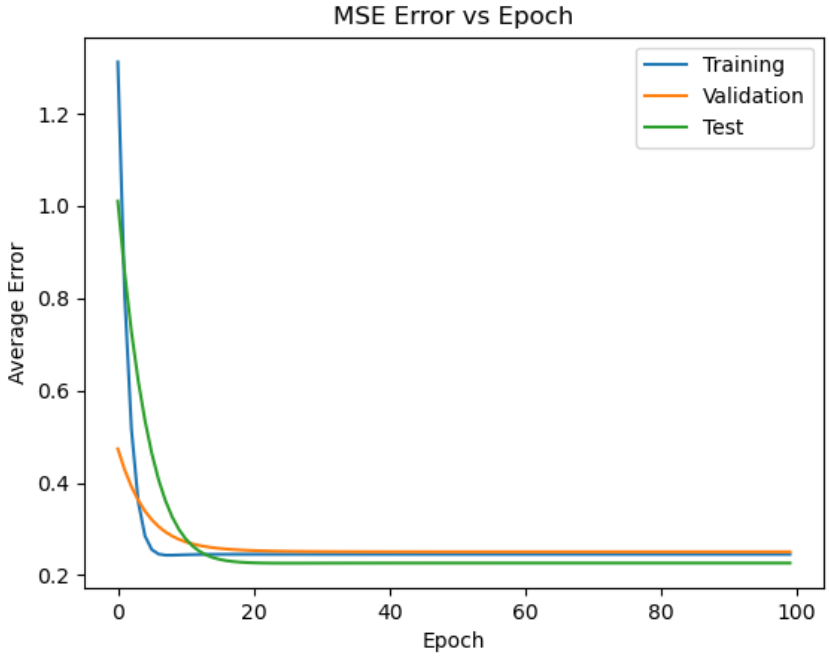
Using the above data we obtained the results shown in orange line. It tries to fit the curve using the linear line (perceptron method)



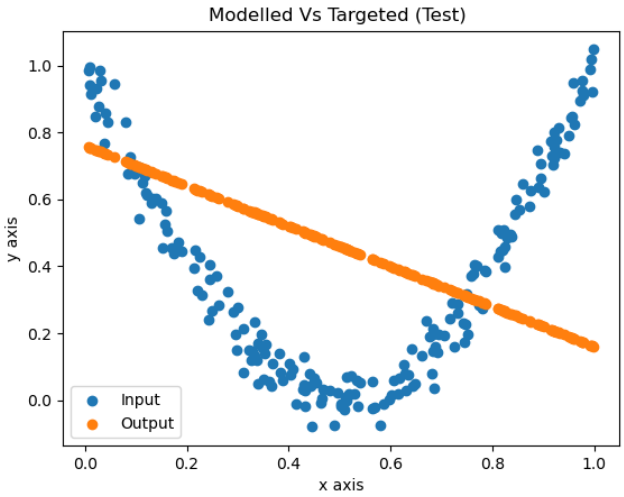
The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch

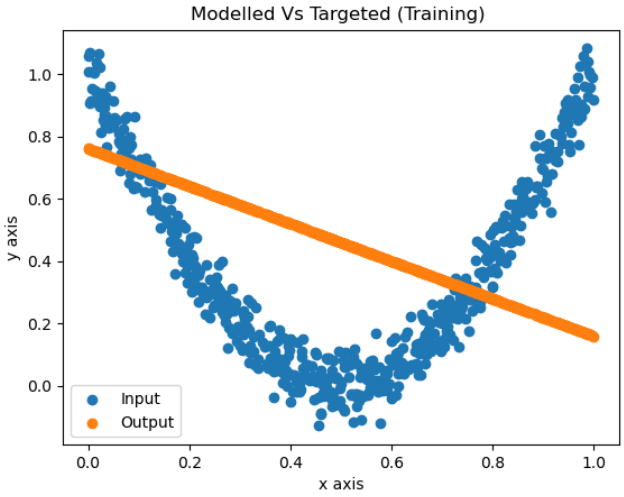


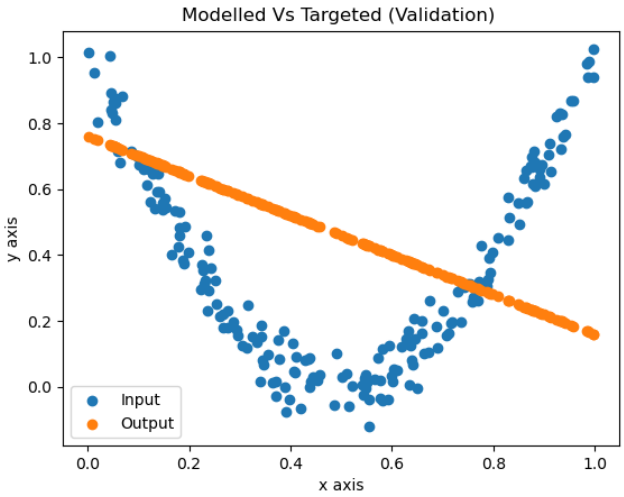
The RMSE error is plotted for each training, validation and test data. We observed that RMSE does not reduce after certain level due to non linearity



