

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from this bar, containing the date.

7/3/2023

Deep Learning Assignment-2

FCNN

Presented by:

1. R. Ramakrishna Kashyap – T22101
2. Anuj Kumar Shukla – T22103
3. Nishant Gupta- T22221

Index

1. Introduction.....	2
2. Stochastic Gradient Descent	4
3. Classification	6
a) Linearly Separable Classes	7.
b) Non Linearly Separable Classes.....	10
4. Regression	13
a) Univariate	14
b) Bivariate.....	17

Perceptron

A perceptron is a type of neural network model used for binary classification tasks. It consists of a single layer of artificial neurons that receive input signals, weight them, and combine them to produce a single output. The output is then passed through an activation function, which maps the output to a binary value, usually 0 or 1.

The perceptron learning algorithm, also known as the delta rule, is used to train the perceptron by adjusting the weights assigned to each input signal in response to errors made during classification. The algorithm continues to adjust the weights until a stopping criterion is met or until the weights converge to a stable set of values.

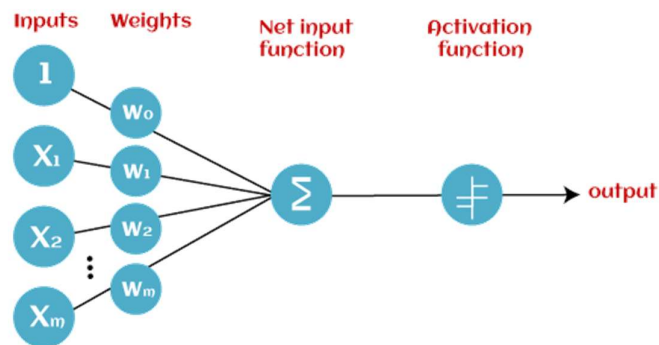


Figure 1: Perceptron Model

Input Nodes or Input Layer:

This is the primary component of Perceptron which accepts the initial data into the system for further processing. Each input node contains a real numerical value.

Wight and Bias:

Weight parameter represents the strength of the connection between units. This is another most important parameter of Perceptron components. Weight is directly proportional to the strength of the associated input neuron in deciding the output. Further, Bias can be considered as the line of intercept in a linear equation.

Activation Function:

These are the final and important components that help to determine whether the neuron will fire or not. Activation Function can be considered primarily as a step function.

Algorithm

The Perceptron algorithm is an iterative procedure for training a binary linear classifier. Given a set of input features and corresponding binary labels, the algorithm learns a set of weights for each feature and a bias term such that the linear combination of inputs and weights plus the bias is a good predictor of the binary labels.

- 1) Initialize the weights and bias term to zero or small random values.
- 2) For each training example in the dataset, do the following:
 - a) Compute the dot product of the input features and the current weights and add the bias term.
 - b) Apply the activation function (usually a step function) to the result of the dot product to obtain a predicted binary label.
 - c) Update the weights and bias term according to the error in the prediction, multiplied by a learning rate and the input features.
- 3) Repeat steps 2 until the predicted labels for all training examples are correct or a maximum number of iterations is reached.

During training, the weights and bias term are updated based on the errors made in the predictions, which drive the algorithm to find the best set of weights that can accurately classify the training examples. Once trained, the weights and bias can be used to predict the binary label of new examples by computing the dot product of the input features and the weights, adding the bias term, and applying the activation function.

Stochastic Gradient Descent

Stochastic Gradient Descent is an optimization algorithm that can be used to train neural network models. The Stochastic Gradient Descent algorithm requires gradients to be calculated for each variable in the model so that new values for the variables can be calculated.

Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

Feed Forward Neural Network

A Feed Forward Neural Network is an artificial neural network in which the connections between nodes does not form a cycle. The opposite of a feed forward neural network is a recurrent neural network, in which certain pathways are cycled. The feed forward model is the simplest form of neural network as information is only processed in one direction. While the data may pass through multiple hidden nodes, it always moves in one direction and never backwards.

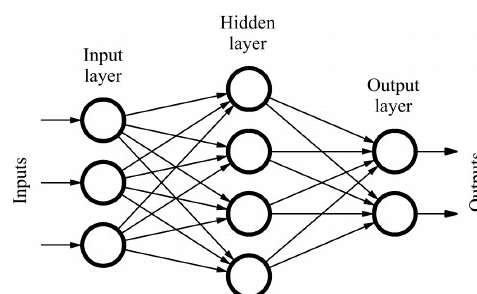


Figure 3: FCNN Model

Input layer: This layer consists of the neurons that receive inputs and pass them on to the other layers. The number of neurons in the input layer should be equal to the attributes or features in the dataset.

Image source: https://www.researchgate.net/figure/Sample-of-a-feed-forward-neural-network_fig1_234055177

Output layer: The output layer is the predicted feature and depends on the type of model you're building.

Hidden layer: In between the input and output layer, there are hidden layers based on the type of model. Hidden layers contain a vast number of neurons which apply transformations to the inputs before passing them. As the network is trained, the weights are updated to be more predictive.

Neuron weights: Weights refer to the strength or amplitude of a connection between two neurons. If you are familiar with linear regression, you can compare weights on inputs like coefficients. Weights are often initialized to small random values, such as values in the range 0 to 1.

Classification

Classification is a supervised machine learning method where the model tries to predict the correct label of a given input data. In classification, the model is fully trained using the training data, and then it is evaluated on test data before being used to perform prediction on new unseen data.

In classification, the input data is classified into different categories or classes, based on a set of predetermined criteria. The categories or classes can be binary, meaning there are only two possible outcomes, or multi-class, meaning there are more than two possible outcomes.

