Report Assignment 3 Pattern Recognition & Machine learning

Riccardo Guderzo GE24Z227
Joanna Kolaczek GE24Z229
Miguel Mauer GE24Z022

Task 1: Function Approximation using MLFFNN

Dataset 1

- Function Approximation
  - MLFFNN
  - 2d data (Assignment 1 Dataset 2a)
  - 1 Hidden Layer 8 nodes

Exploratory analysis

We conducted a small exploratory analysis to get an idea of the distribution of the training data.

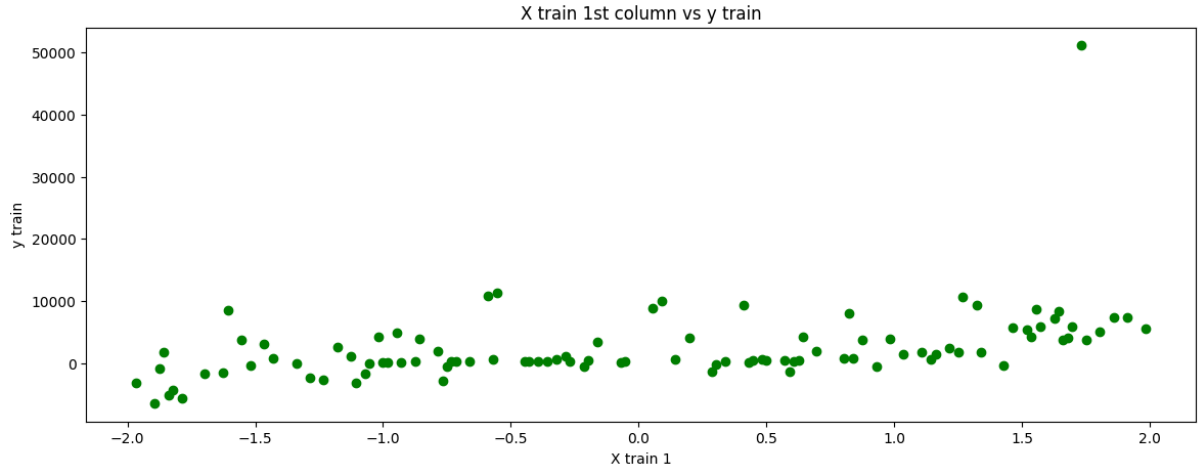


Fig 1

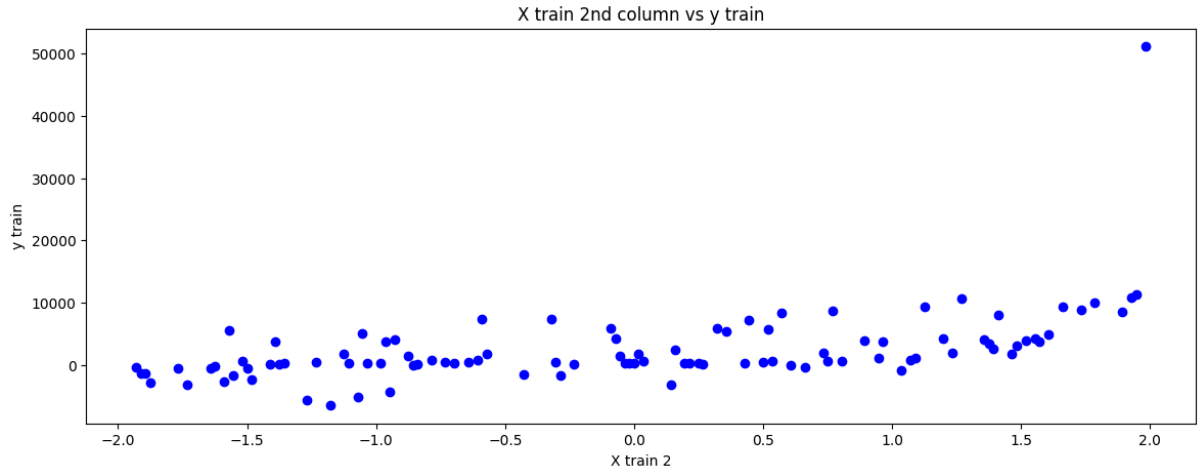


Fig 2

In both cases we can spot an outlier. This might influence the fitting of the neural network (we tried and received a very high MSE). Therefore we tried to normalize the data, so as to have a smoother convergence. Obviously that outlier can create problems in particular when the difference between the “average” points is so big (around 4000!). Normalizing will help the model to fit better and to decrease the variance.

In this case we have to define a function Approximation for Dataset 1 using MLFFNN with one hidden layer having 8 nodes. The Neural network designed is the following:

Layer (type)	**Output Shape**	**Param #**
Linear-1	[-1, 1, 8]	24
Tanh-2	[-1, 1, 8]	0
Linear-3	[-1, 1, 1]	9
Total parameters: 33		

Table 1

- ## Training error (ξav) vs epoch

NB: the learning rate is very high (0.7!). This can cause some troubles. One example is Fig 3: high learning rate can cause the model to oscillate around a solution rather than converging, resulting in the fluctuations of the loss function (the model to overcorrect rather than gradually minimizing the error). It can therefore result in an overshooting (skipping the optimal solution due to large updates).

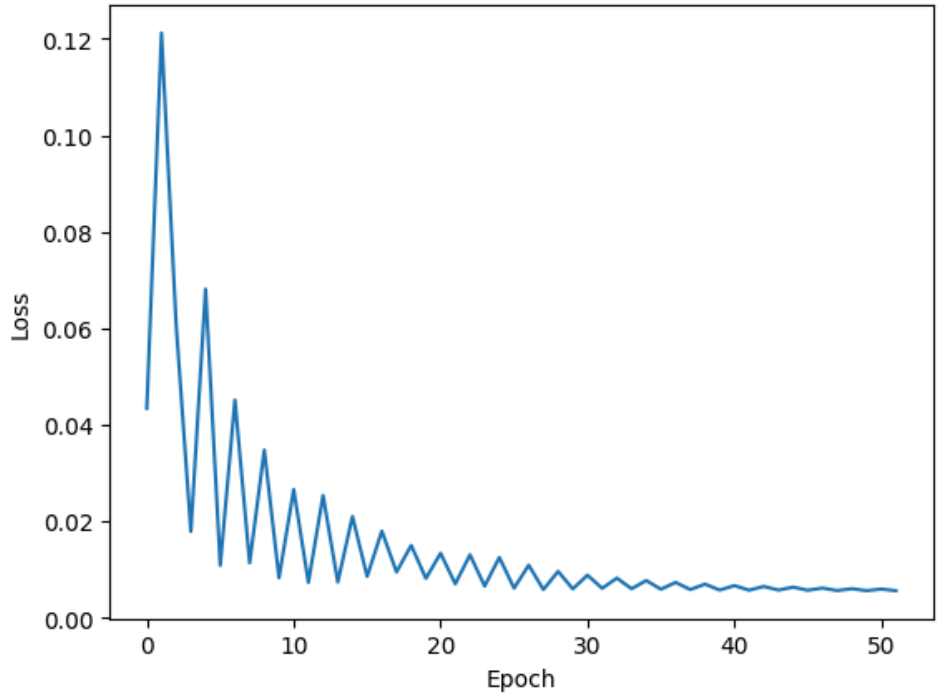


Fig 3

Below (Fig 4), we used a smaller learning rate (0.01). In this case it can be seen that the loss function converges much smoother than the curve on Fig 3.

