CS671: Deep Learning and Applications

Group20 Assignment1 Report

Aakash Maurya	*Tirthashree*	*Mohit Kumar*
B18042	*T19165*	*B18072*

Abstract

In this report, Perceptron and Multilayer feed-forward neural network (MLFFNN) models are applied for classification and regression tasks. Classification tasks were applied on three types of datasets i.e. Linearly separable, Non-linearly separable, and Image dataset. For the classification of the image dataset, the Bag-of-visual-words (BoVW) algorithm was used to convert images into 32-d BoVW vector representations of the image. Regression tasks were applied on two types of data i.e. Univariate and bivariate data. Model complexities and epochs were varied during the application of the subsequent models.

1 Theory

- **Classification Modeling:**
  It is the task of approximating a mapping function f, from the input variables x and discrete output variables y. In the classification problem, we require the classifier examples with having two or more than two classes. A classification model can easily predict the values as the probability of a given sample belonging to which of the output class and the predicted probability can be converted into the class value and can be assigned to the class with the highest probability. It can be evaluated using accuracy. We can also use the loss function to predict the output.

$$-(y\log(p)+(1-y)\log(1-p))$$

Formula for cross-entropy

This formula will give the value of error which we can use to predict output Where y is the true class and p is the probability. In this assignment, we have used MSE as the loss function for classification at times.

- **Regression Modelling:**
  It is the task of approximating a mapping function f, from the input variables x and continuous output variables y. In the regression problem,

we require the prediction of a quantity so the model will give us the value of the error. To calculate the error we use the Root mean square error because the unit of the error score is similar to the predicted value. So we can say that it can be evaluated using RMSE.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Formula for MSE

And then taking the root of MSE we can get the RMSE.
Where y^ is the predicted values and y is the observed value of the variable being predicted.

- **Perceptron:**
  It was invented in early 1958 by Frank Rosenblatt, at the Cornell Aeronautical Laboratory which was a science and technology company funded by the United States. At that time the perceptron was intended to be a machine instead of a neural network model. It was designed for image recognition.
  The perceptron model is a linear classifier so it can perform well in linearly separable data and does not perform well in non-linearly separable data.

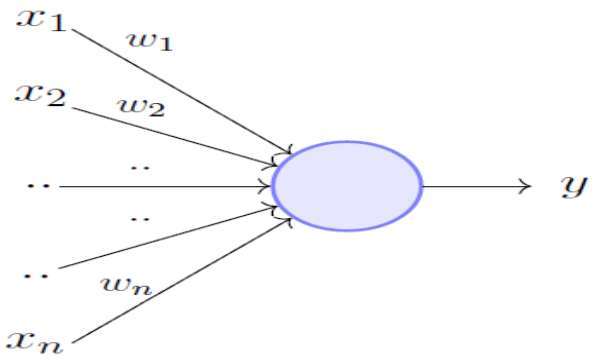


Fig. 1: Perceptron Learning Algorithm

Where (x1,x2,...., xn) are the input vectors and the (w1, w2, …., wn) are the weights and the learning rate of the perceptron is between 0 and 1. So the perceptron model takes the input as x and w of each input, takes the sum of weighted input and the activation function applied to it, and gives us the output.

- **MLFFNN (Multi-layer feed-forward neural network):**
  It is an interconnection of perceptrons and the flow of data and calculations in it is from the input data to output. As our single layer perceptron model works well only for linearly separable data and does not work well with non-linearly separable data so to overcome this failure MLFFNN is introduced to work on non-linearly separable data. In this model, we first divide the problem into smaller linearly separable data and then use perceptron for each linearly separable data and then combine them all. But in some cases, we are trying to solve more complicated problems of classification and regression and so for this, we can add more hidden layers and we can also use the non-linear activation function in the hidden layers. So that's why we need MLFFNN.

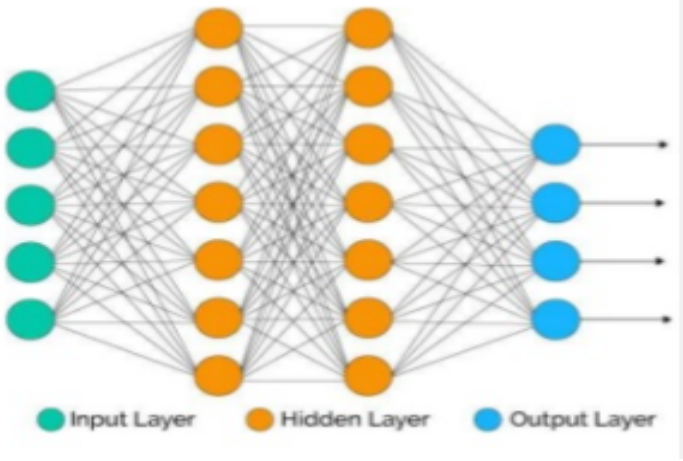


Fig 2. : Structure of MLFFNN

Where the green points are the input layers i.e. x and the orange points are the hidden layers used by us and the blue points indicate the output layer i.e. y.
The performance of the model can be evaluated by calculating the error function or loss function. By choosing the right error functions we can help our algorithm to get the best values for the weights. We can calculate the error function with the help of Sum squared error or cost entropy. We use the Gradient Descent algorithm to calculate the new values of weights and the activation function must be a defined and differentiable function.

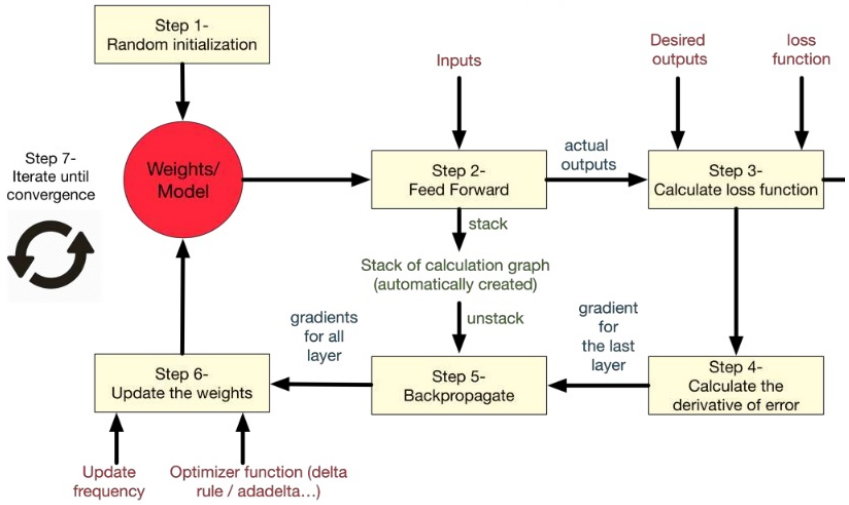


Fig 3. : Learning process of ANN

- **Backpropagation:**
  It is the most commonly used algorithm for training the feed-forward neural networks. It computes the gradient of the loss functions with respect to the weights of the neural network for single input and output and computes the loss function so efficiently. Due to this high efficiency, it is used as a gradient method for training the multilayer neural networks and updating the weights to minimize the loss function of the neural networks. But it has a problem that we may not be able to get the gradient descent with backpropagation to find the global minimum error for the error function. It also requires the derivatives of the activation function which we have used in our neural network model.