The process of classification involves representing input data as a feature vector, which comprises numerical or categorical values that describe the relevant characteristics of the data. The primary goal of a classification algorithm is to learn a function that can map these feature vectors to the target variable accurately. This learned function is then used to classify new instances of the data. The ability of the algorithm to accurately classify new instances depends on the quality of the mapping function, which should be generalizable and capable of making accurate predictions on previously unseen data.

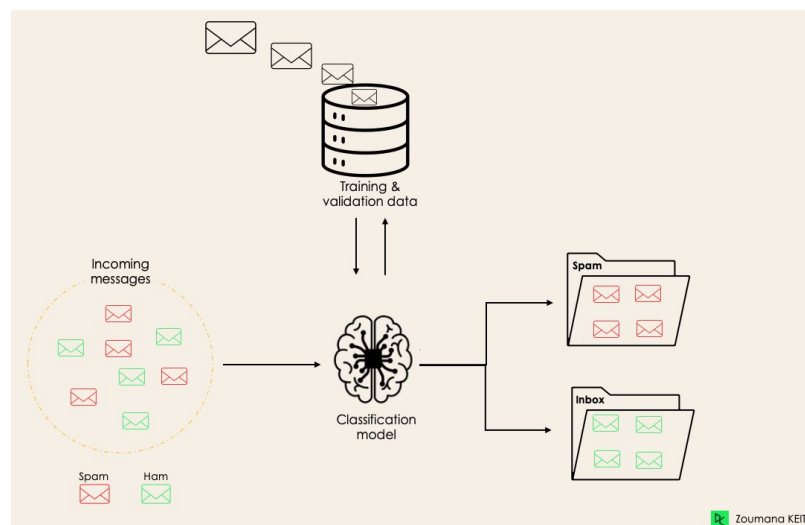


Figure 2: Classification Model

Image source: <https://www.datacamp.com/blog/classification-machine-learning>

Results:

Linearly Separable Classes

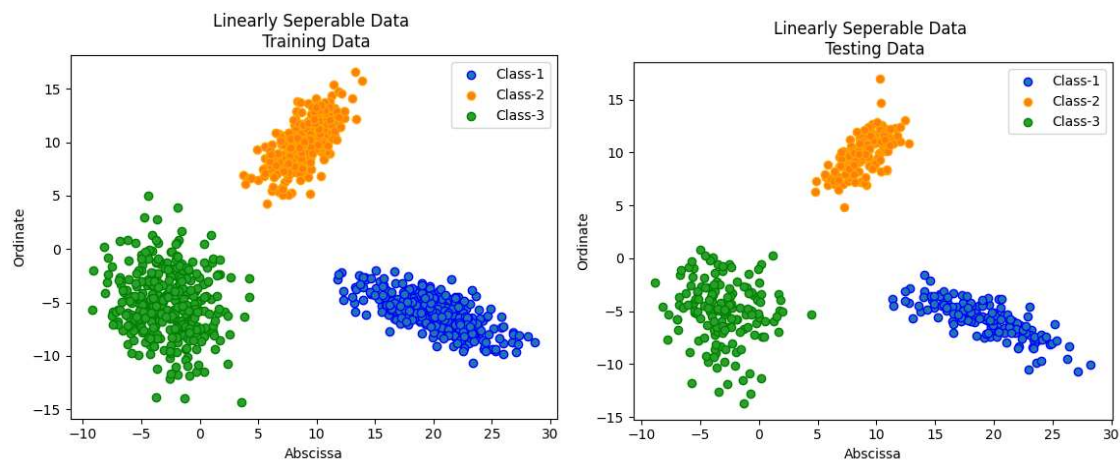
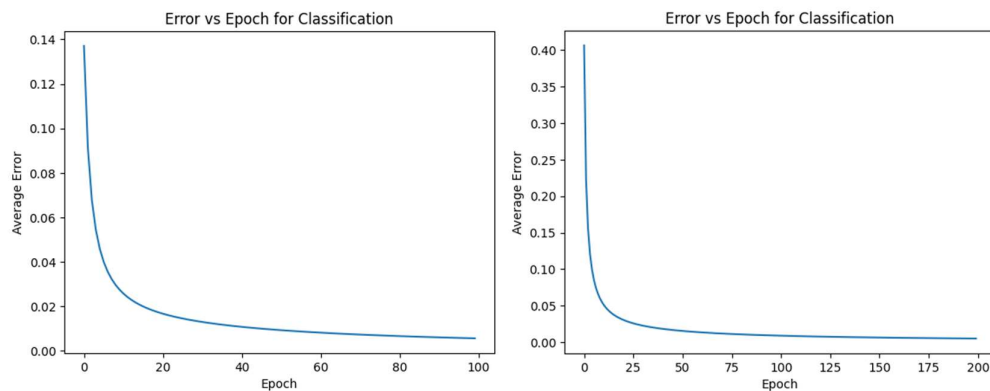


Figure 4: Training and Testing data



(a) 100 Epochs

(b) 200 Epochs

Figure 5: Average Error vs Epochs

Error after each epoch is decreasing and converging to zero as epochs are increased.