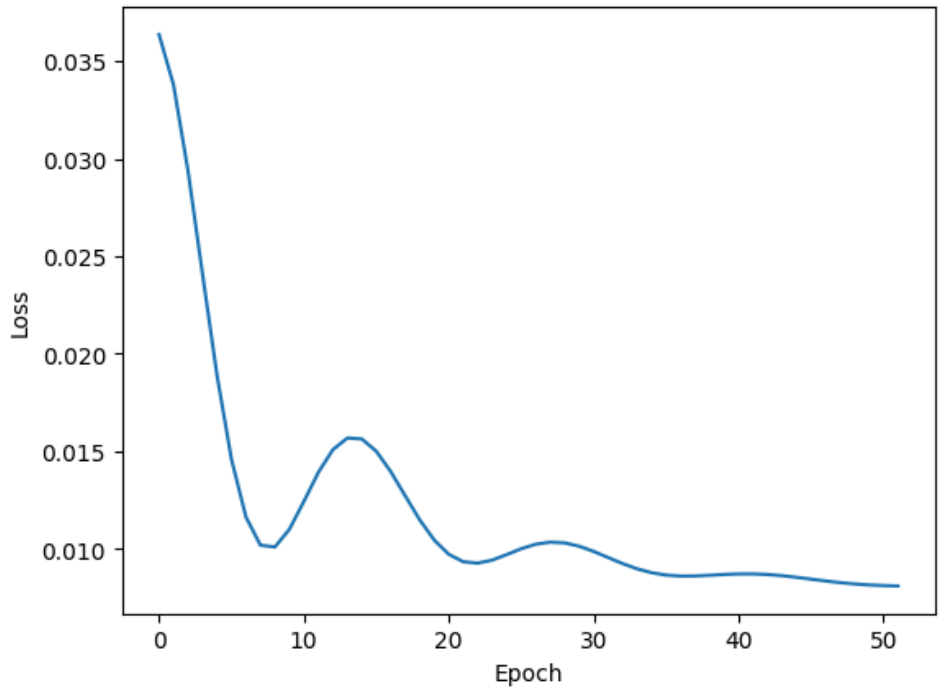


Fig 4

In any case, for the rest of the assignment we kept the model trained with a LR = 0.7 (as required). Below we reported the performance indexes of the model:

Training data:
Final RMSE: 0.0783
Final R-squared: 0.4579
Test data:
Final RMSE: 0.0892
Final R-squared: 0.4762

- Scatter plots for the training data and the test data

Fig 5 and Fig 6 represent the fit of the values for training and testing respectively. In both the cases it can be seen that the outlier “creates some troubles”, as the model is not able to fit effectively that value. That might be also the problem for the low R squared.

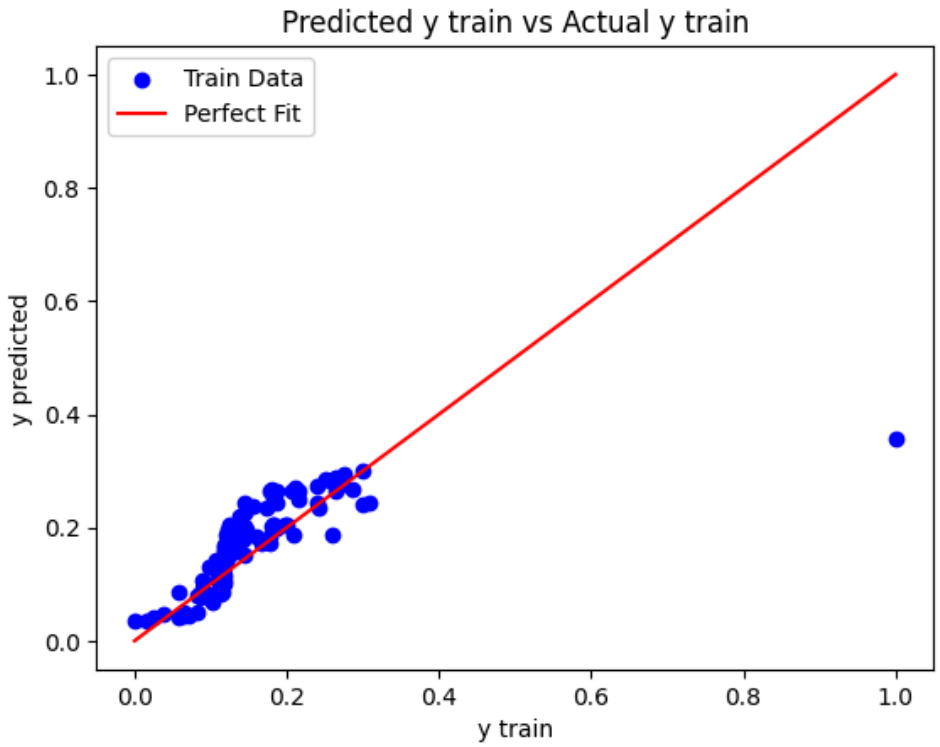


Fig 5

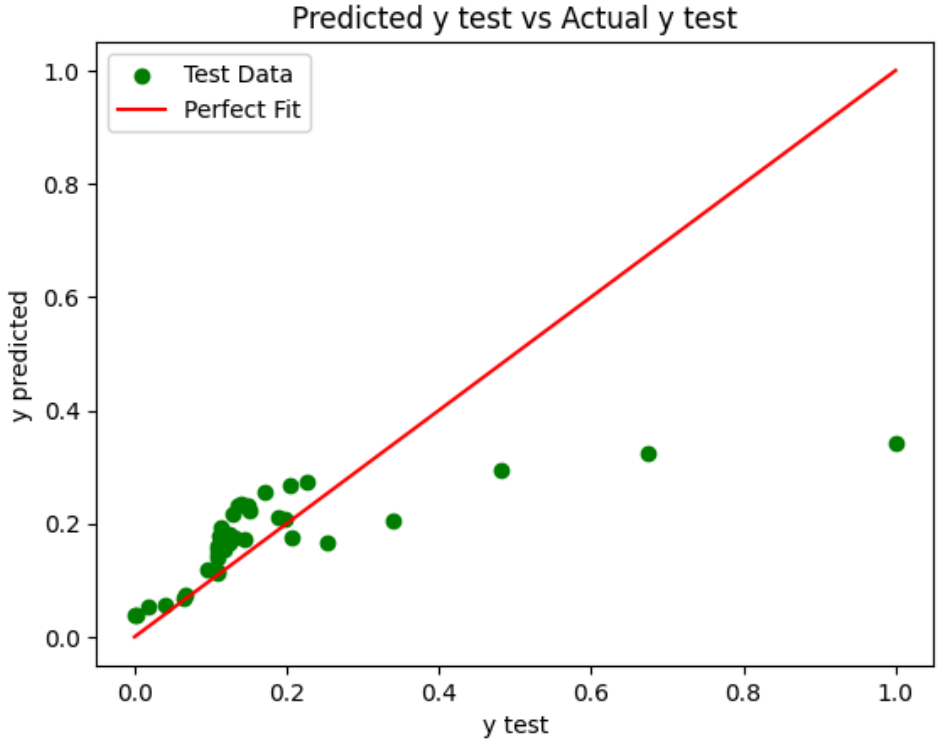


Fig 6

Since the fit was not very good.We also tried to reduce the learning rate and add a Sigmoid activation function on the output (since the data are normalized). The Network is therefore as follow:

Layer (type)	Output Shape	Param #
Linear-1	[-1, 1, 8]	24
Tanh-2	[-1, 1, 8]	0
Linear-3	[-1, 1, 1]	9
Sigmoid-4	[-1, 1, 1]	0
Total parameters: 33		

The results though have not improved. It is very likely to be the outlier shown above in Fig 1 and Fig 2 that make it difficult for the Net to converge properly.

- 4 surface plots (after Epochs 1, 10, 50, and convergence)

From Fig 7 to Fig 10 we can get an insight of how the Neural Net is learning.

We can observe that since the first epoch the Neural Net is able to get some interesting patterns that are then redefined during the training process.