2 Data

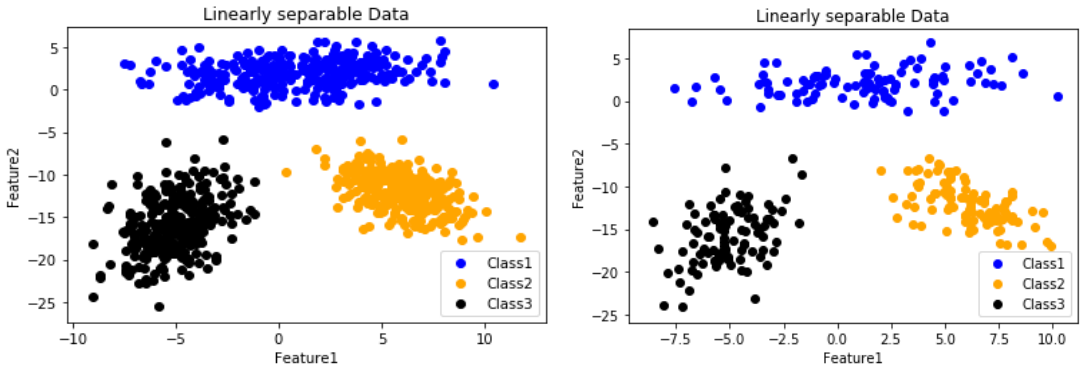


Fig 4. : Linear Classification Train and validate data

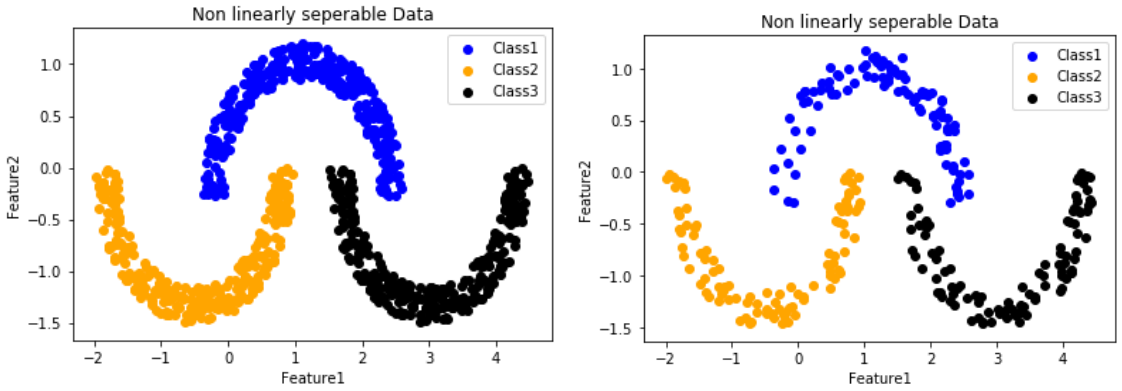


Fig 5. : Non-Linear Classification Train and validate data

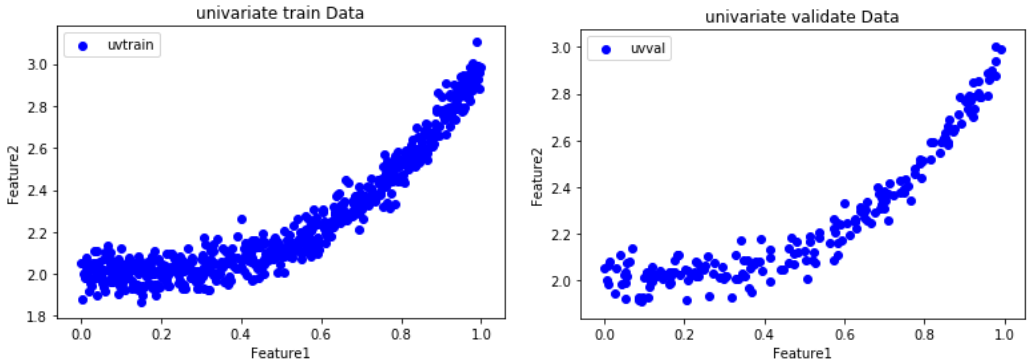


Fig 6. : Univariate Regression Train and validate data

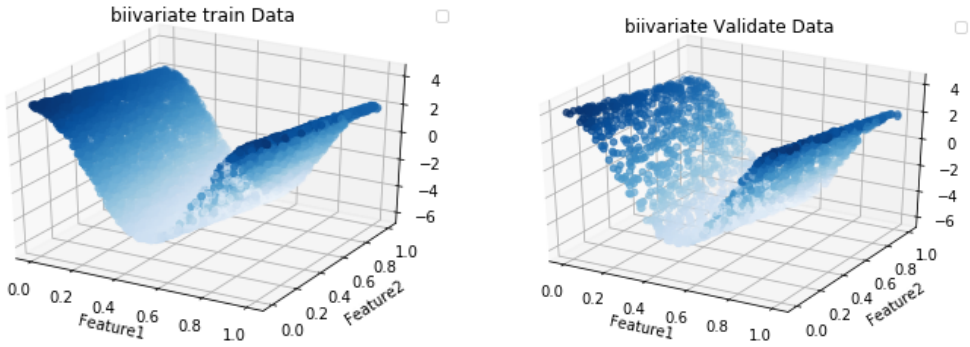


Fig 7. : Bivariate Regression Train and validate data

The image dataset corresponded to 3 classes:

1. Badminton Court
2. Cemetery
3. Music Stores.

3 Approach and Considerations

3.1 Bag of Visual Words (BoVW)

1. Converted images into 32-dimensional vectors using the algorithm mentioned in the theory.
2. The resultant 32-dimensional BoVW representation vector of each image was saved into a CSV file.
3. This file was further extracted to apply perceptron and MLFFNN models during the classification tasks of the image data.

3.2 Classification Tasks

1. Perceptron Model

    i. Datasets used for classification tasks using the perceptron model were Linearly separable Dataset and Non-linearly separable dataset.
    ii. We have used the one vs rest approach in this classification task.
    iii. There were 3 output classes, so 3 perceptron models were trained corresponding to each class.
    iv. The output of each perceptron model can be considered as the probability of the input belonging to the corresponding class.
    v. The final output was taken as the class with the highest probability.
    vi. The loss function used in the perceptron model is the Mean Squared Error function.
    vii. The hyperparameters were:
        - Number of epochs: 100, 250, 500, 750, 1000, 1500, 2000, 2500 for all datasets.
        - Learning rate: 0.01 for all datasets.
        - Activation function: Sigmoidal for all datasets.

2. MLFFNN model

    i. Datasets used for classification tasks using the perceptron model were Linearly separable Dataset, Non-linearly separable dataset, and Image dataset (regenerated using BoVW).
    ii. The loss function used in the MLFFNN model is the Cross-Entropy function.

iii. The hyperparameters were:
- Number of epochs: Up to 300
- Learning rate: 0.005
- Activation function: Sigmoidal for all datasets.
- Number of hidden layers: 2
- Number of hidden nodes:
  - Linearly and nonlinearly data: 4 in layer 1 and 3 in layer 2
  - Image data: 38 in layer 1 and 36 in layer 2

3.3 Regression Tasks

1. Perceptron Model

i. Datasets used for regression tasks using the perceptron model were the Univariate Dataset and Bivariate dataset.
ii. The loss function used in the perceptron model is the Mean Squared Error function.
iii. The hyperparameters were:
- Number of epochs: 100, 250, 500, 750, 1000, 1500, 2000, 2500 for all datasets.
- Learning rate: 0.01 for all datasets.
- Activation function: Linear for all datasets.