MODELLED OUTPUT Vs TARGET OUTPUT

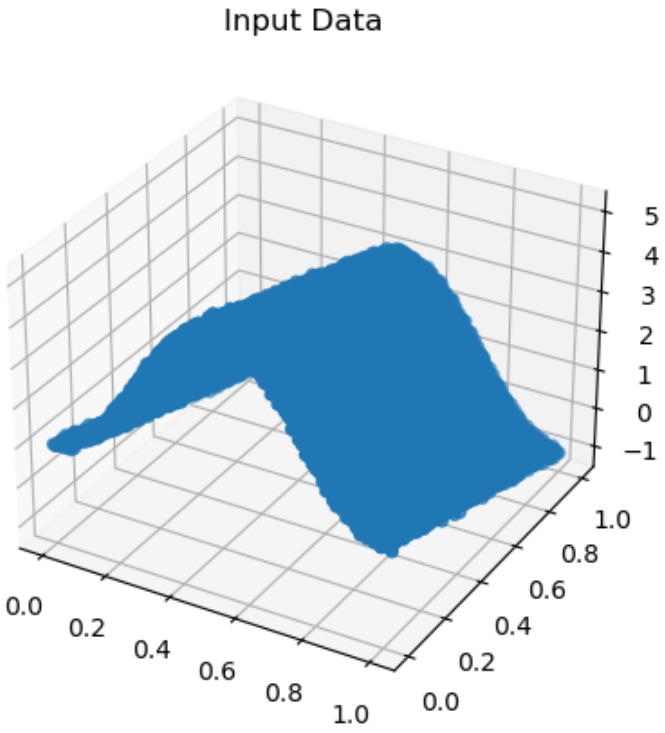




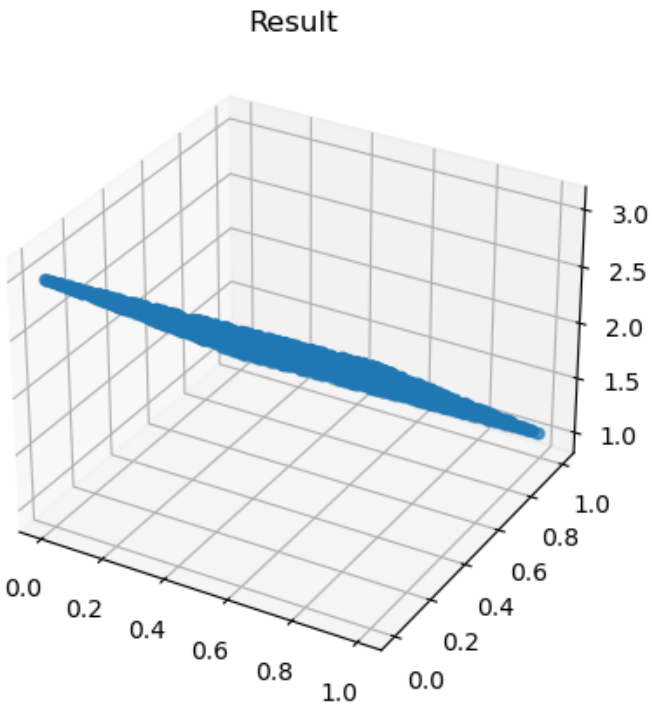


BIVARIATE DATA RESULT USING PERCEPTRON

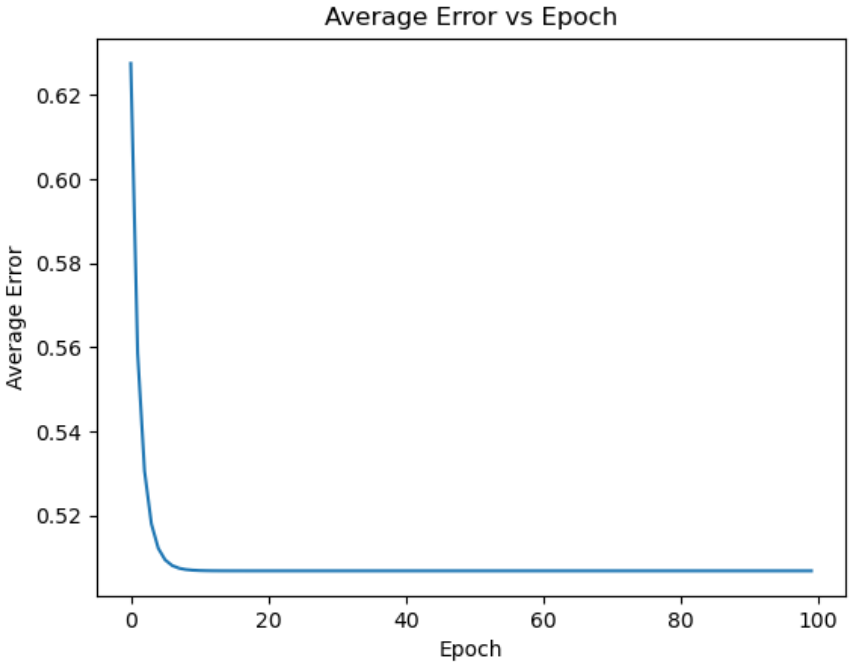
The perceptron model is tested using the bivariate data shown below