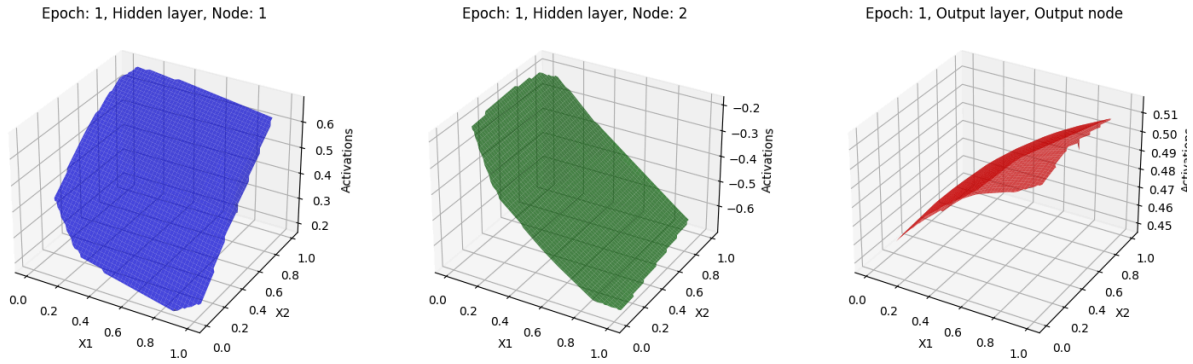


Fig 7

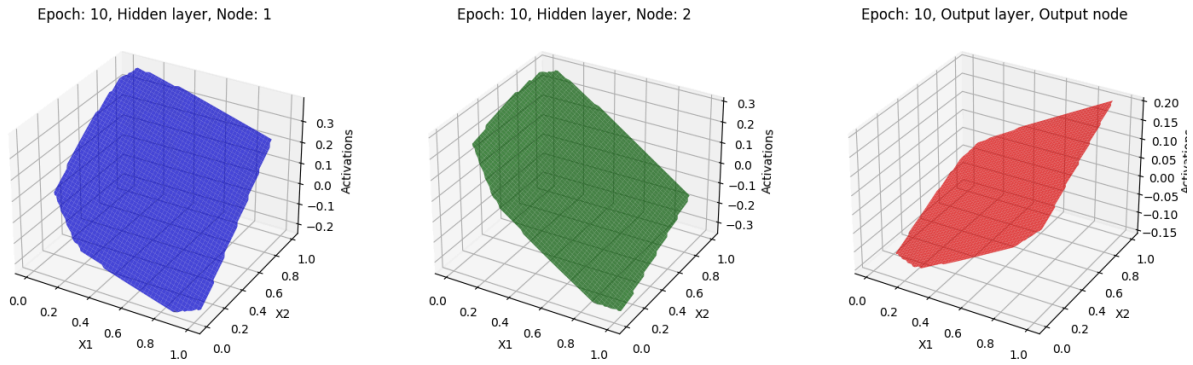


Fig 8

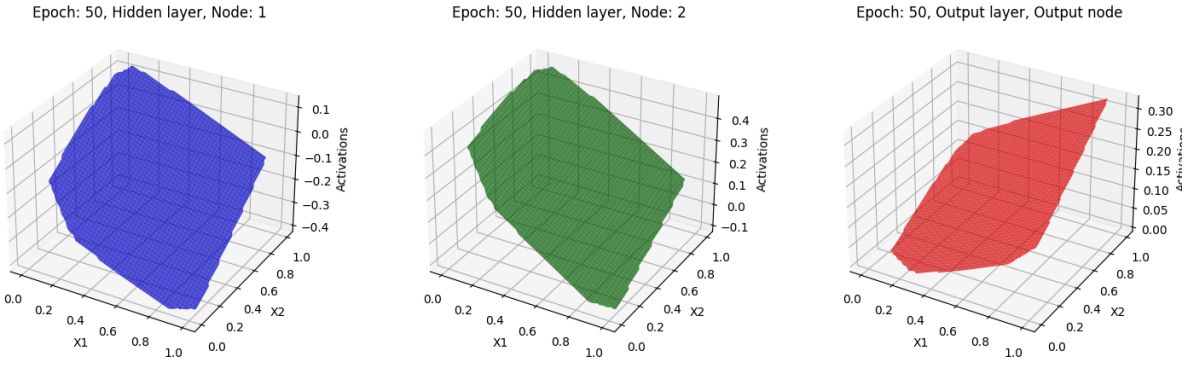


Fig 9

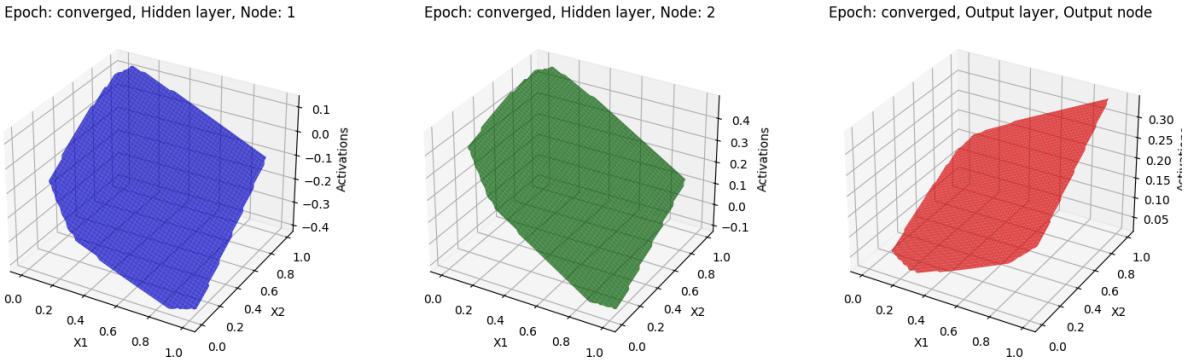


Fig 10

Task 2: Function Approximation using MLFFNN

Dataset 2

Plotting the data

We have plotted the data for every dimension of X, to get some insights about the distribution of the data (eg: Fig 11 represents the first dimension vs y of the training dataset). No particular features have emerged. In any case, we decided to normalize the data to get a better scale.

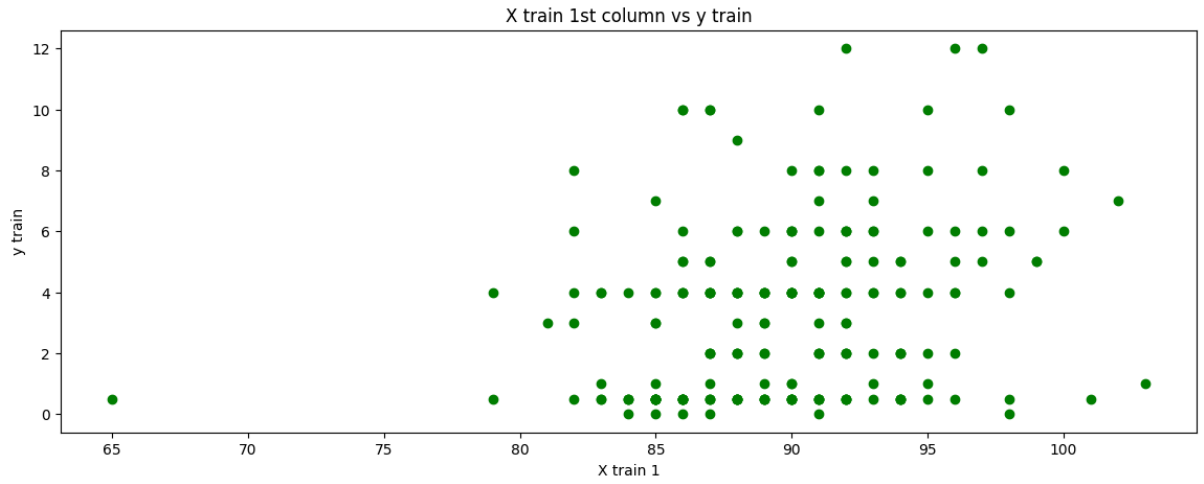


Fig 11

The Neural network designed is the following:

Layer (type)	**Output Shape**	**Param #**
Linear-1	[-1, 1, 15]	90
Tanh-2	[-1, 1, 15]	0
Linear-3	[-1, 1, 10]	160
Tanh-4	[-1, 1, 10]	0
Linear-5	[-1, 1, 1]	11
Total parameters: 261		

Table 2

- ## Training error (ξav) vs epoch

Even in this case (Fig 12) we have an oscillating loss function. This is due to the quite large learning rate (as explained above). In particular we have had run the training many times to get quite good results and not big oscillations (or loss function tending to infinity).

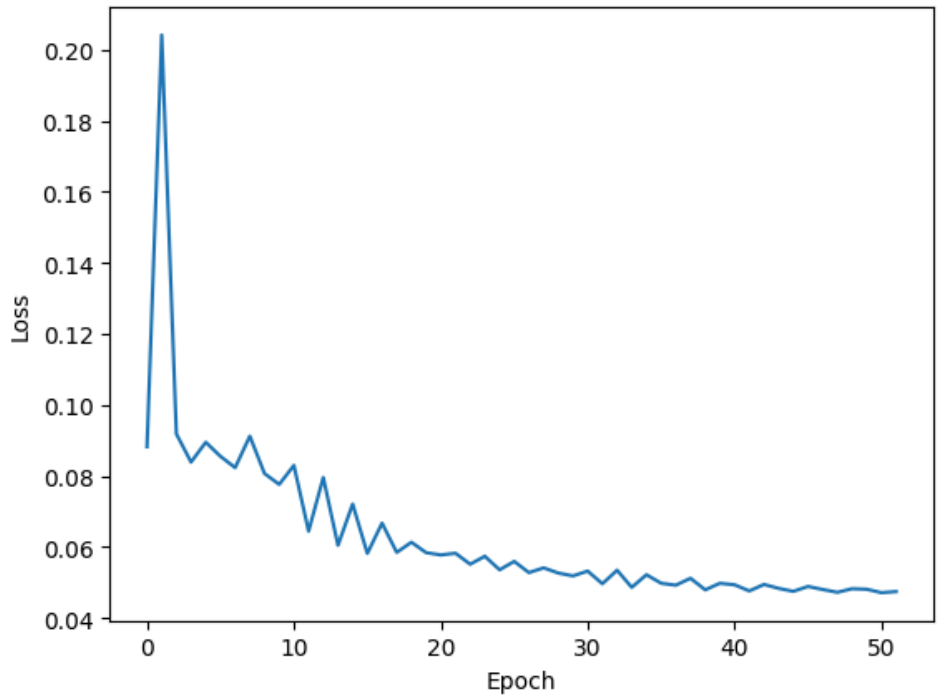


Fig 12

Below we reported the performance indexes of the model:

Training data:
Final RMSE: 0.2189
Final R-squared: 0.1413
Test data:
Final RMSE: 0.2993
Final R-squared: 0.1627

The R squared shows that the model does not explain very well the data. This might be due to several reasons. For instance, the Net might be too simple, or there might be some outliers/noise in the data.