2. MLFFNN model

i. Datasets used for regression tasks using the MLFFNN model were the Univariate Dataset and Bivariate dataset.
ii. The loss function used in the MLFFNN model is the ...
iii. The hyperparameters were:
- Number of epochs: 10, 50, 75, 100, 150 for all datasets
- Learning rate: 0.01
- Activation function: Linear for all datasets
- Number of hidden layers: 2
- Number of hidden nodes: 4 in layer 1 and 2 in layer 2

4 Observations

4.1 Classification Tasks

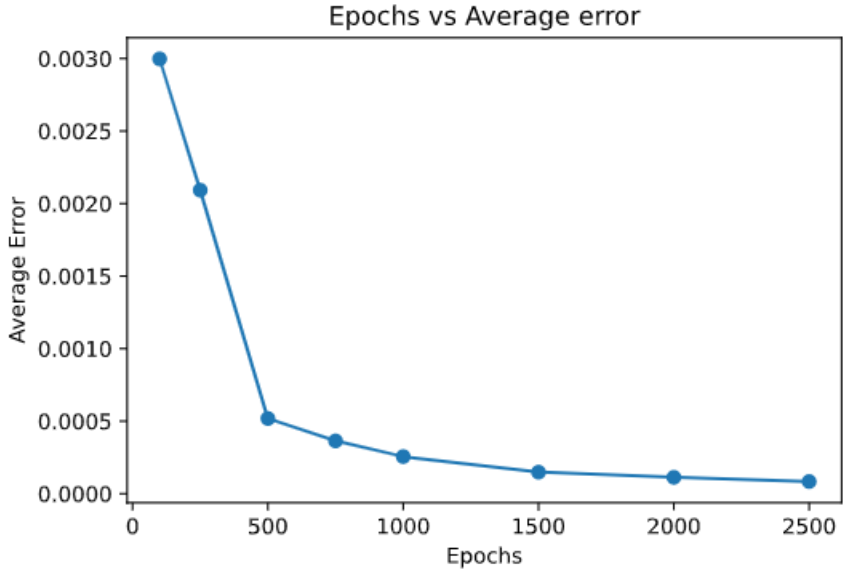
4.1.2 Linearly Separable Dataset

1. Perceptron Model

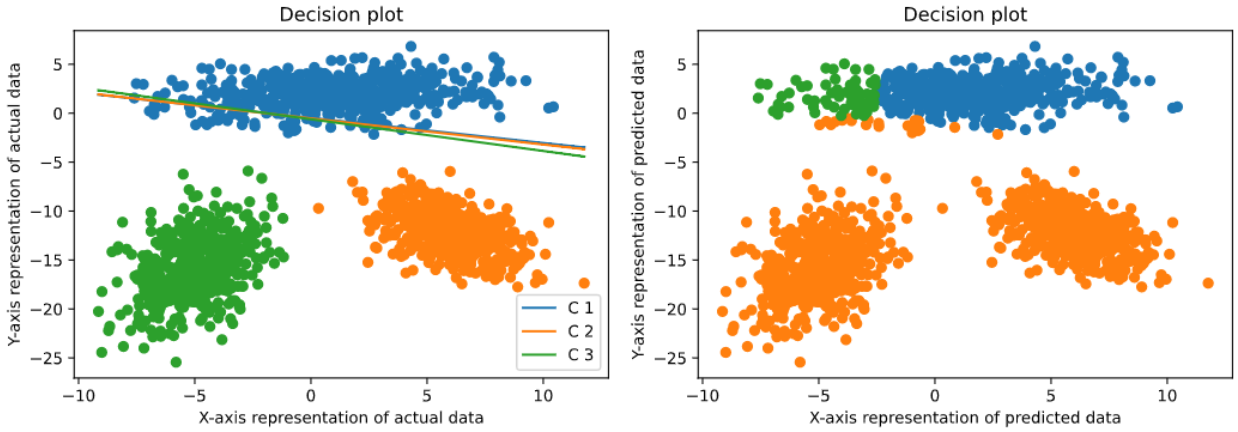
   i. Best hyperameter: Epochs = 1000
   ii. Average classification accuracy = 61.67%

   iii. Confusion matrix: [[85 5 10]
   [0 100 0]
   [0 100 0]

   iv. Plot of epochs vs average error:



   v. Decision region plots (decision boundary lines):



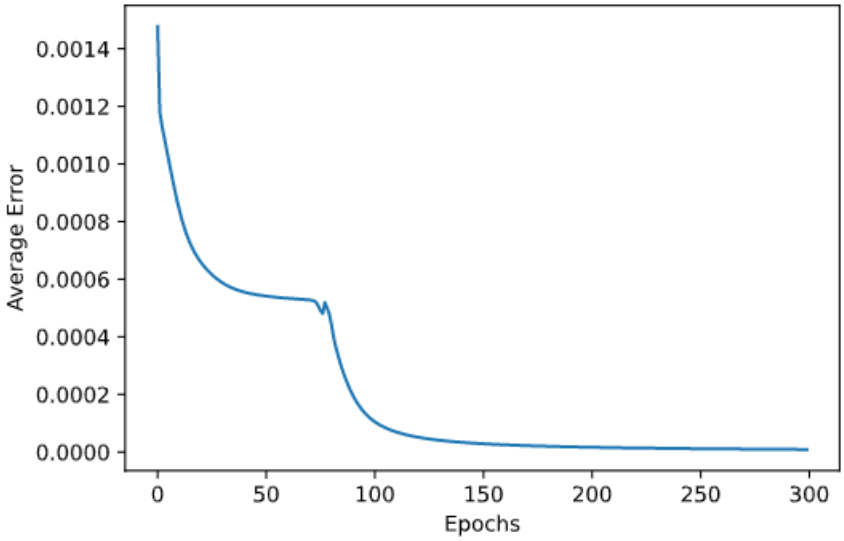
2. MLFFNN Model

i. Best hyperameter: Epochs = 300

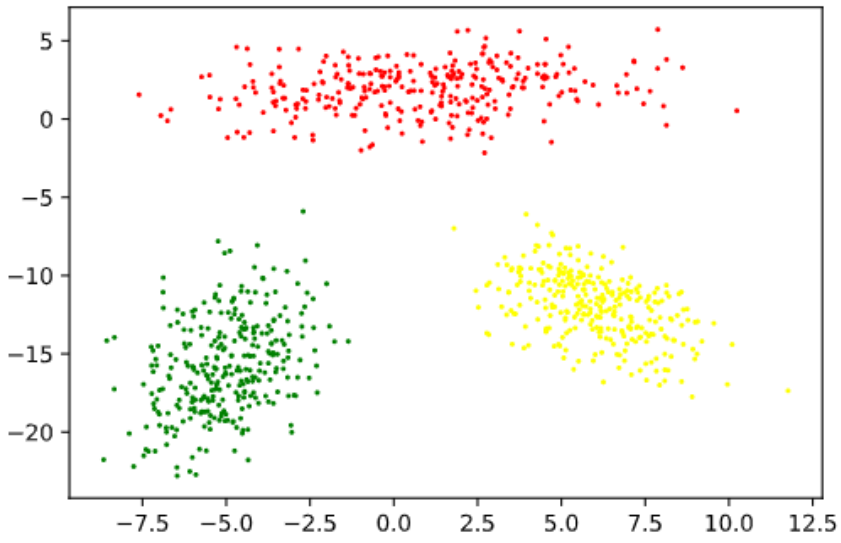
ii. Average classification accuracy = 100.00%

iii. Confusion matrix: [[94 0 0]
[0 105 0]
[0 0 101]]

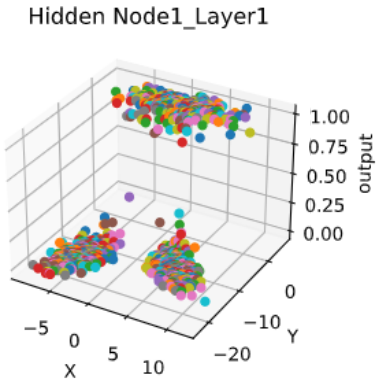
iv. Plot of epochs vs average error:

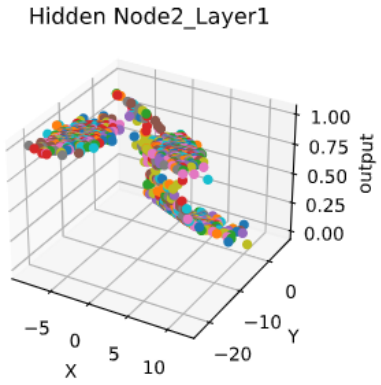


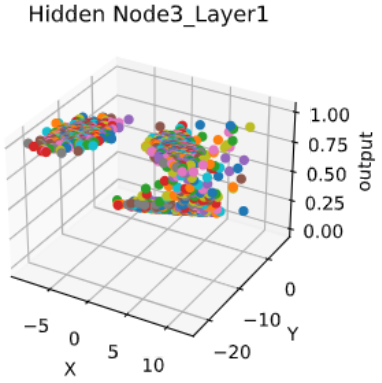
v. Decision Plot (colored points in each region):