Decision Regions

Parameter's considered:

Learning Rate = 0.03

No of Neurons in hidden layer = 5

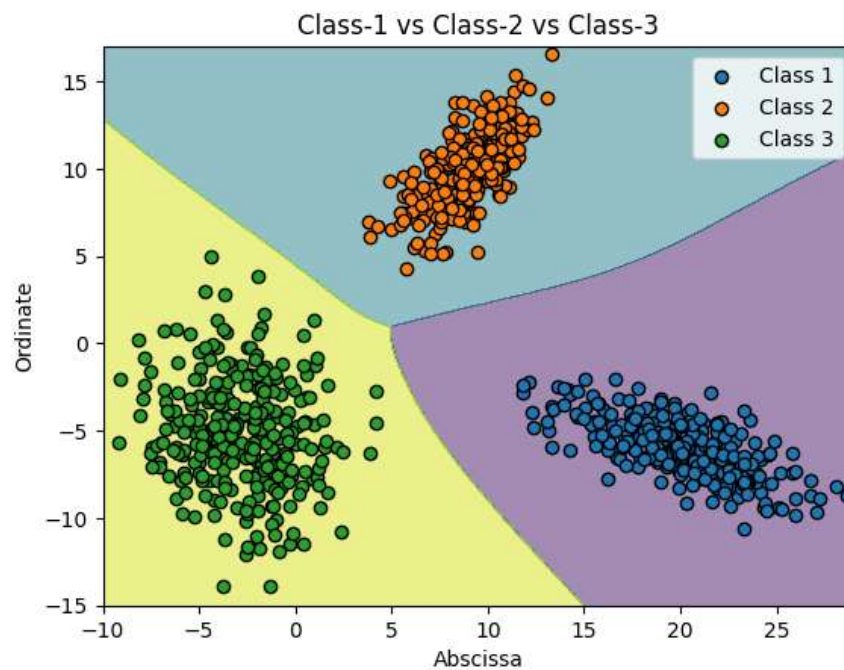


Figure 6: Decision boundaries for 3 classes

Test samples were plotted on the decision region (Region we obtained by training the Perceptron model with train samples) and were lying accurately in their respective regions.

Confusion Matrix:

		Actual Class		
Predicted Class		Class 1	Class 2	Class 3
	Class 1	100	0	0
	Class 2	0	100	0
	Class 3	0	0	100

Accuracy: $\frac{\text{Number of samples correctly classified}}{\text{Total number of samples used for testing}} * 100$

Accuracy = 100%

Accuracy of the system is 100% as the data is linearly separable i.e. samples from different classes can be distinguished completely by a straight line. Hence all the test samples are lying strictly in their respective regions.

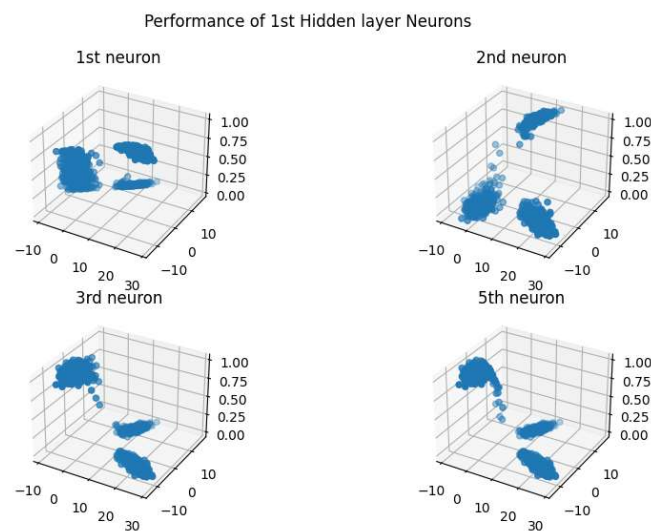


Figure 9: Performance of Neurons in Hidden Layers

Non-Linearly Separable Classes

Sample Data

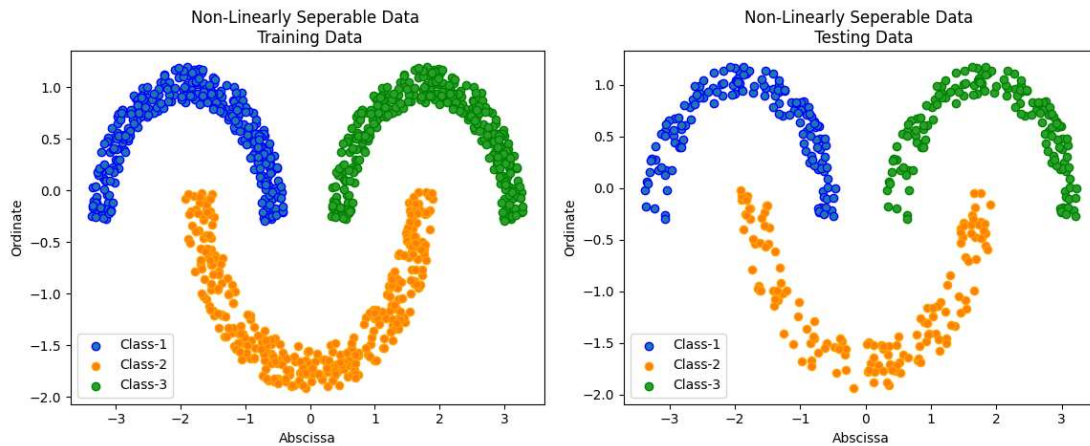
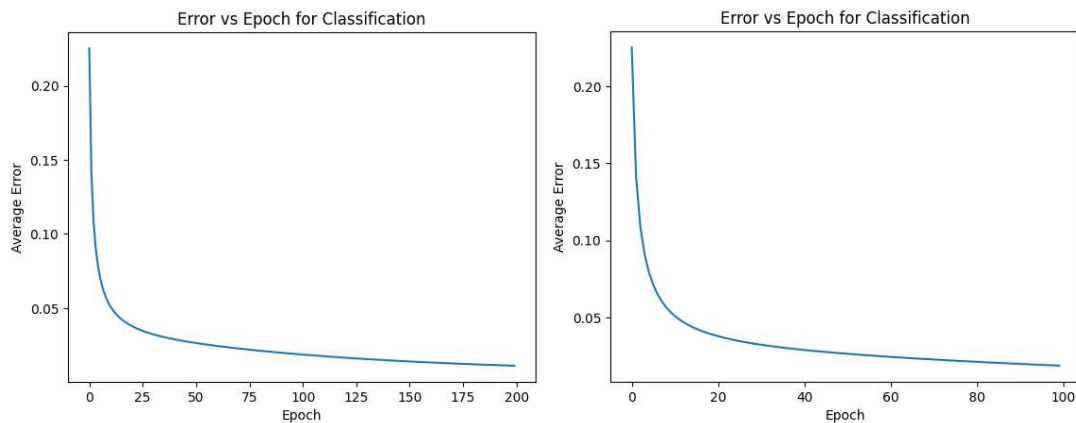


Figure 7: Training and Testing data

Test samples were plotted on the decision region (Region we obtained by training the Perceptron model with train samples) and were lying almost accurately in their respective regions.