- ## Scatter plots for the training data and the test data

As R squared already suggested, the fitting of the data is not very good. It seems the model does not capture some important pattern of the higher values of the data (ie: it is underfitting the data).
Probably the net is not complex enough to capture all the patterns of the data. Adding another hidden layer could be a good idea.

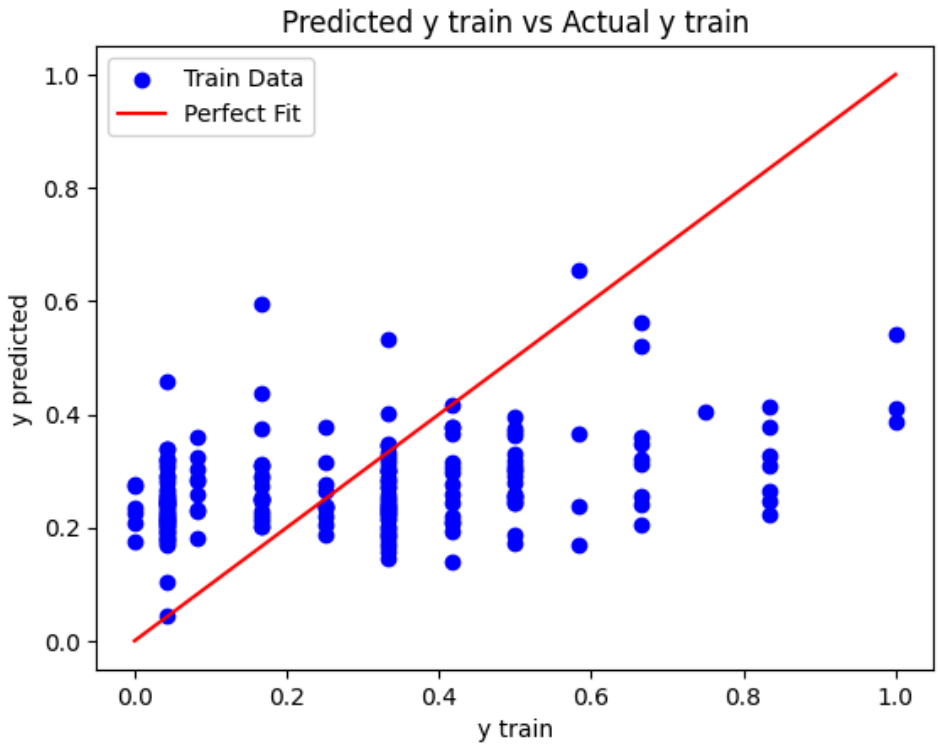

Predicted y train vs Actual y train
Train Data
Perfect Fit
y predicted
y train
0.0
0.2
0.4
0.6
0.8
1.0

Fig 13

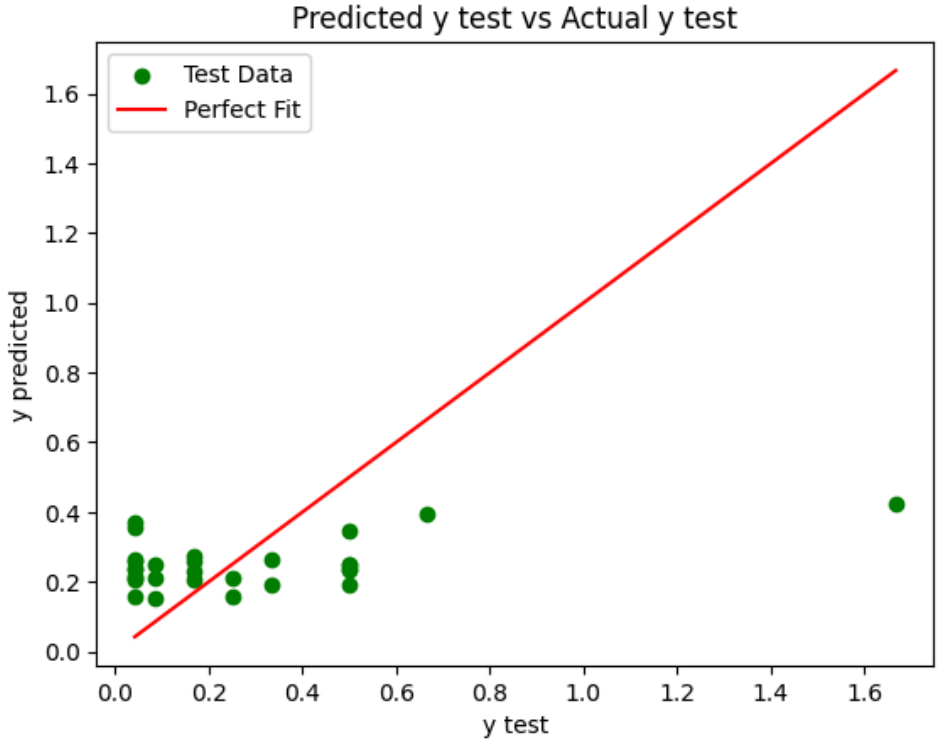

Predicted y test vs Actual y test
Test Data
Perfect Fit
y predicted
y test
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

Fig 14

Task 3: Classification using MLFFNN

Dataset 3

Exploratory analysis

We plotted the data to get a rough idea of the data. The target variable is dichotomous {0,1}.

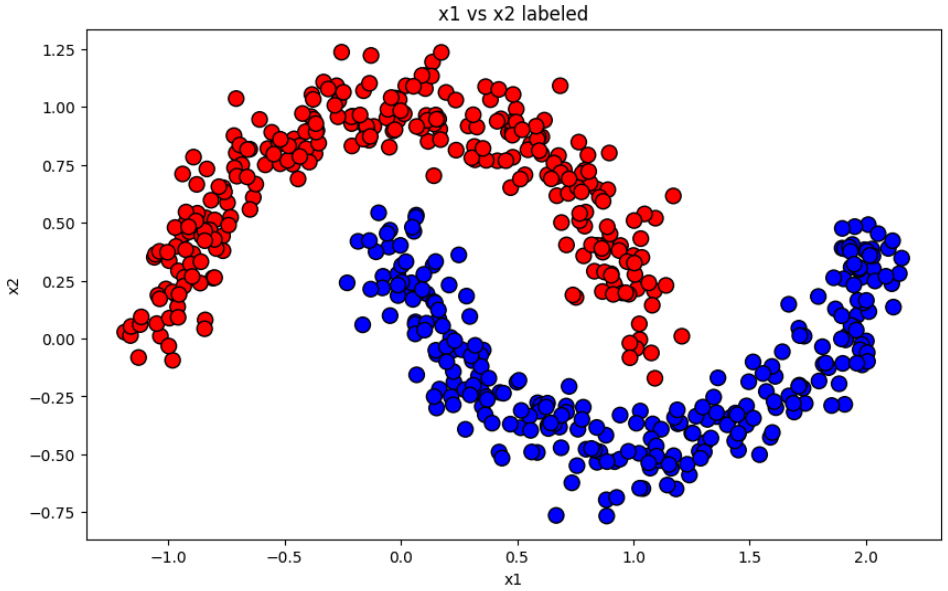
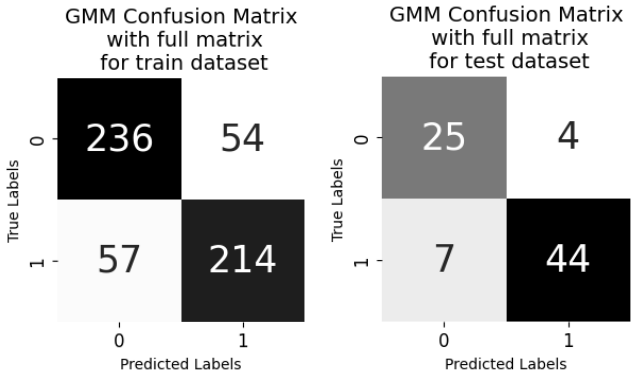