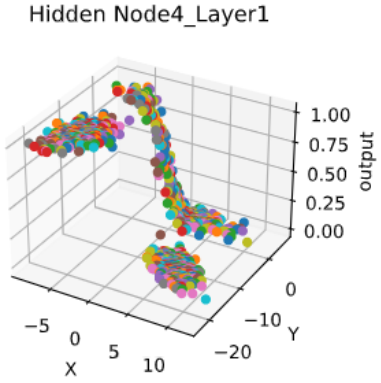


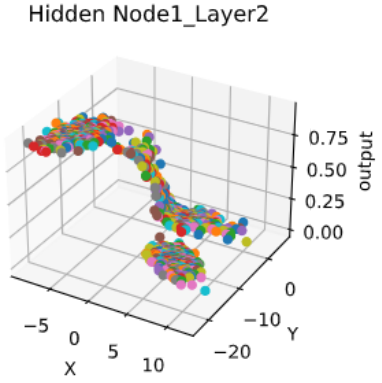
vi. Output of hidden layers:

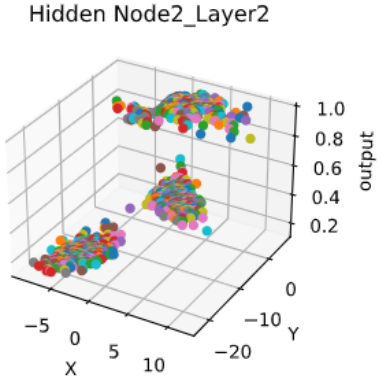


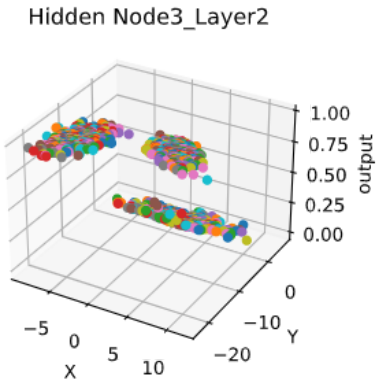








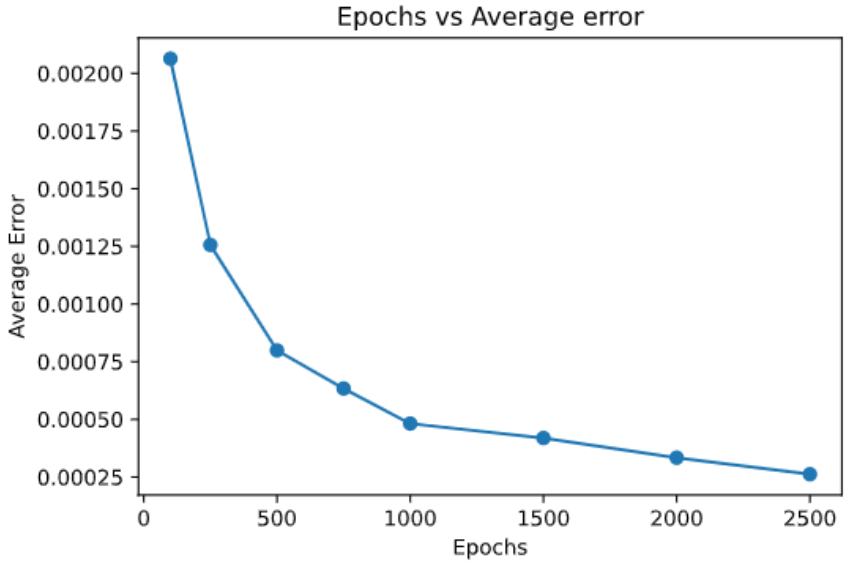




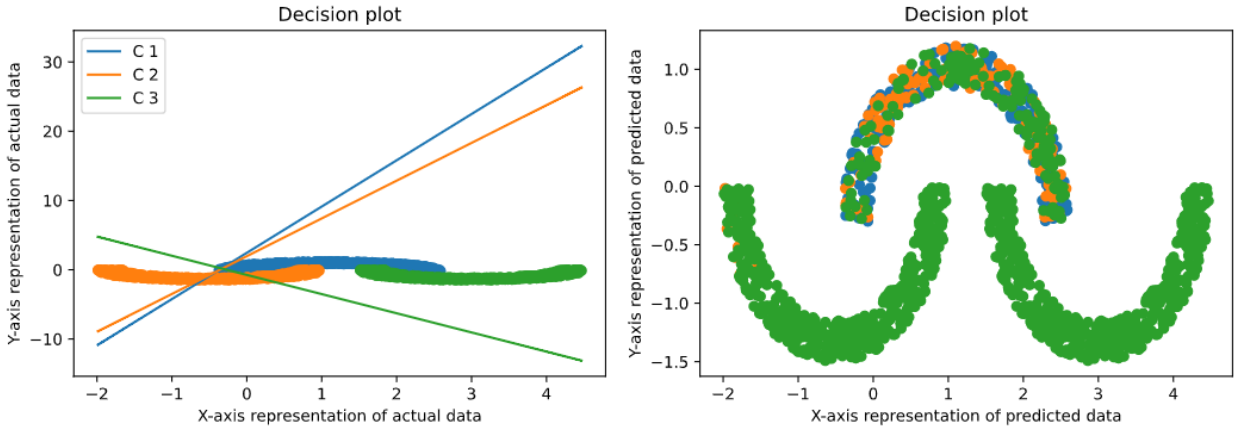
4.1.2 Non-linearly Separable Dataset

1. Perceptron Model

   i. Best hyperameter: Epochs = 1000
   ii. Average classification accuracy = 40.00
   iii. Confusion matrix: [[2 19 79]
   [0 18 82]
   [0 0 100]
   iv. Plot of epochs vs average error:



   v. Decision region plots (decision boundary lines):