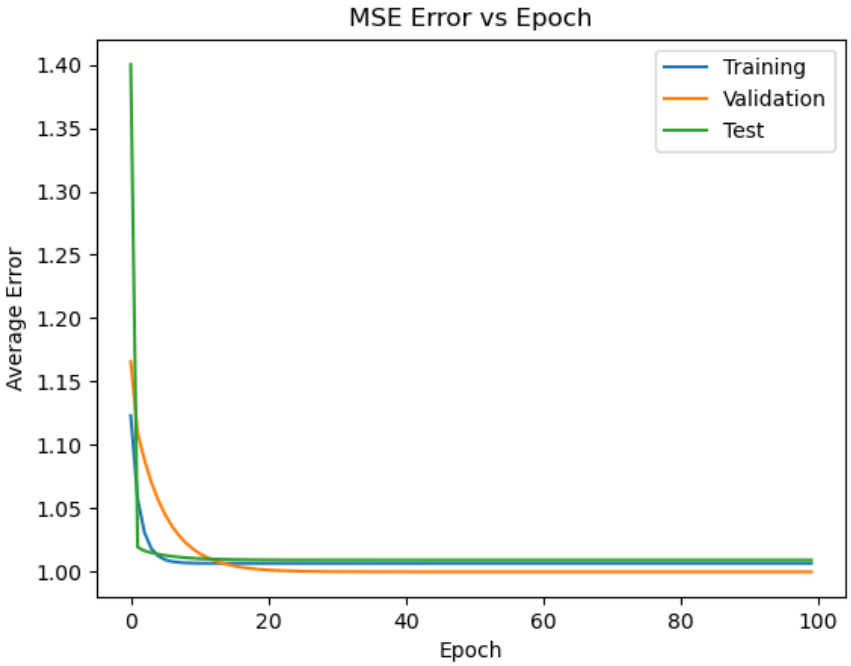
Using the above data we obtained the results shown. It tries to fit the surface using the linear surface (perceptron method)



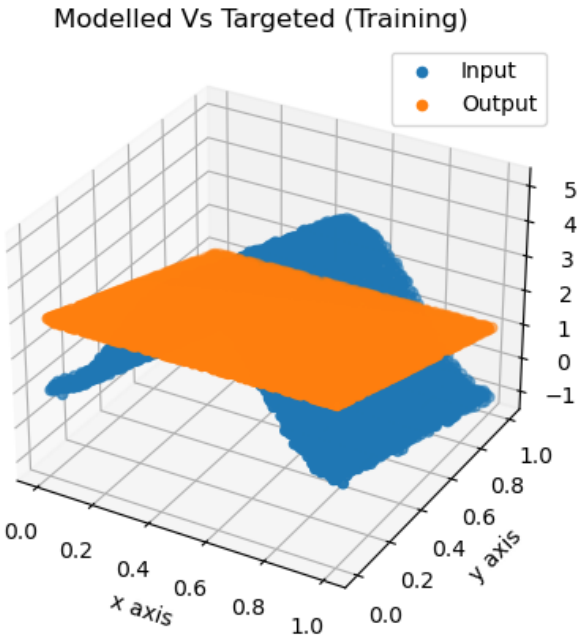
The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch but converges at very large value itself

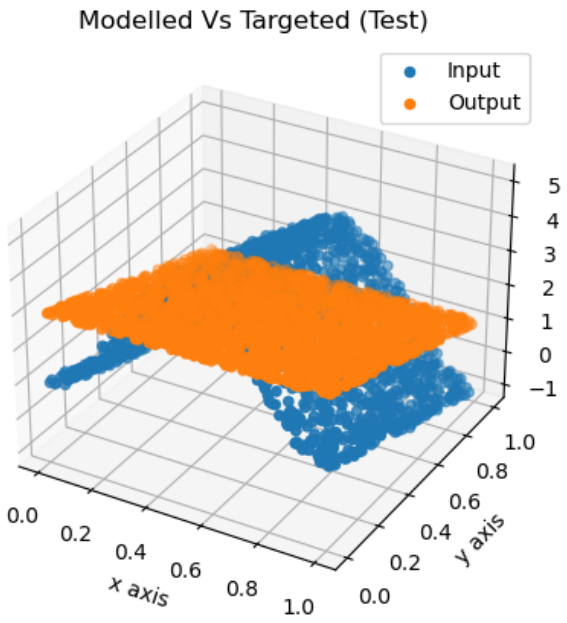


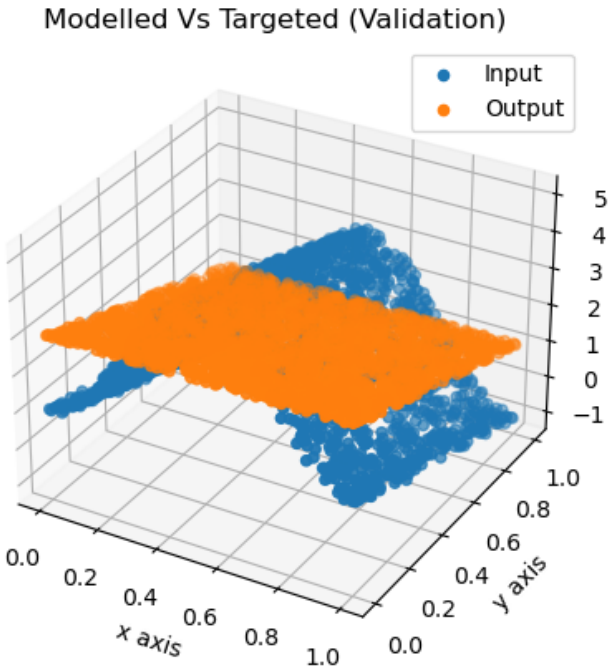
The RMSE error is plotted for each training, validation and test data. We observed that RMSE does not reduce after certain level due to non linearity



MODELLED OUTPUT Vs TARGET OUTPUT





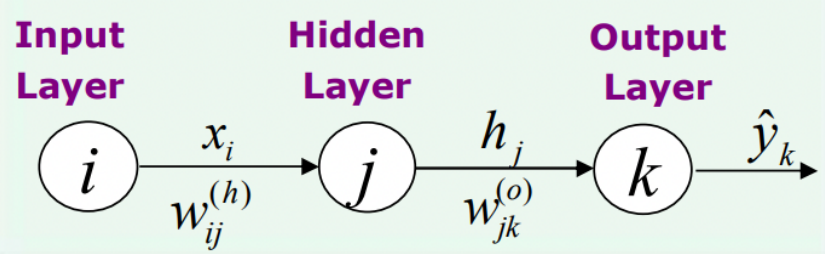


FCNN MODEL

ONE HIDDEN LAYER

STOCHASTIC GRADIENT DESCENT ALGORITHM (TRAINING):