Fig 15

GMM Classifier

- **Confusion matrices**



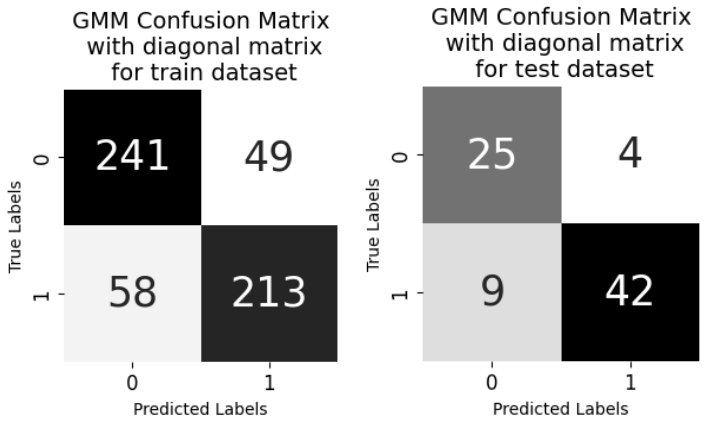


Fig 16

- **classification accuracies**

Classification accuracy for GMM with full matrix			Classification accuracy for GMM with diagonal matrix		
Training	Test	Validation	Training	Test	Validation
80.21%	86.25%	77.99%	80.93%	83.75%	78.62%

Table 3

For two dimensional, non-linearly separable data the result of GMM classification for both cases (with full and diagonal matrices) is similarly good either for training and test/validation data.

- **Decision region plots -> Superpose the training data on the decision region plot. Superpose the plots of level curves on the training data.**

On plots below Decision Boundaries for GMM are shown (both for full and diagonal matrices). Dots represent training data and cross - testing data.

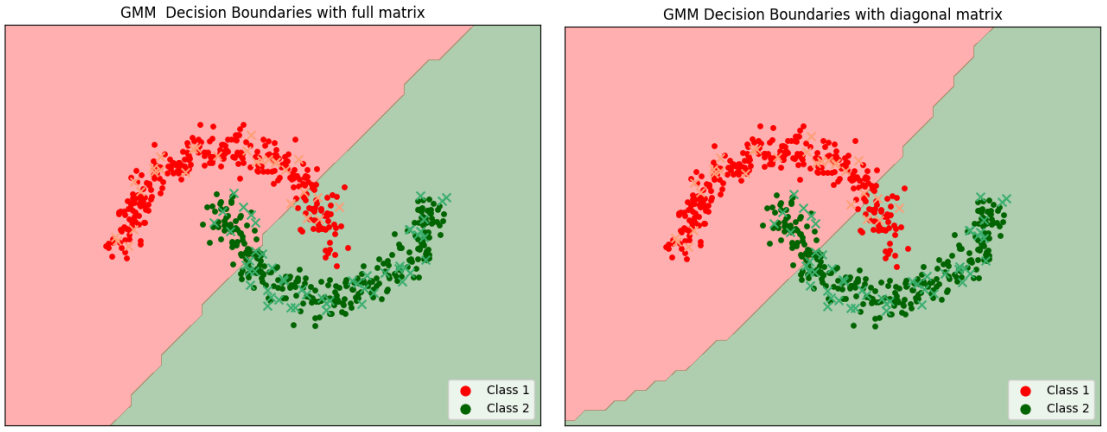


Fig 17

On the decision boundary plot we see that despite relatively high accuracy, we see that separation of classes could be better. In this case MLFFNN classifier performs better (see next point).

MLFFNN based Classifier

The Neural network designed is the following:

Layer (type)	**Output Shape**	**Param #**
Linear-1	[-1, 1, 12]	36
Tanh-2	[-1, 1, 12]	0
Linear-3	[-1, 1, 8]	104
Tanh-4	[-1, 1, 8]	0
Linear-5	[-1, 1, 2]	18
Total parameters: 158		

Table 4

NB: we defined the output layer (Linear-5) as just a linear transformation because PyTorch cross entropy loss automatically applies Softmax transformation on the output data.

- ## Confusion matrices and classification accuracies.

Below we report the confusion matrices for both the training and test data. It can be seen that the fit is very good for both the training and test data (good balance between bias and variance). In particular the training data are classified correctly 100% of the time, while the test data 99.37% of the time.

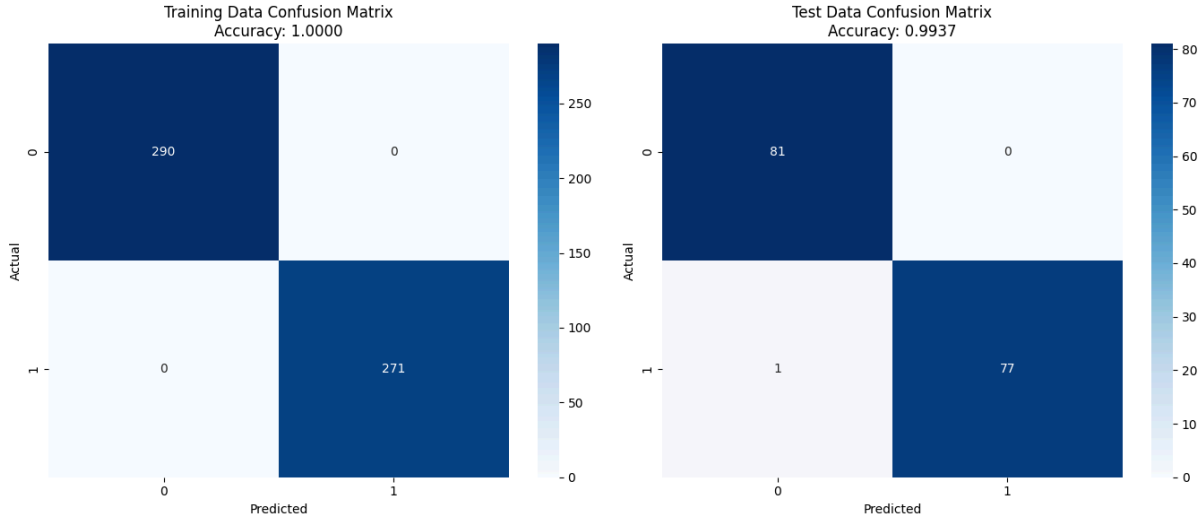


Fig 18

- ## Decision region plots

It is also interesting to plot the decision region plots of the Neural Network. From the Fig we can see that the decision region borders correctly follow the pattern of the data.