(a) 200 Epochs

(b) 100 Epochs

Fig 8: Average Error vs Epochs

Error after each epoch is decreasing and converging to zero as epochs are increased.

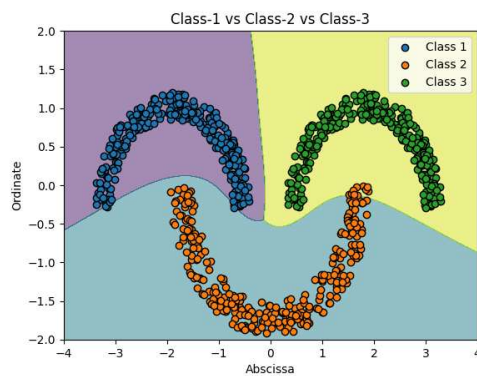
Decision Regions

Parameter's considered:

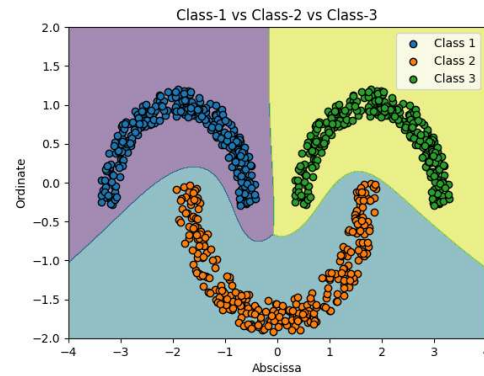
No of hidden Layers = 2

Learning Rate = 0.03

No of neurons in each hidden layer = 50



(a) 100 epoch



(b) 200 epoch

Figure 9: Decision Regions

Confusion Matrix:

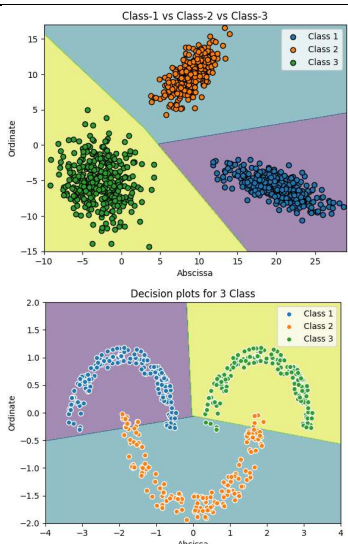
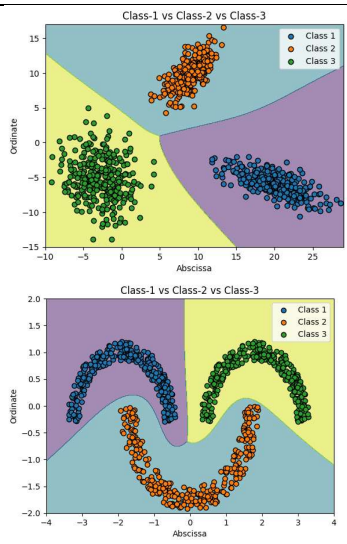
	Actual Class			
		Class 1	Class 2	Class 3
	Class 1	100	0	0
	Class 2	0	100	0
	Class 3	0	0	100

Accuracy: $\frac{\text{Number of samples correctly classified}}{\text{Total number of samples used for testing}} * 100$

Accuracy = 100%

Since we are using multiple hidden layers and neurons the decision boundary has been bent in order to accommodate all the points from a particular class into the same boundary. Hence we are getting 100% accuracy indicating all points are correctly classified.

Comparison between Single neuron perceptron & FCNN

Parameter	Single Neuron	FCNN
Decision Boundary		
Accuracy	Linearly Separable = 100% Non-Linearly Separable=94%	Linearly Separable = 100% Non-Linearly Separable=100%

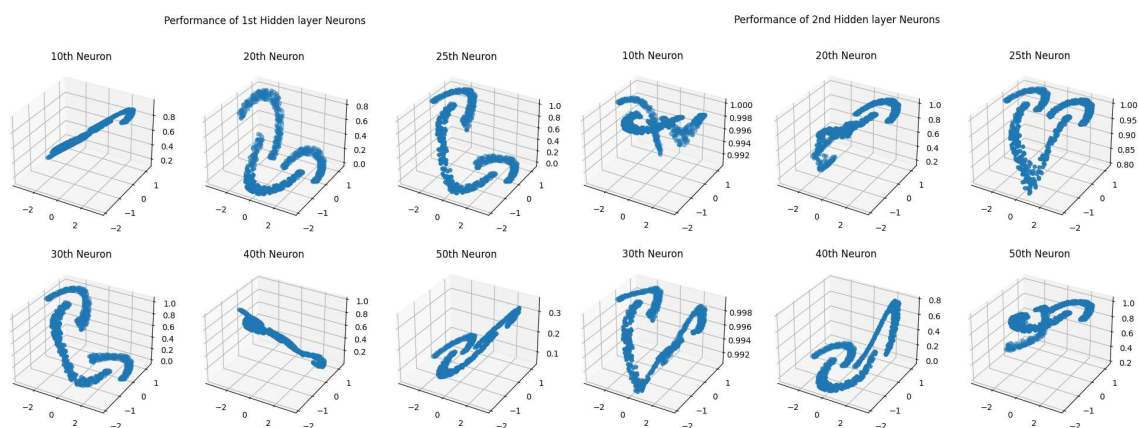


Figure 9: Performance of Neurons in Hidden Layers

Regression:

In deep learning, a regression task involves predicting a continuous output value based on a set of input features. The goal is to learn a function that maps the input features to the target output variable.