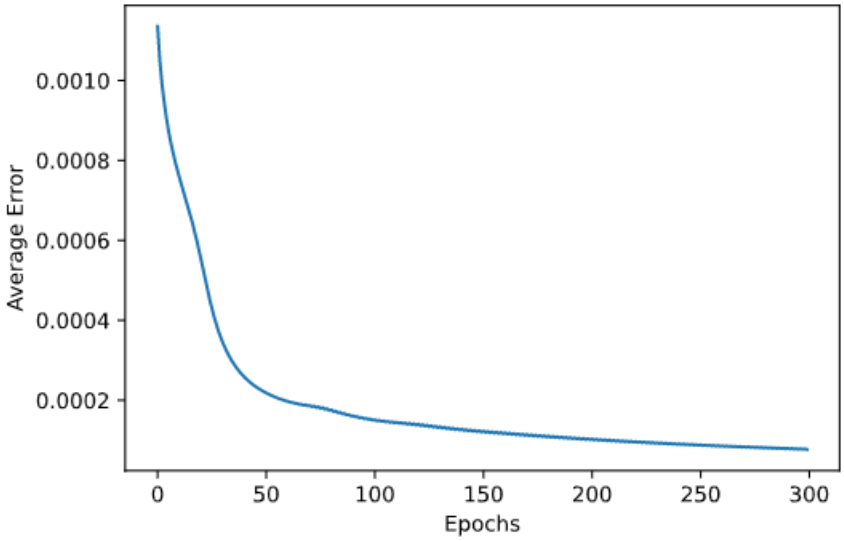


2. MLFFNN Model

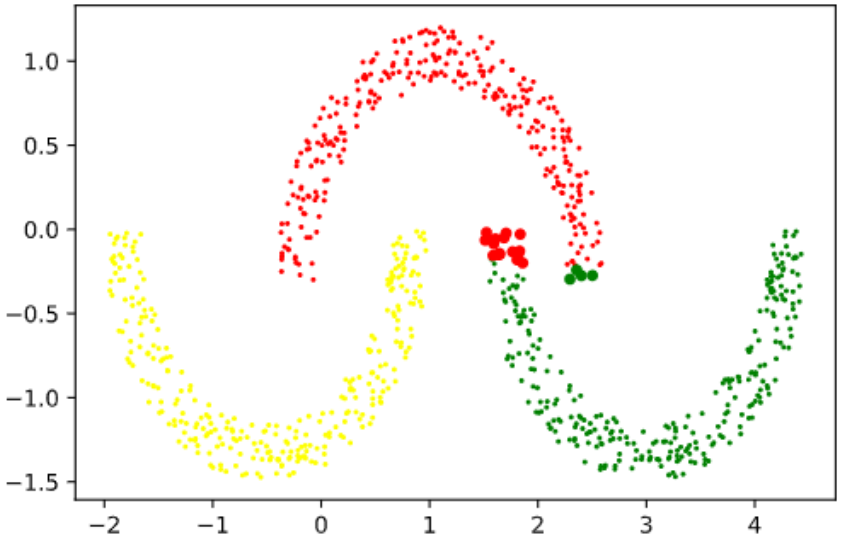
   i. Best hyperameter: Epochs = 300
   ii. Average classification accuracy = 97.67%

iii. Confusion matrix: [[92 0 9]
[0 105 0]
[2 0 92]]

iv. Plot of epochs vs average error:

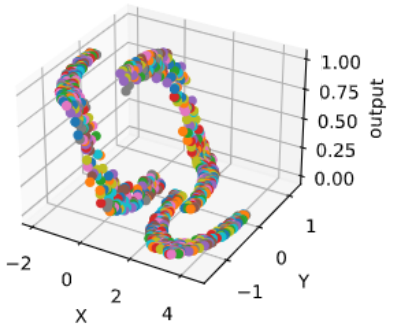


v. Decision Plot (colored points in each region):

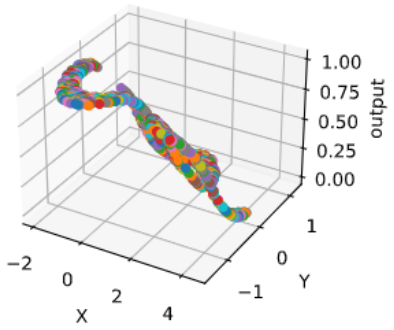


vi. Output of hidden layers:

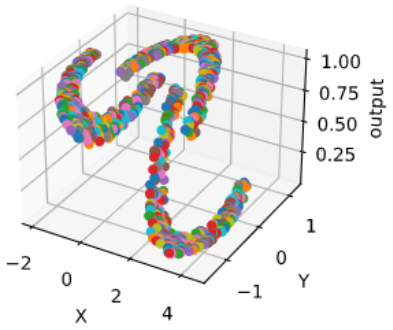
Hidden Node1_Layer1



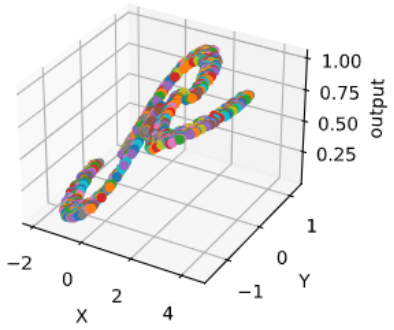
Hidden Node2_Layer1



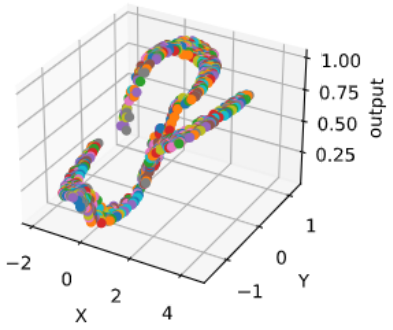
Hidden Node3_Layer1



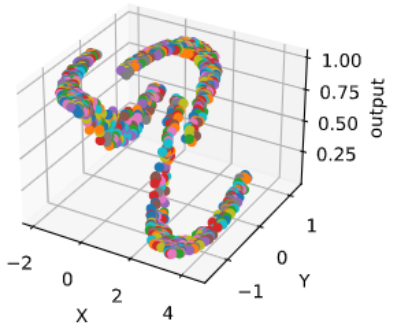
Hidden Node4_Layer1



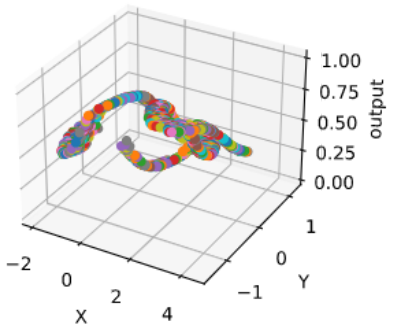
Hidden Node1_Layer2



Hidden Node2_Layer2



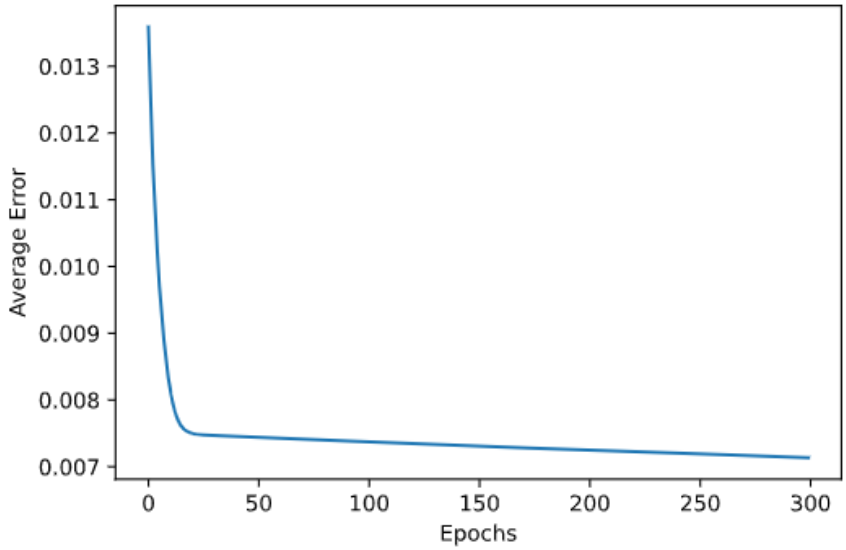
Hidden Node3_Layer2



4.1.3 Image Dataset

1. MLFFNN Model

   i. Best hyperameter: Epochs = 300
   ii. Average classification accuracy = 57.33%
   iii. Confusion matrix: [[0 4 1]
   [0 10 16]
   [0 11 33]]
   iv. Plot of epochs vs average error:



   v. Decision Plot: 32-dimensional plot not interpretable
   vi. Output of hidden layers: More than 32 nodes in each layer so not plotted