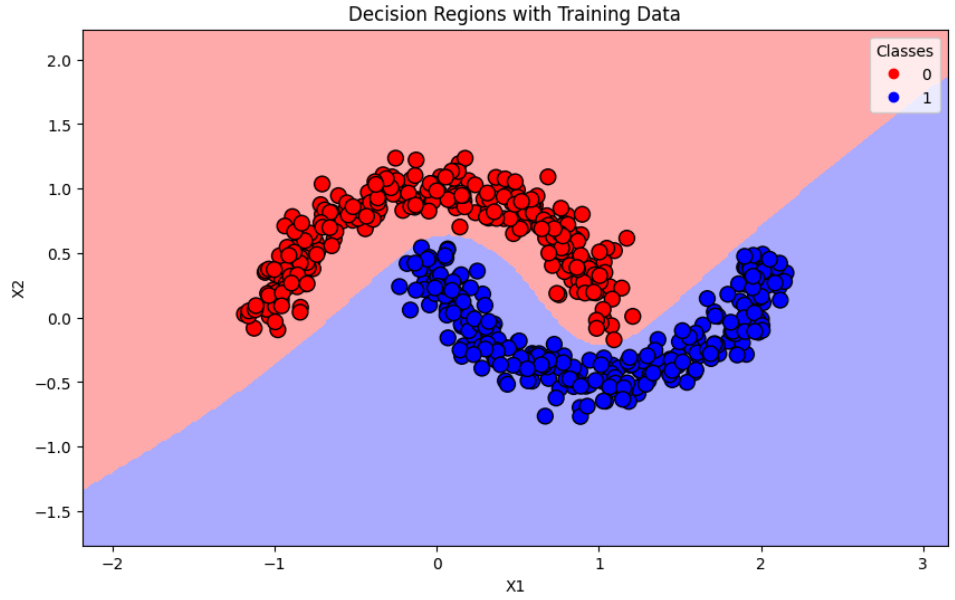


Fig 19

- ## 4 Surface plots (after Epochs 1, 10, 50, and convergence)
  - ### Epoch 1

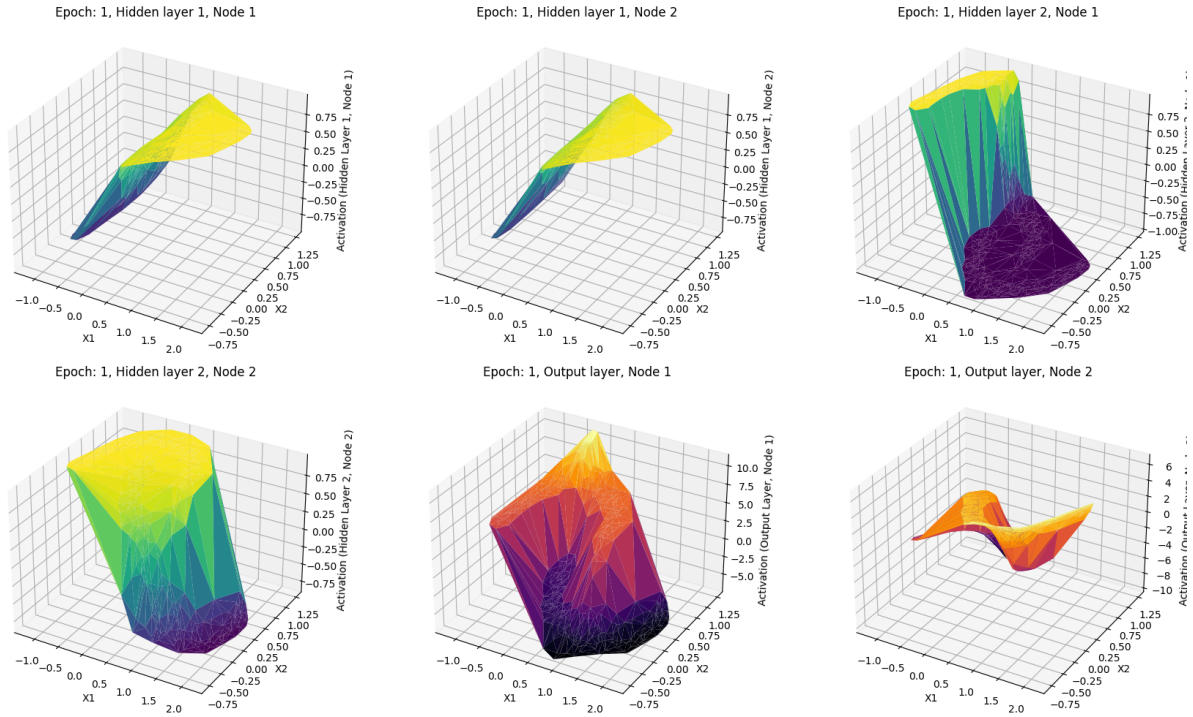


Fig 20

- **Epoch 10**

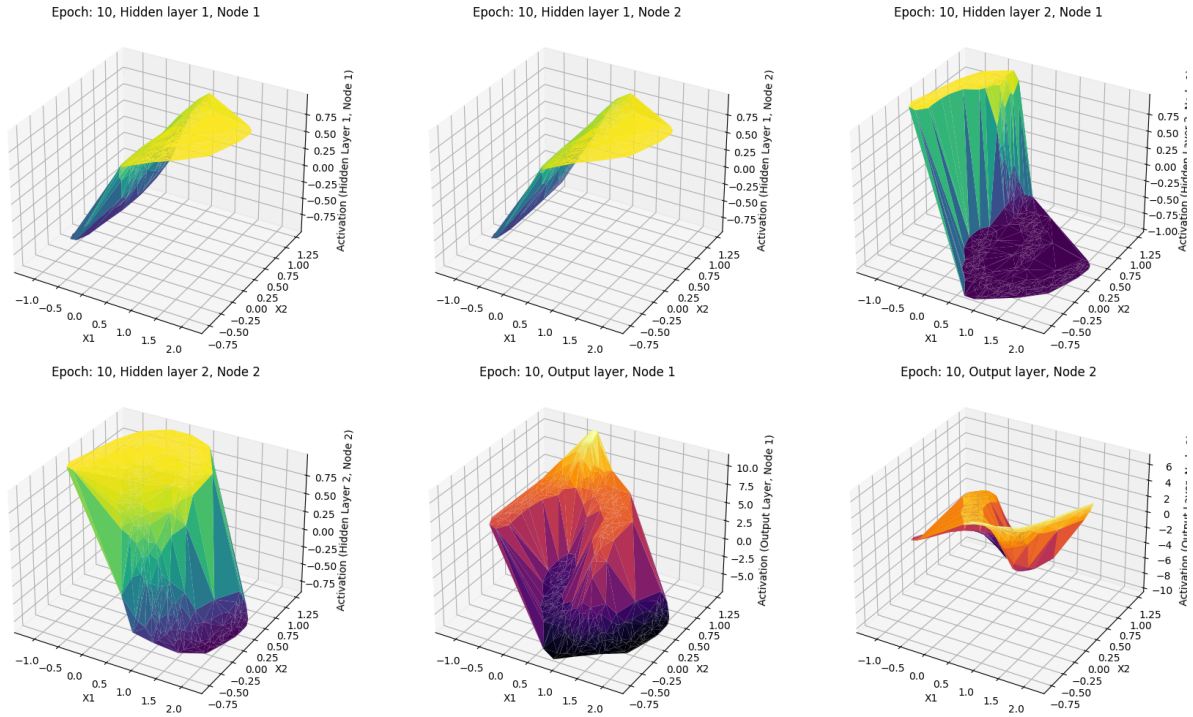


Fig 21

- **Epoch 50**

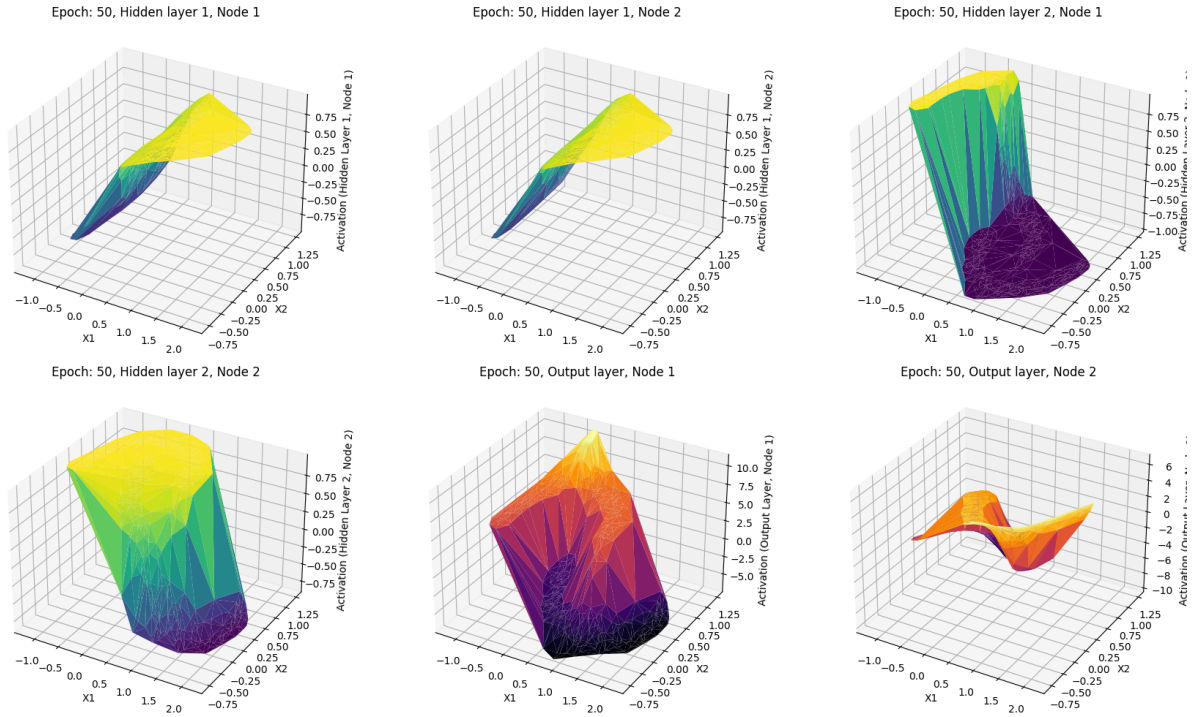


Fig 23

- Convergence

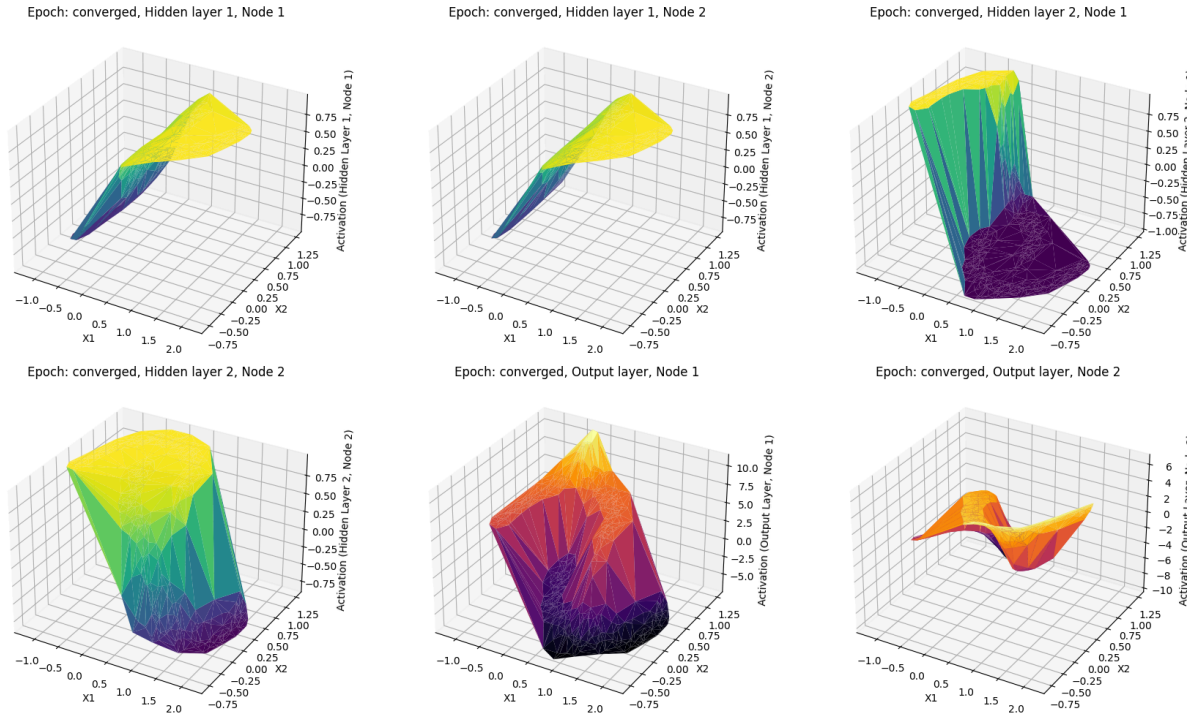


Fig 24

In all the plots above, it can be seen that the neuron's activations follow a pattern. It is difficult though to explain the activation of the nodes due to the level of abstraction in the hidden