Deep learning models can be used for regression tasks by constructing a neural network architecture that is designed to learn a complex mapping between the input features and the output variable. The neural network can be trained on a dataset of labelled examples, where each example includes the input features and the corresponding output value.

During training, the neural network is adjusted to minimize the difference between its predicted output and the true output value for each example in the dataset. This is typically achieved by minimizing a loss function, such as mean squared error, which measures the difference between the predicted output and the true output.

Once the model is trained, it can be used to make predictions on new, unseen data by feeding the input features through the neural network and obtaining the corresponding output value.

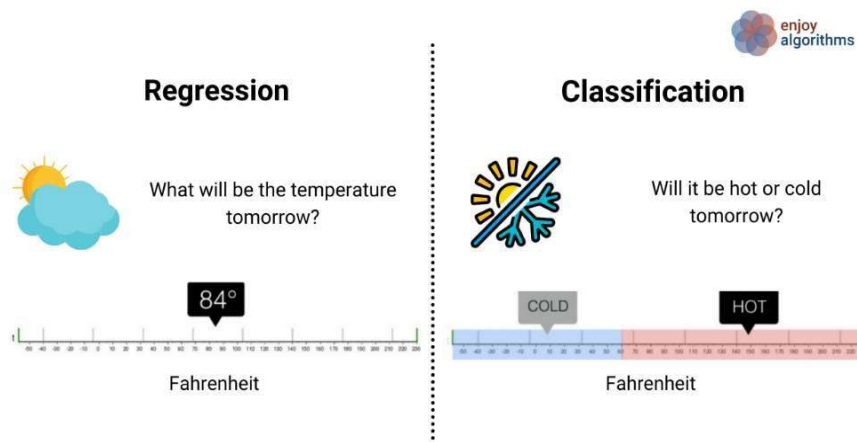


Figure 10: Regression vs Classification

Univariate Data

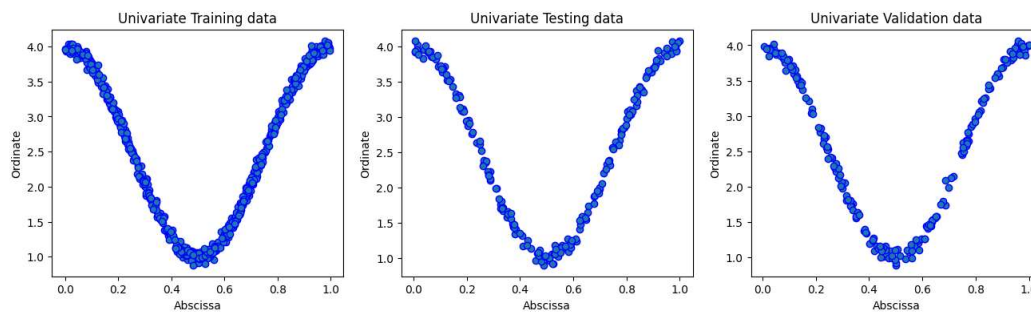


Figure 11: Training, Testing & Validation Data

Results:

No. of epochs= 1000

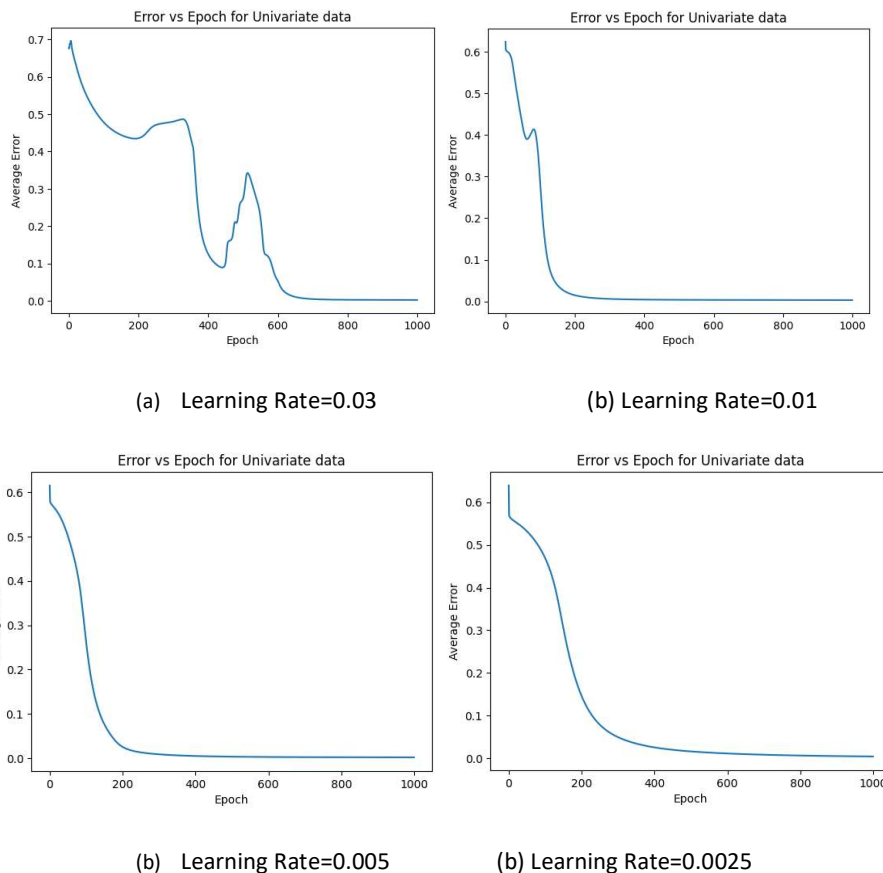


Figure 12: Error vs Epochs

When the learning rate was 0.03 we encountered a local minima at 420 epoch sue to which the average error increased up to a certain epoch and gradually the error decreased to zero. As the learning rate was decreased the error curve was becoming much smoother indicating a better approximation of the sample.