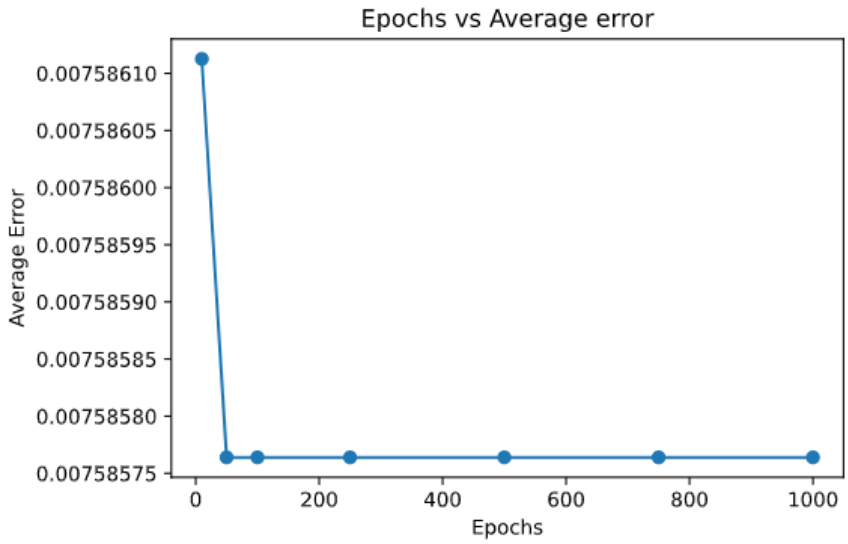
4.2 Regression Tasks

4.2.1 Univariate Dataset

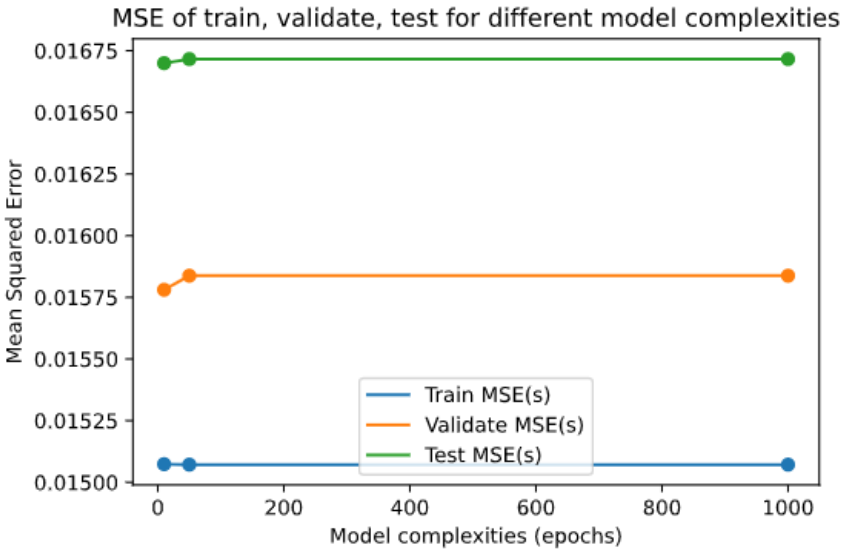
1. Perceptron Model

   i. Best hyperameter: Epochs = 10
   ii. MSE = 0.0167

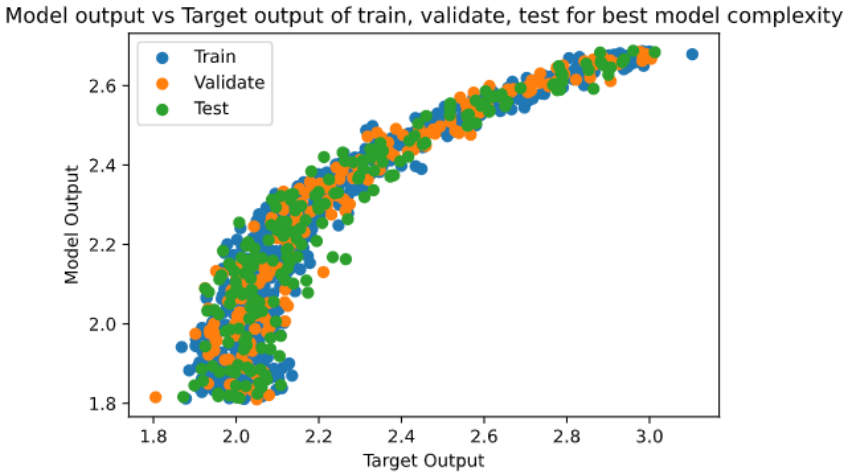
iii. Plot of epochs vs average error:



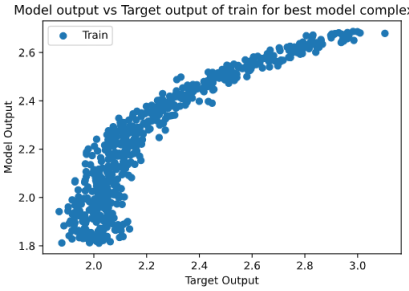
iv. MSE for different model complexities:

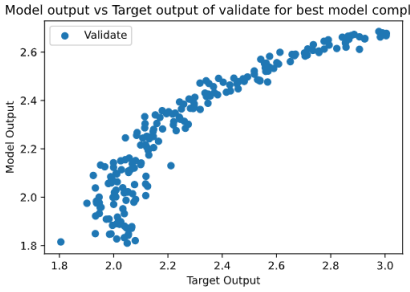


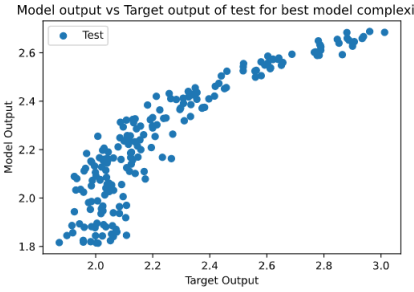
v. Model output vs Target output:



vi. Model output vs Target output (individually):



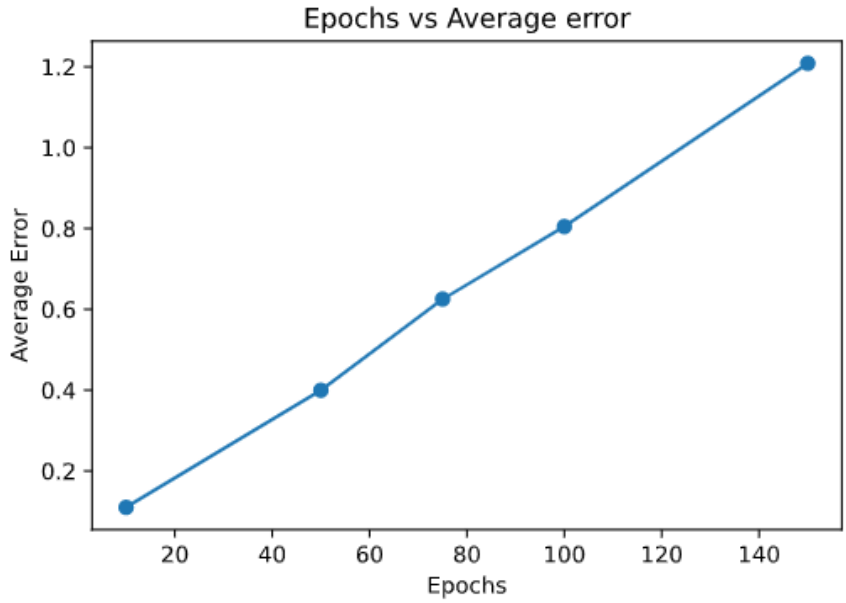




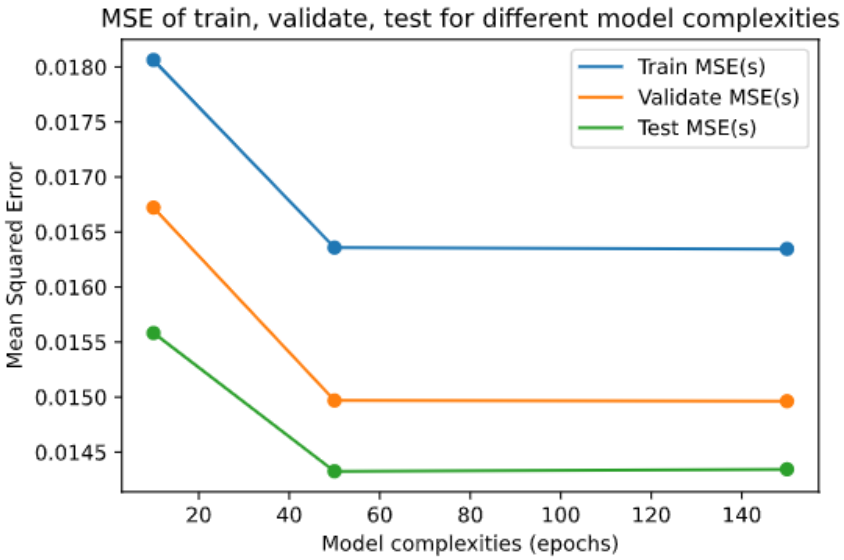
2. MLFFNN Model

i. Best hyperameter: Epochs = 10
ii. MSE = 0.0158

iii. Plot of epochs vs average error:

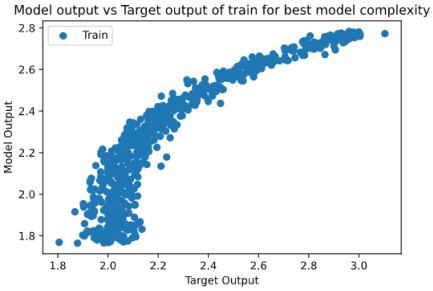


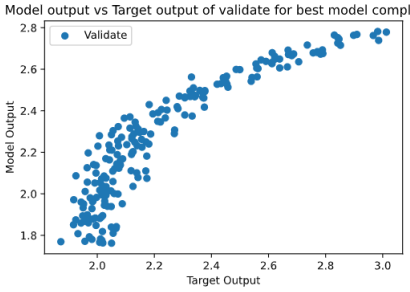
iv. MSE for different model complexities:

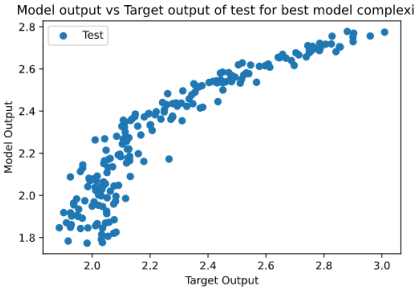


v. Model output vs Target Output:

vi. Model output vs Target Output (individually):





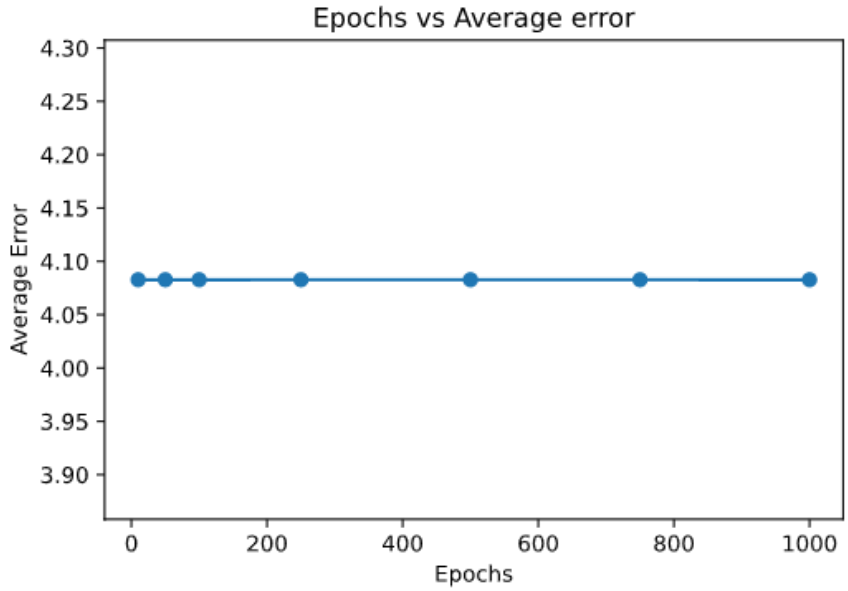


4.2.2 Bivariate Dataset

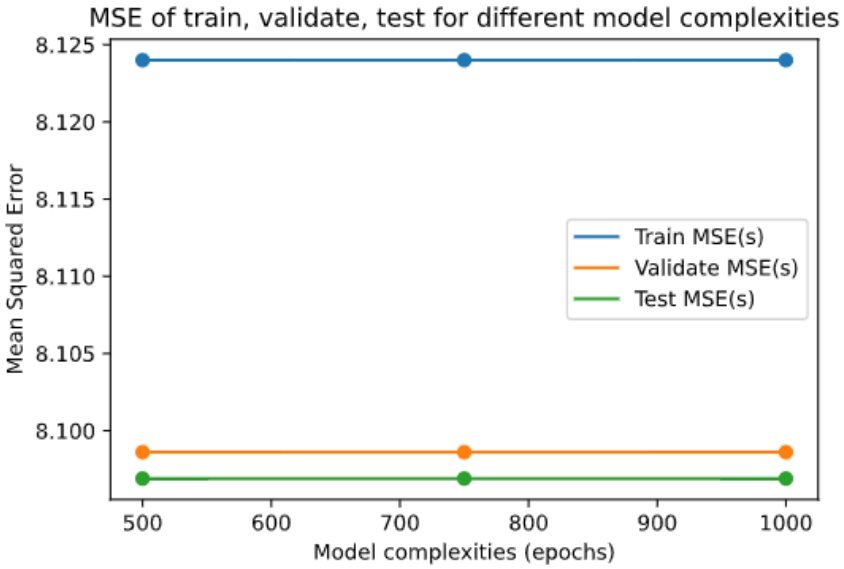
1. Perceptron Model

i. Best hyperameter: Epochs = 10
ii. MSE = 8.097

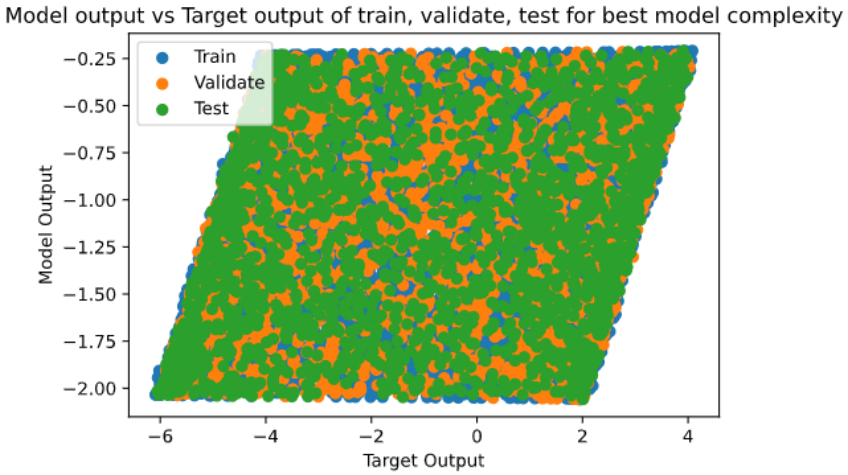
iii. Plot of epochs vs average error:



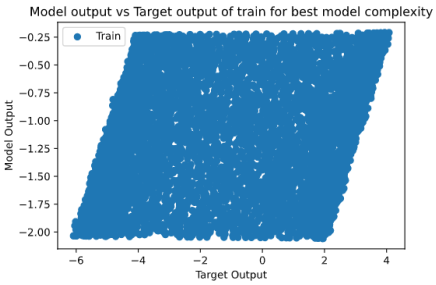
iv. MSE for different model complexities:

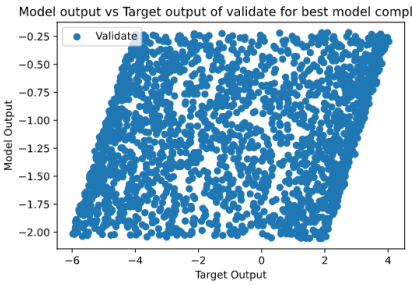


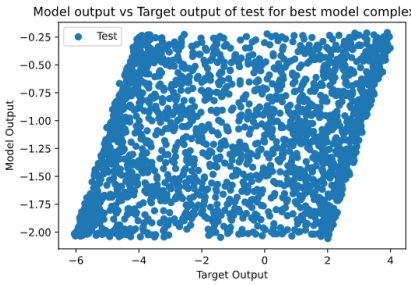
v. Model Output vs Target Output:



vi. Model Output vs Target Output (individually):