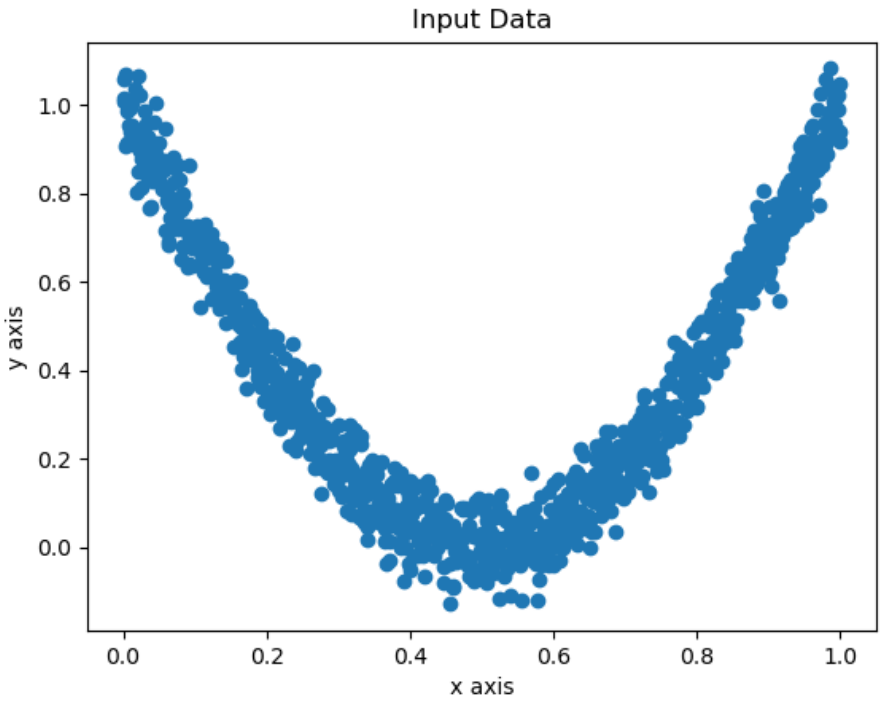
1. *Initialised the learning parameter* $w^{(h)}$ *and* $w^{(o)}$ *with random values*
2. *Chosen single value of* x_n *from the vector. For example, bivariate regression* $\hat{x}_n = [1, x_n[0], x_n[1]]$ *where* $x_n \in \mathbb{R}^2$
3. ***Forward Computation****: Output of all the neuron,* $a_n = w^T \hat{x}_n$



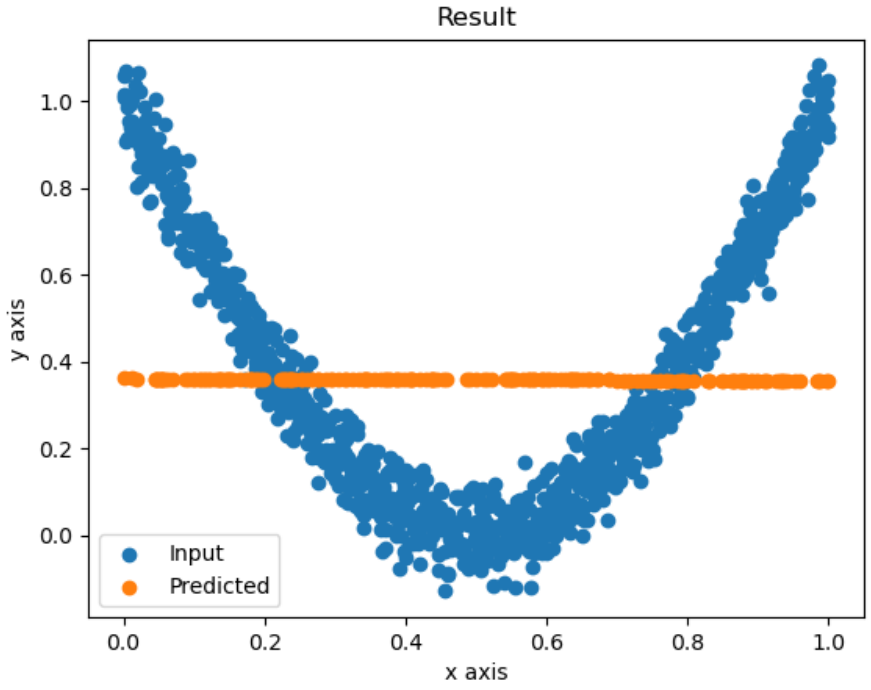
4. ***Backward Computation:*** *Computation of instantaneous error,* $E_n = \frac{1}{2}(y_n - s_n)^2$
5. *Updation of weights* $w^{(h)}$ *and* $w^{(o)}$*:* $w_{jk}^{(o)} = w_{jk}^{(o)} + \eta \Delta w_{jk}^{(o)}$ *and* $w_{ij}^{(h)} = w_{ij}^{(h)} + \eta \Delta w_{ij}^{(h)}$, *where* $\Delta w_{jk}^{(o)} = \eta(y_{nk} - \hat{y}_{nk}) h_{nj}$ *and* $\Delta w_{ij}^{(h)} = \eta \sum_{k=1}^{K} \left((y_{nk} - \hat{y}_{nk})\right) w_{jk}^{(o)} x_i$ *and* $0 \leq \eta \leq 1$.
6. *Repeating the steps 2 to 5 for all the training data set: defined by epoch.*
7. *Calculating the average error,* $E_{avg} = \frac{1}{2N} \sum_{n=1}^{N} E_n$
8. *Repeating steps 2 to 7 until the convergence criteria: 100 epoch or Average error falls below* 10^{-3}