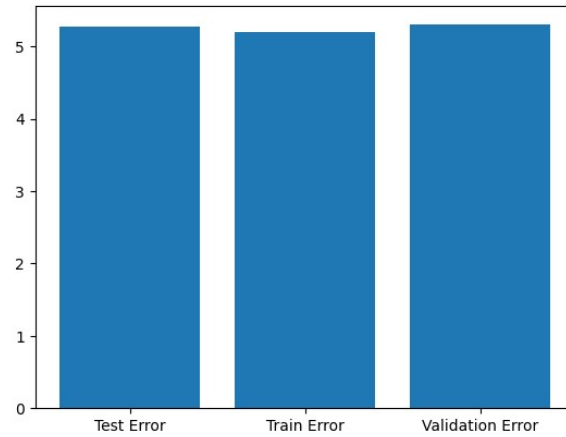


Figure 13: Mean Square Error in each data sets

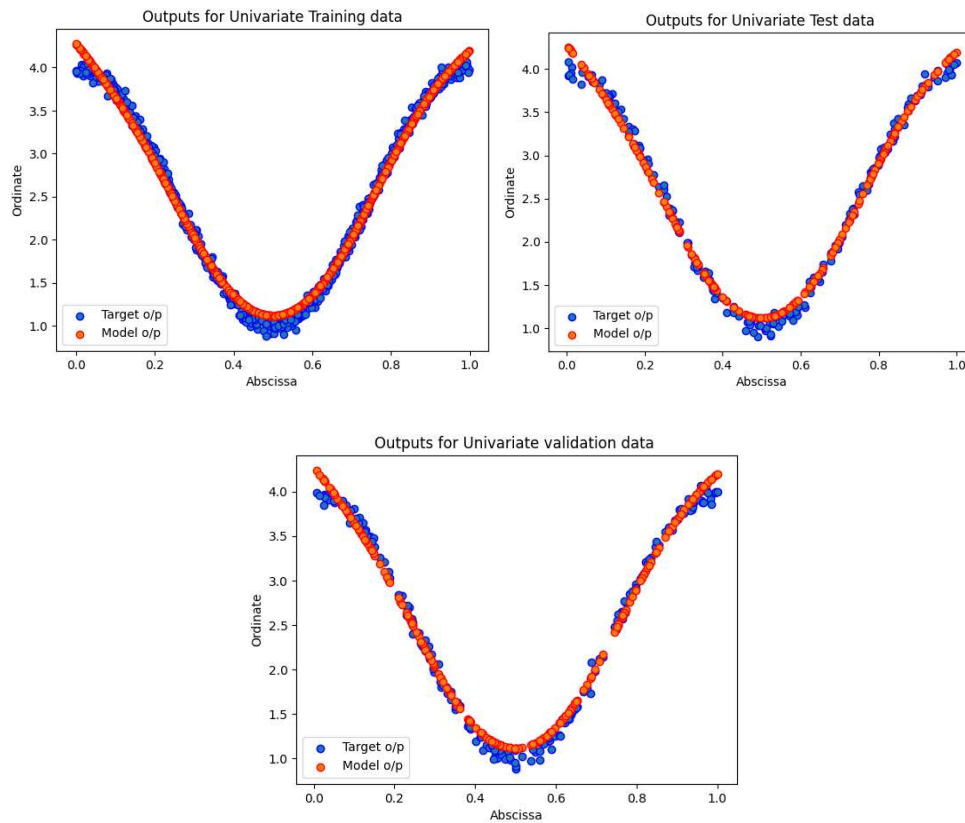


Figure 14: Model output superimposed on Target output

Model output is clearly following the target output. Some points at the ends were not as close as other points because of small learning rate. Increasing the learning rate was leading to increase in mean square error.