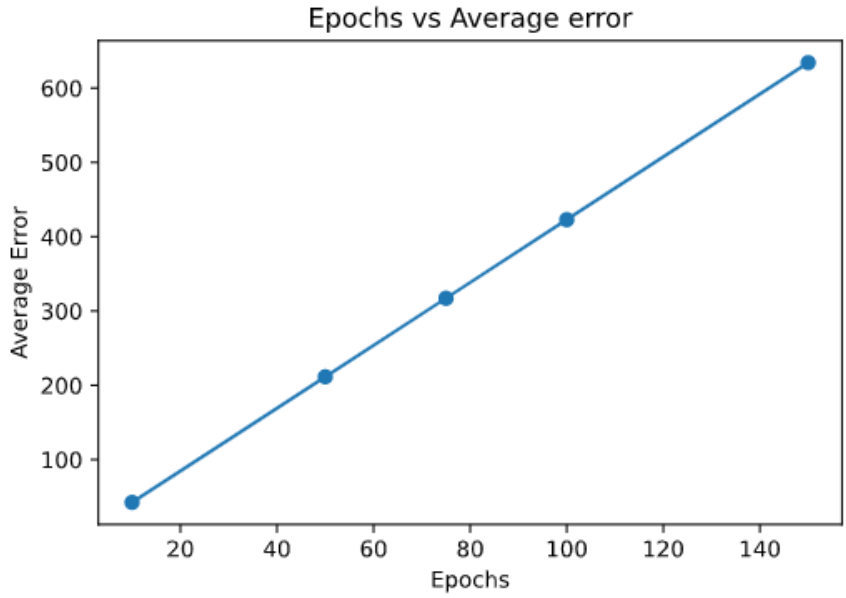




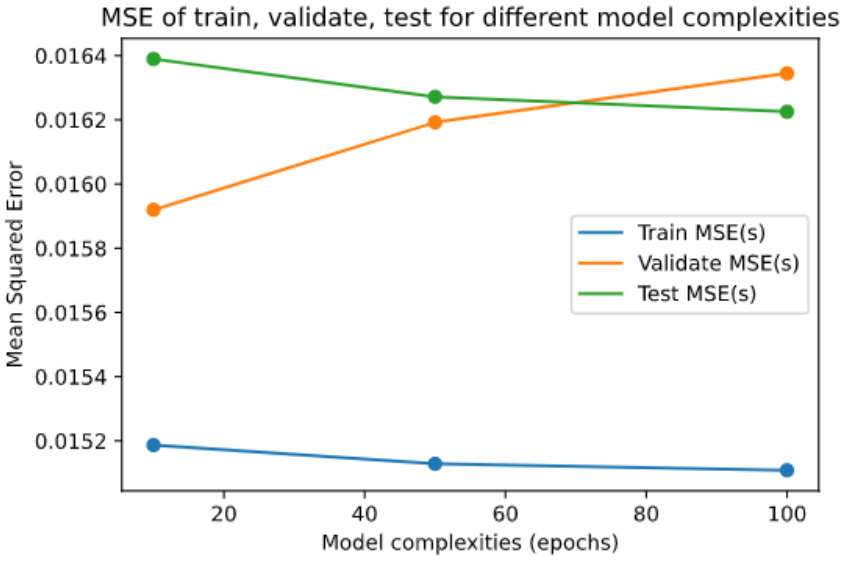
2. MLFFNN Model

i. Best hyperameter: Epochs = 10
ii. MSE = 8.381

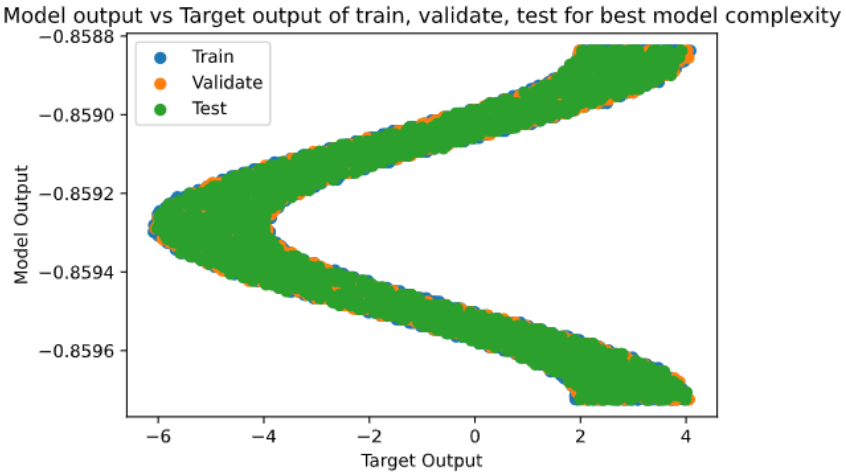
iii. Plot of epochs vs average error:



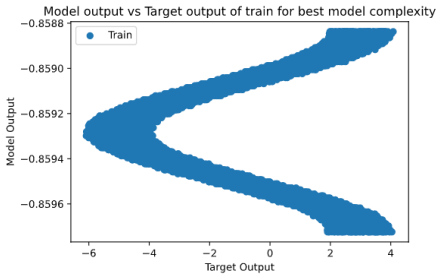
iv. MSE for different model complexities:

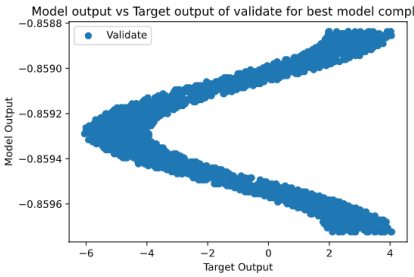


v. Model output vs Target Output:



vi. Model output vs Target Output (individually):





5 Inferences

1. For classification tasks, with the hyperparameters we used both Perceptron and MLFFNN model provided satisfactory results.
2. In the linearly separable dataset, classification using the perceptron model gave an accuracy of 61.67% as compared to 100.0% that of MLFFNN. Hence, neural networks are good classifiers of the linearly separable dataset, and adding more layers helps get better results.
3. In the non-linearly separable dataset, classification using the perceptron model gave an accuracy of 40.00% as compared to 97.67% that of MLFFNN. Hence, the perceptron model lacks considerably behind the MLFFNN model when classifying non-linearly separable datasets.
4. In the image dataset classification, the MLFFNN model gave an accuracy of 57.33%. Hence, MLFFNN is an average classifier for such an image dataset.
5. BoVW representation of images gives significant information about the original image and helps reduce the size of the dataset (100 MB to 100 KB).

6. For regression tasks, with the hyperparameters we used both Perceptron and MLFFNN model did not provide satisfactory results. The average error increased on an increasing number of epochs in our chosen range.
7. In the univariate dataset, regression using the perceptron model gave an MSE of 0.0167 as compared to 0.0158 that of MLFFNN. Hence, neural networks are good classifiers of the univariate dataset, and adding more layers helps get marginally better results.
8. In the bivariate dataset, classification using the perceptron model gave an MSE of 8.097 as compared to 8.381 that of MLFFNN. Hence, complex regression datasets require a more complex structure of neural networks.

6 References

I. https://en.wikipedia.org/wiki/Perceptron#Learning_algorithm
II. https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975
III. https://www.slideshare.net/TamerAhmedFarrag/04-multilayer-feedforward-networks
IV. https://en.wikipedia.org/wiki/Backpropagation
V. https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/#:~:text=Classification%20predictive%20modeling%20is%20the,category%20for%20a%20given%20observation.