UNIVARIATE DATA RESULT USING FCNN ONE HIDDEN LAYER

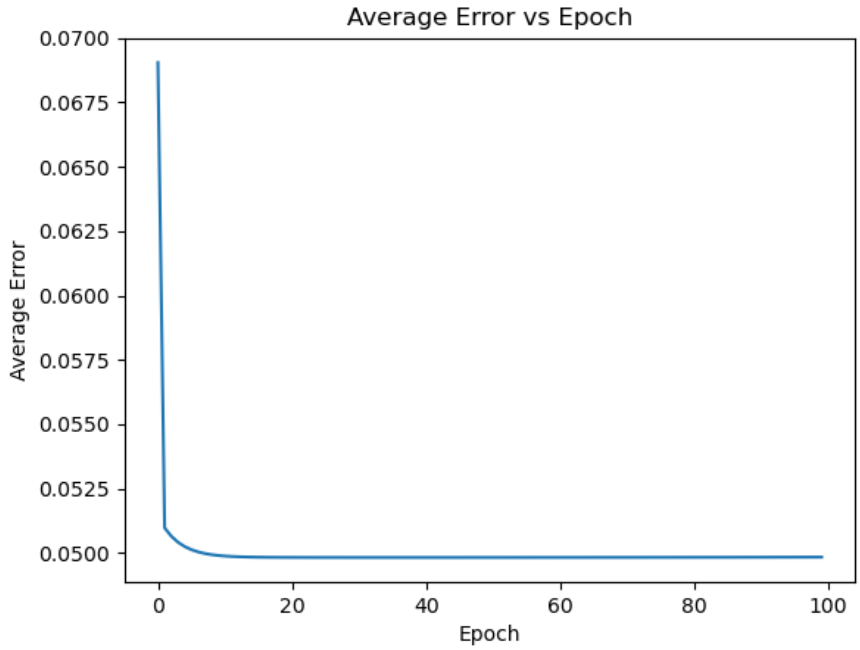
The FCNN model is tested using the univariate data shown below



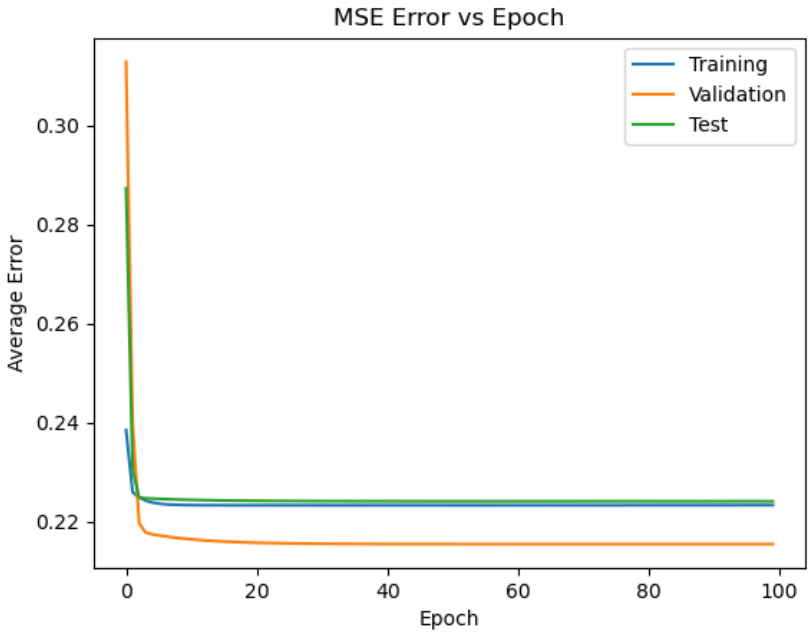
Using the above data we obtained the results shown in orange line. It tries to fit the curve using the one layer but fails.



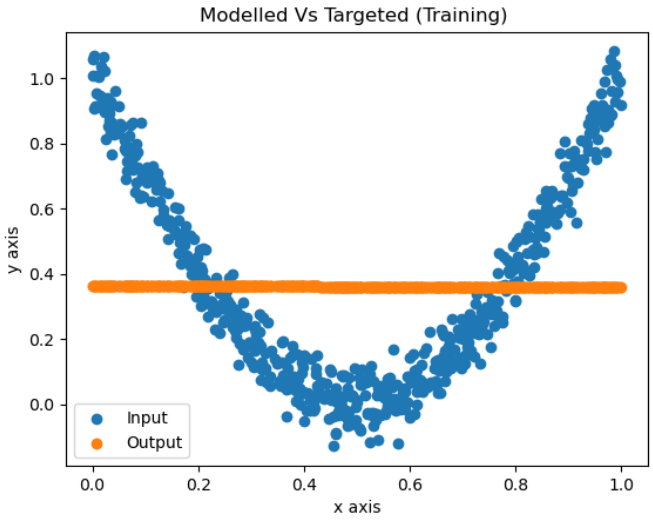
The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch

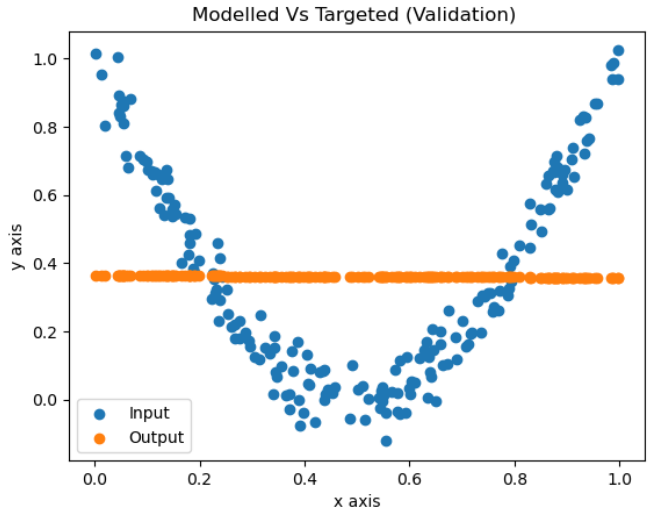


The RMSE error is plotted for each training, validation and test data. We observed that RMSE does not reduce after certain level.

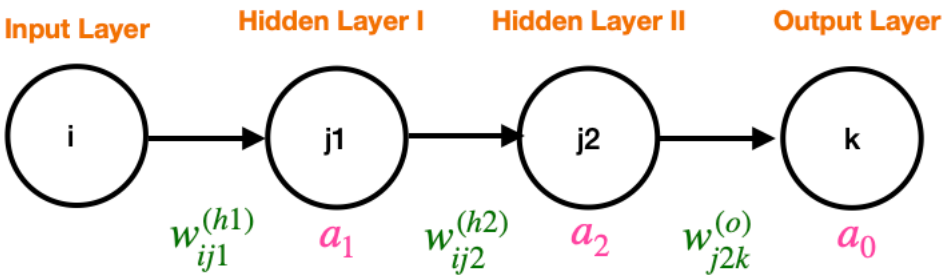


MODELLED OUTPUT Vs TARGET OUTPUT





TWO HIDDEN LAYER



STOCHASTIC GRADIENT DESCENT ALGORITHM (TRAINING):

1. *Initialised the learning parameter* $w^{(h1)}$, $w^{(h2)}$ *and* $w^{(o)}$ *with random values*

2. *Chosen single value of* x_n *from the vector. Here for the bivariate data* $\hat{x}_n = [1, x_n[0], x_n[1]]$ *where* $x_n \in \mathbb{R}^2$

3. ***Forward Computation****: Output of all the neuron,* $a_n = w^T \hat{x}_n$

4. ***Backward Computation:*** *Computation of instantaneous error,* $E_n = \frac{1}{2}(y_n - s_n)^2$

5. *Updation of weights* $w^{(h1)}$, $w^{(h2)}$ *and* $w^{(o)}$*:*

$w^{(o)} = w^{(o)} + \eta\delta^{(o)}a_2$ *where* $\delta^{(o)} = (y - a_0)$

$w^{(h2)} = w^{(h2)} + \eta\delta^{(h2)}a_1$ *where* $\delta^{(h2)} = \left(\sum w^{(o)}\delta^{(o)}\right)$

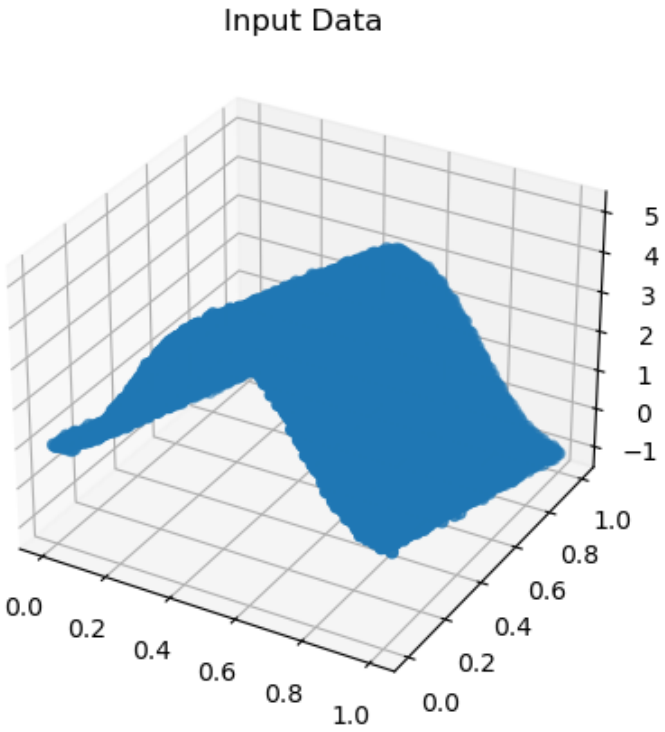
$w^{(h1)} = w^{(h1)} + \eta\delta^{(h1)}x_i$ *where* $\delta^{(h1)} = \left(\sum w^{(h2)}\delta^{(h2)}\right)$