layers. Even for the output layers it is difficult to get an idea due to the complexity of the surfaces.

- Training error(ξav) vs epoch plot

Even with a quite high learning rate the training loss function has a smooth trend toward zero. Therefore the Neural Network is learning correctly.

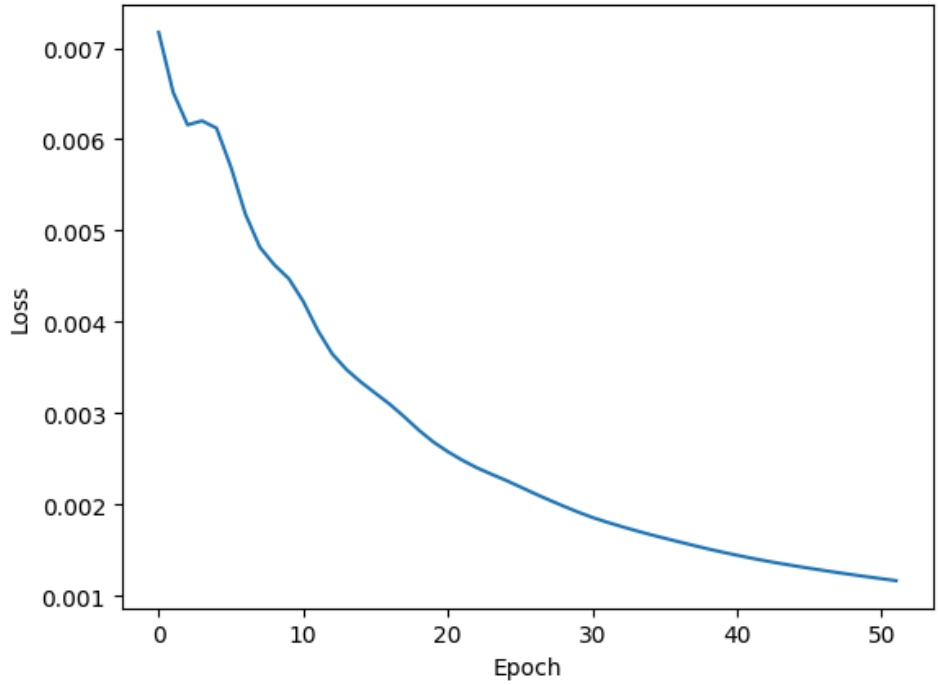


Fig 25

Task 4: Classification using MLFFNN

Dataset 4

GMM Classifier

- **Confusion matrices**

In the multidimensional case GMM classification accuracy was high for training data but the model performed much worse on test and validation data.