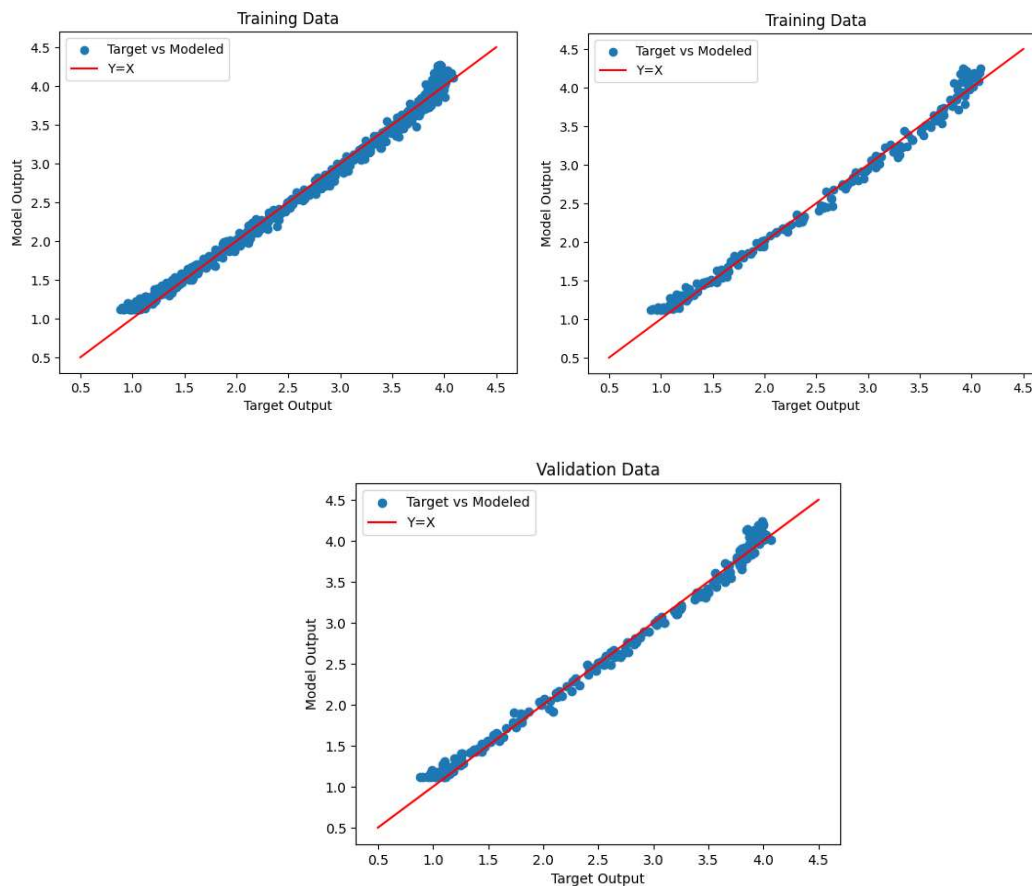


Figure 15: Target output and Model output Scatter plots

Fig.15 shows that the test sample fed into the model was clearly approximated onto a linear function.

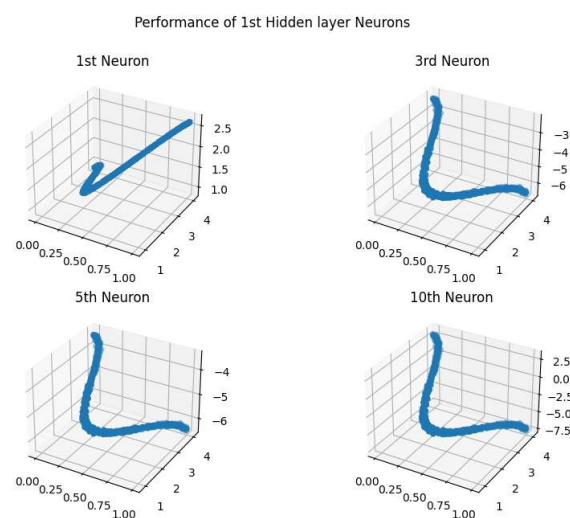


Figure 16: Performance of neurons in Hidden Layers

Bivariate Data

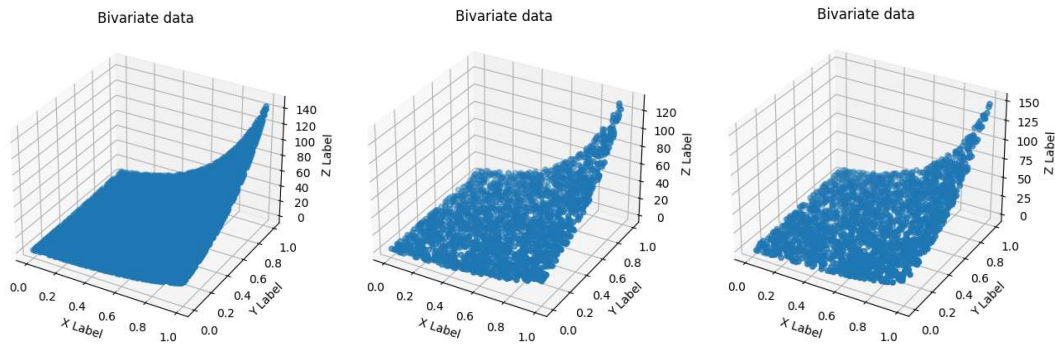
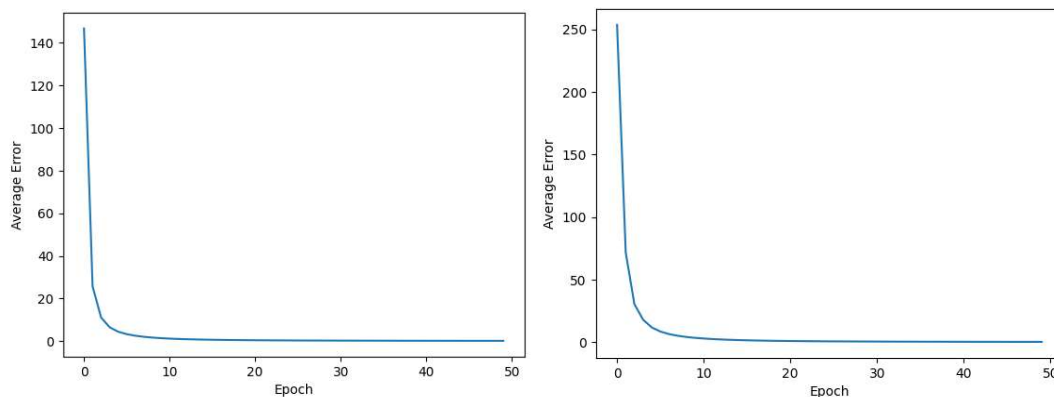


Figure 16: Training, Testing & Validation Data



(a) Learning Rate=0.001

(b) Learning Rate = 0.0005

Figure 17: Error vs Epochs

We can see that the error is getting reduced with increasing number of epochs. As the learning rate was decreased the error curve was becoming much smoother indicating a better approximation of the sample.