6. *Repeating the steps 2 to 5 for all the training data set: defined by epoch.*

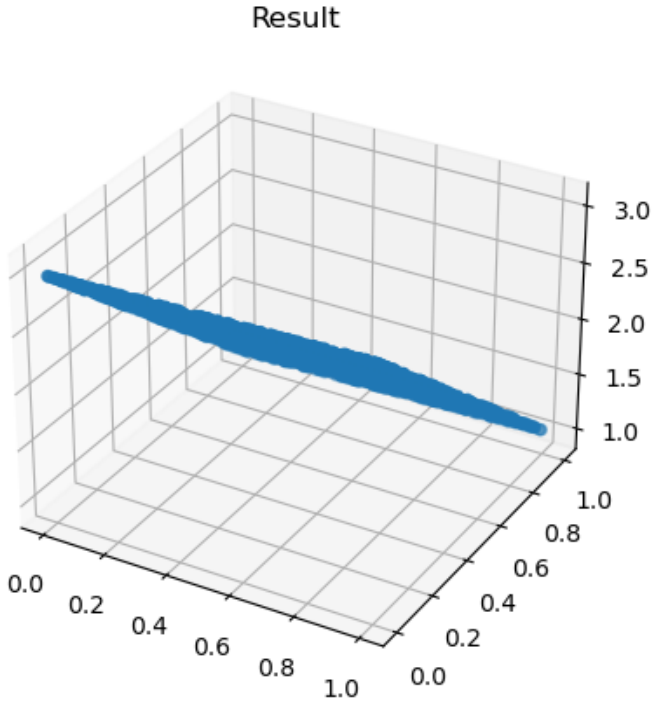
7. *Calculating the average error,* $E_{avg} = \frac{1}{N}\sum_{n=1}^{N} E_n$

8. *Repeating steps 2 to 7 until the convergence criteria: 100 epoch or Average error falls below* 10^{-3}

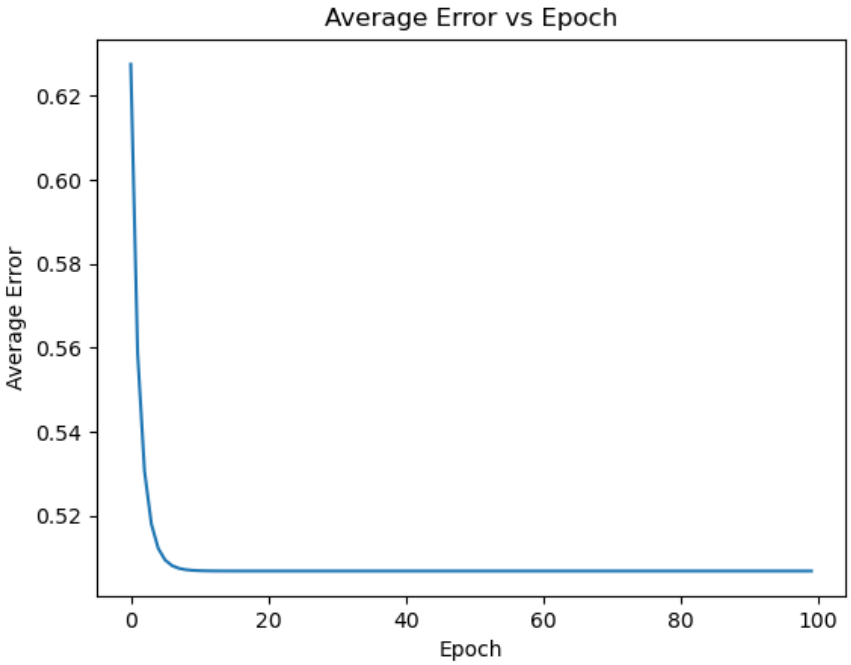
The perceptron model is tested using the bivariate data shown below



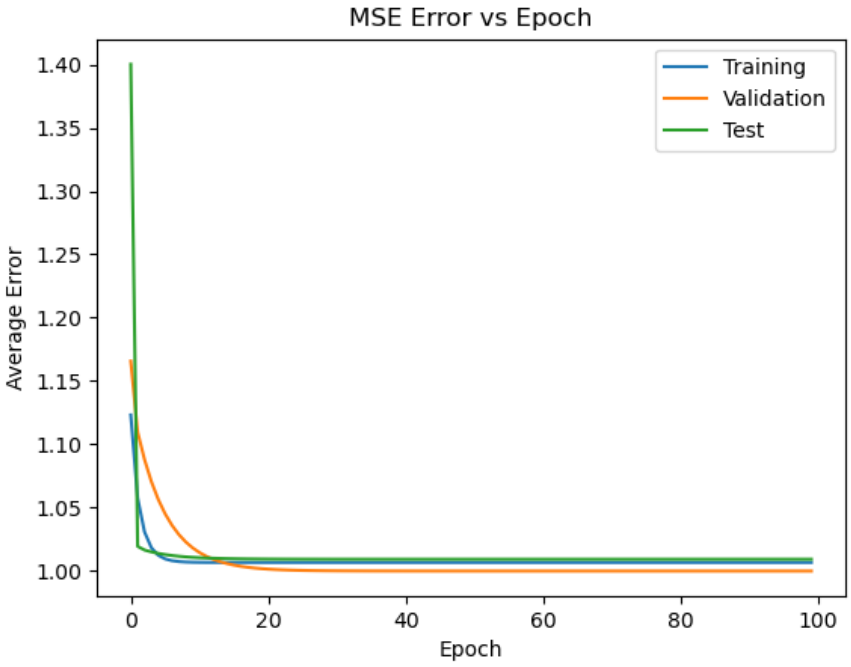
Using the above data we obtained the results shown. It tries to fit the surface but fails.