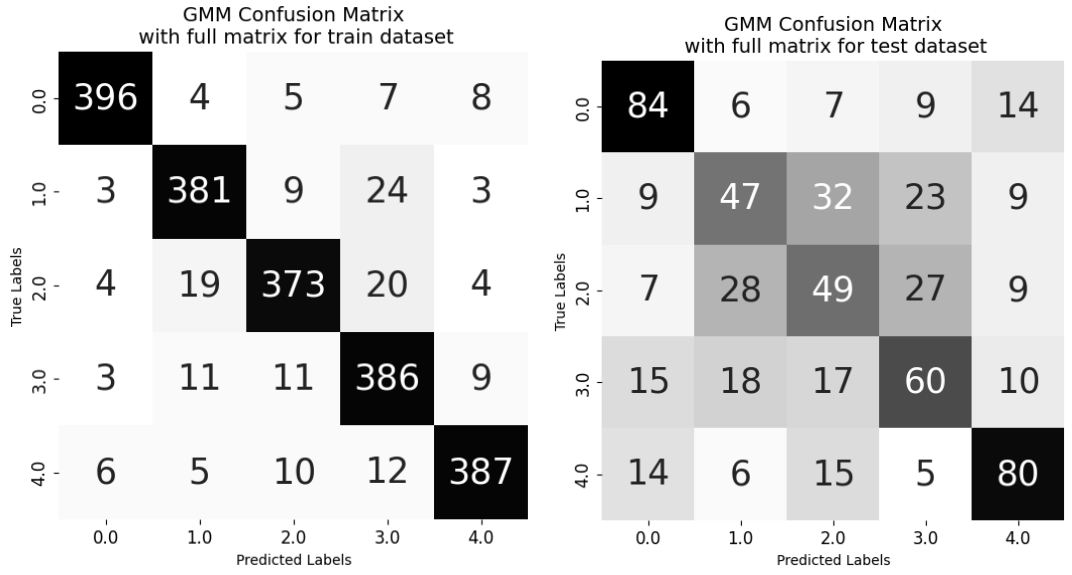


Fig 26

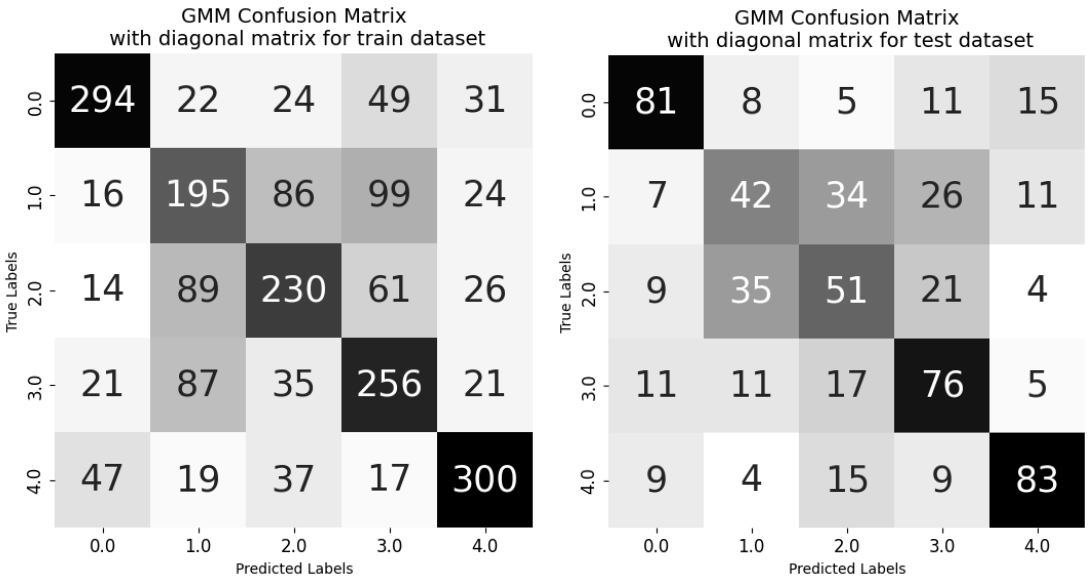


Fig 27

- **classification accuracies**

Classification accuracy for GMM with full matrix			Classification accuracy for GMM with diagonal matrix		
Training	Test	Validation	Training	Test	Validation
91.57%	53.33%	56.33%	60.71%	55.50%	62.67%

Table 5

MLFFNN based Classifier

The Neural network designed is the following:

Layer (type)	**Output Shape**	**Param #**
Linear-1	[-1, 1, 25]	925
Tanh-2	[-1, 1, 25]	0
Linear-3	[-1, 1, 15]	390
Tanh-4	[-1, 1, 15]	0
Linear-5	[-1, 1, 5]	80
Total parameters: 1395		

Table 6

- **Confusion matrices and classification accuracies**

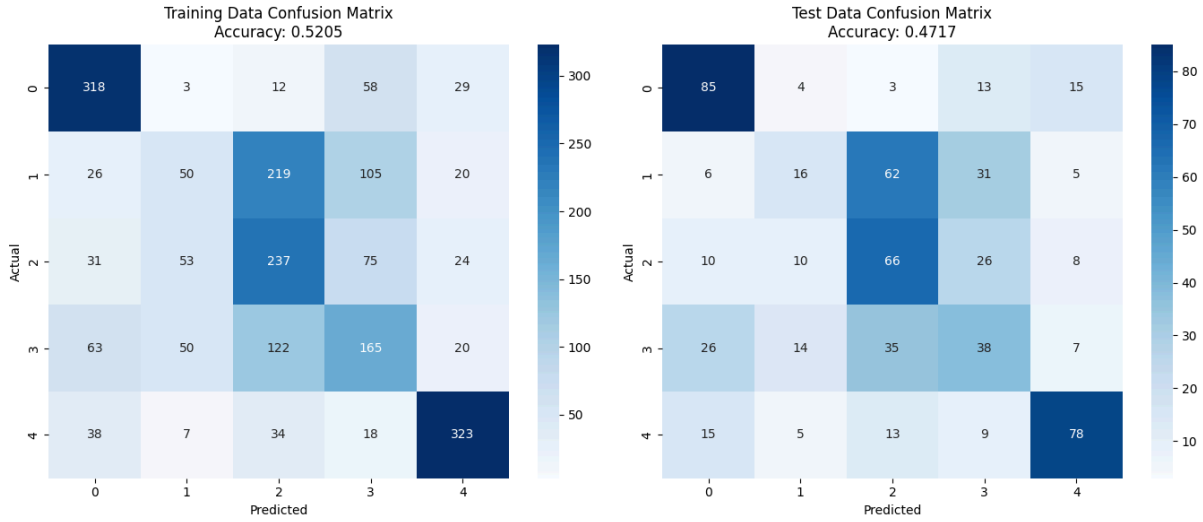


Fig 28

In higher dimensional data we can observe that for a proposed neural network, predictions are not as good as predictions obtained with Gaussian Mixture Model (see plot above). Although with increase of number of epochs we observe decrease of training error (see plot below).
It seems the Neural Network has some difficulties in explaining the data. By plotting the data we can see that many times the data overlaps. In Figure 29 it can be seen the second and third dimension of x labeled.

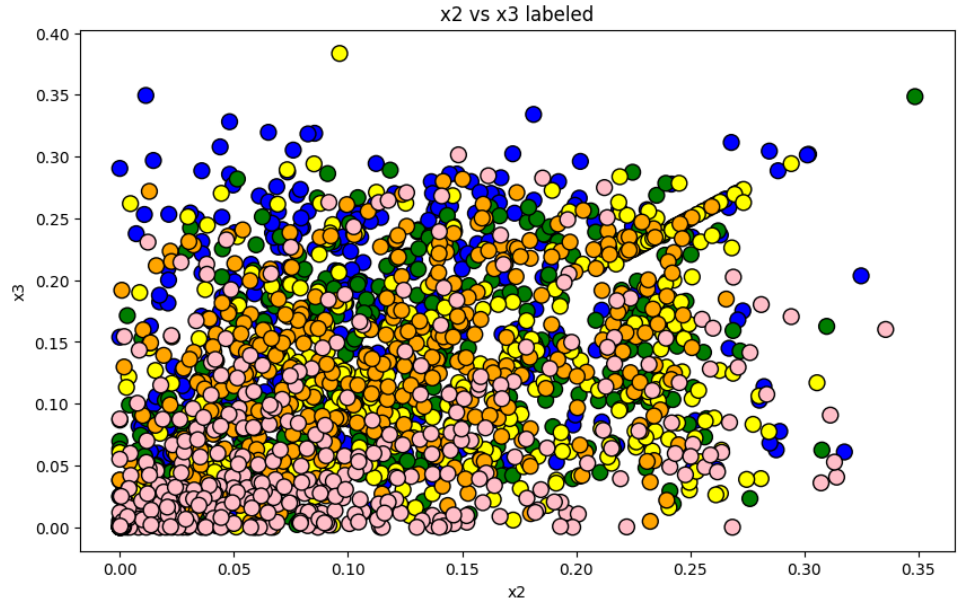


Fig 29

- **Training error(ξav) vs epoch plot**

In any case, from the loss function representation (Fig 30) we can see that the Neural Network is actually learning some patterns of the data.

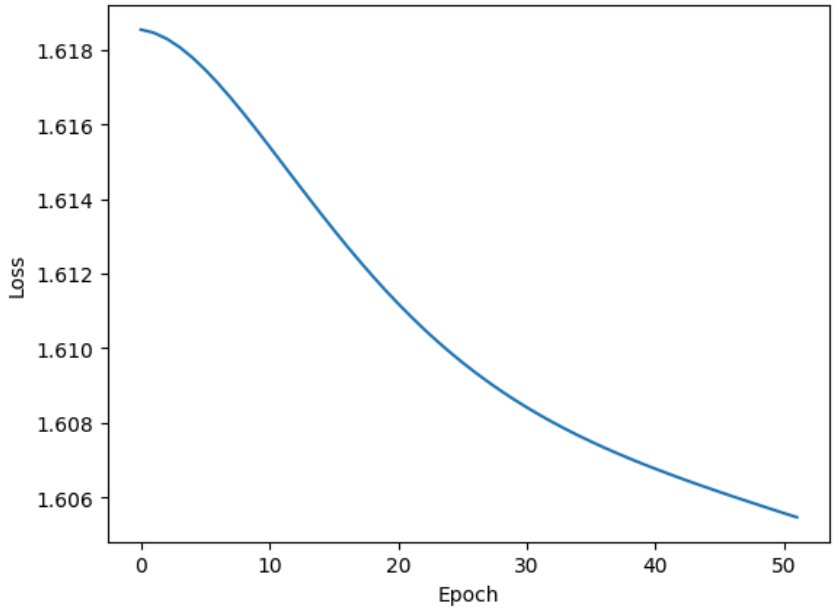


Fig 30