Figure 18: Mean Square Error for each datasets

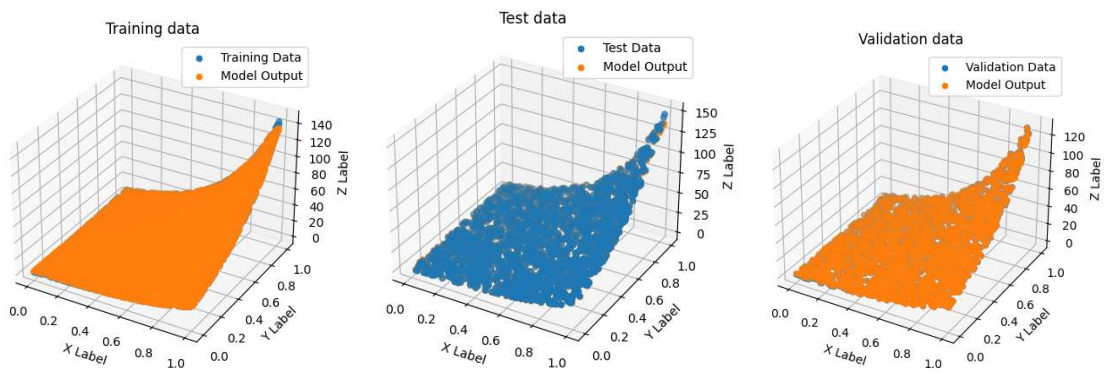


Figure 19: Model and Target outputs for each datasets

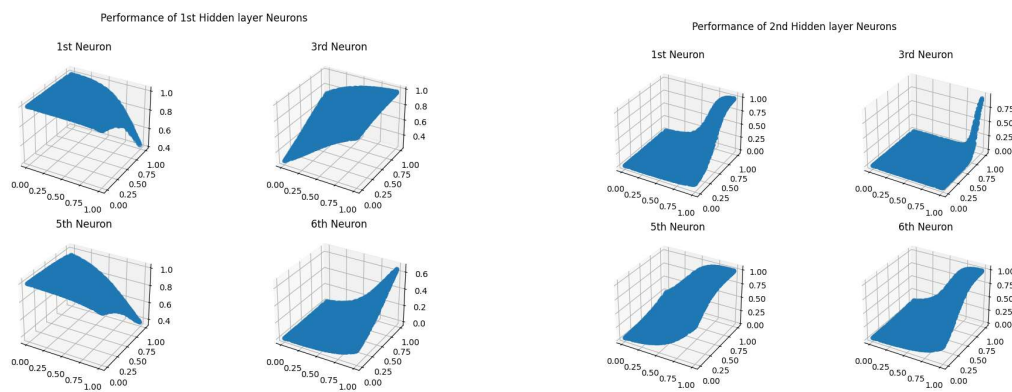


Figure 20: Performance of neurons in hidden layers

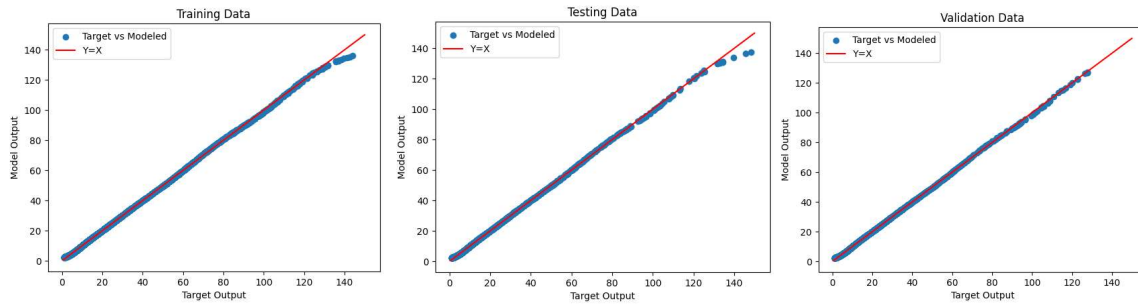


Figure 21: Scatter plots for Model and Target outputs for each datasets

Fig.21 shows that the samples fed into the model was clearly approximated onto a linear function i.e. Model output is following Target output linearly.

Comparison between Single neuron perceptron & FCNN

Parameter	Single Neuron	FCNN
Model vs Target Output	<p>Two 2D scatter plots for the Single Neuron model. The left plot is 'Modelled vs Targeted for Training' and the right is 'Modelled vs Targeted for Testing'. Both show 'Output values' on the Y-axis (1.0 to 4.0) and 'Input values' on the X-axis (0.0 to 1.0). The data points form a parabolic shape, deviating from the linear $Y=X$ line (red).</p>	<p>Two 2D scatter plots for the FCNN model. The left plot is 'Outputs for Univariate Training data' and the right is 'Outputs for Univariate Test data'. Both show 'Output values' on the Y-axis (10 to 40) and 'Input values' on the X-axis (0.0 to 1.0). The data points closely follow the linear $Y=X$ line (red).</p>
Scatter plot	<p>Two 3D surface plots for the Single Neuron model. The left plot is 'Modelled vs Targeted for Training' and the right is 'Modelled vs Targeted for Testing'. Both show 'Output' on the Z-axis (-25 to 150) and 'x1' and 'x2' on the X and Y axes (0.0 to 1.0). The surfaces show a non-linear, curved relationship.</p>	<p>Two 3D surface plots for the FCNN model. The left plot is 'Training data' and the right is 'Test data'. Both show 'Output' on the Z-axis (-25 to 150) and 'X Label' and 'Y Label' on the X and Y axes (0.0 to 1.0). The surfaces show a linear relationship.</p>