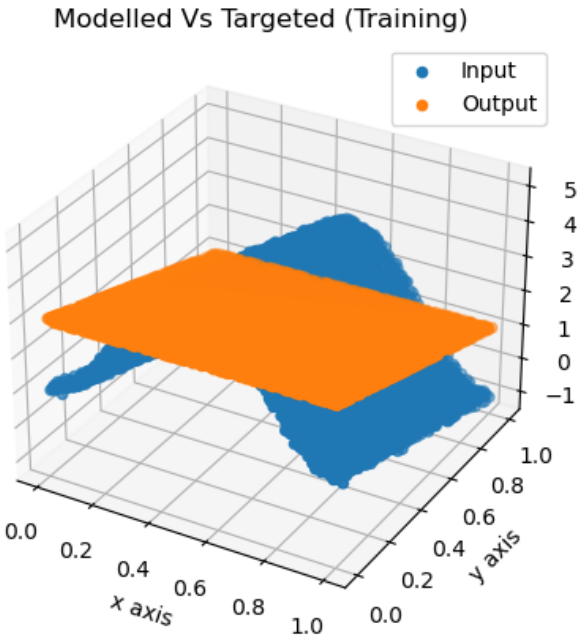
The average error and the epoch is plotted and observed that average error decreases exponentially with the number of epoch but converges at very large value itself

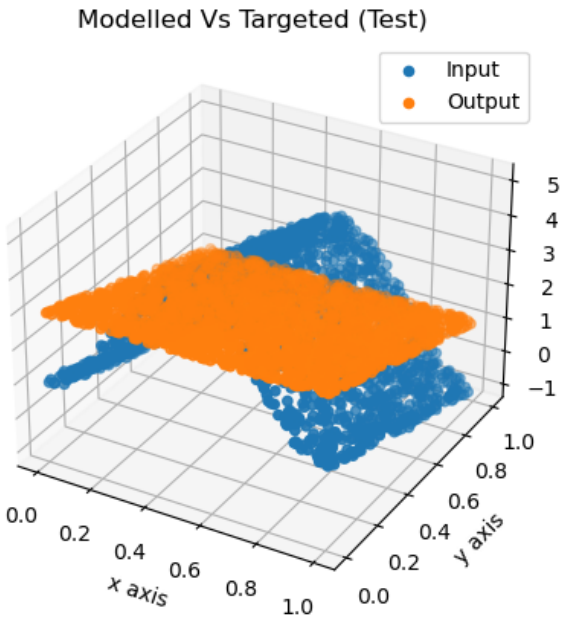


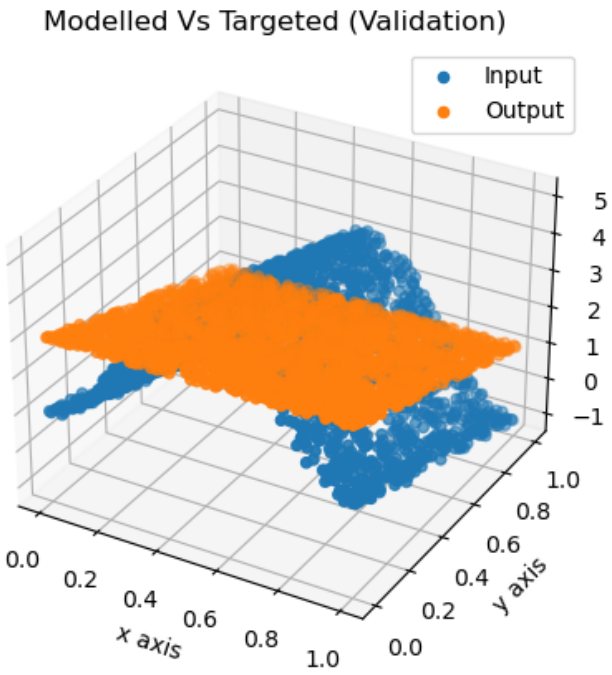
The RMSE error is plotted for each training, validation and test data. We observed that RMSE does not reduce after certain level.



MODELLED OUTPUT Vs TARGET OUTPUT







Appendix

1) Activation Function

The sigmoidal activation function is used for the tasks. There are family of sigmoidal functions such as logistic, tan hyperbolic etc.

We used logistic activation function which is defined by,

$$f(x) = \frac{1}{1 + e^{-\beta x}}$$

The logistic function for different β is plotted

Derivative:

$$\frac{d}{dx} f(x) = \frac{d}{dx}\left(\frac{1}{1 + e^{-\beta x}}\right) = \frac{(1 + e^{-\beta x}) \cdot \frac{d}{dx}(1) - (1)\frac{d}{dx}(1 + e^{-\beta x})}{(1 + e^{-\beta x})^2}$$

$$= \frac{\beta e^{-\beta x}}{(1 + e^{-\beta x})^2} = \beta\left(\frac{1}{1 + e^{-\beta x}}\right)\left(\frac{e^{-\beta x}}{1 + e^{-\beta x}}\right)$$

$$= \beta\left(\frac{1}{1 + e^{-\beta x}}\right)\left(+1 - 1 + \frac{e^{-\beta x}}{1 + e^{-\beta x}}\right)$$

$$= \beta\left(\frac{1}{1 + e^{-\beta x}}\right)\left(1 - \frac{1}{1 + e^{-\beta x}}\right)$$

$$\frac{d}{dx} f(x) = \beta f(x)(1 - f(x)) \qquad \text{.....(I)}$$

2) COST FUNCTION

The Instantaneous cost function is used as the error function. It is defined by,

$$Error = \frac{1}{2}(y - \hat{y})^2$$

y is the actual output and $\hat{y}$ is the predicted output.

GITHUB LINK

https://github.com/Rajesh-Smartino/Deep-Learning

TEAM MEMBERS

GROUP 22

NAME	ROLLNO	MAIL ID
Rajesh R	S21005	s21005@students.iitmandi.ac.in
Naisarg Pandya	S21012	s21012@students.iitmandi.ac.in
Ashok Kumar	D19042	d19042@students.iitmandi.ac.in