WARSAW UNIVERSITY OF TECHNOLOGY

FACULTY OF MATHEMATICS AND
INFORMATION SCIENCE

FIRST SEMESTER 2021/2022

NEURAL NETWORK

PROJECT 1

BY

Swetha Suresh Kumar (324353)

Andra Umoru (324334)

SUBMITTED TO:

Prof. Jacek Mańdziuk

Table of Contents

List of Figures

1. Problem description

This project is an implementation of Multilayered Perceptron trained with Backpropagation and tested on two data sets from the UCI Machine Learning repository. Also, this project is a low-level implementation of the neural network architecture and the backpropagation method.

1.1 Parameters

Parameters to be considered in this project are listed below; these parameters are to be taken as input at runtime are:

1. Number of hidden layers and number of neurons in hidden layers
2. Activation Function
3. Batch size
4. Number of epochs
5. Learning rate and
6. Momentum

1.2 Elements to analyse

The following are the elements to analyse at the end of the project:

- How does the Activation function selection affect the model's accuracy?
- How does the number of hidden layers and size of this layers affect the model's accuracy?
- The application should plot the training and test error.

2. Theoretical Introduction

This project encompasses some neural networks concepts. They are: perceptrons, multi-layered perceptron, feedforward, backpropagation, and dataset.

2.1 Perceptrons

A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and process elements in the training set one at a time. It is made of inputs X1,X2,X3, …, Xm their corresponding weights w1,w2,w3,..A function known as

activation function takes these inputs, multiplies them with their corresponding weights and produces an output y.

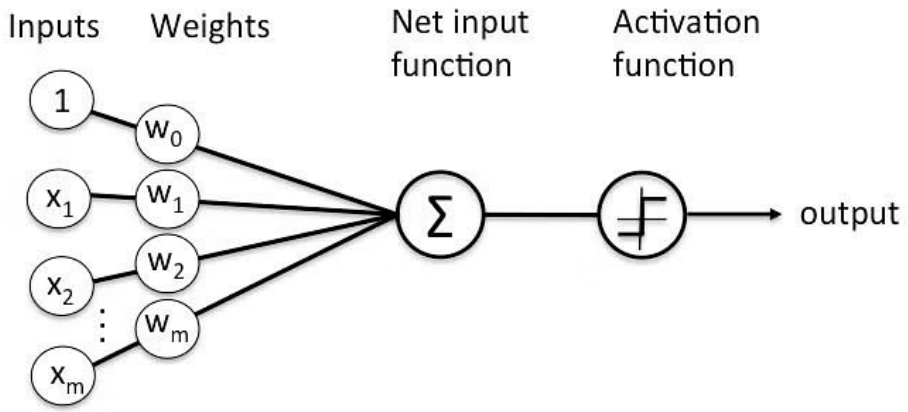

Figure 1: *Perceptron*

2.2 Multilayer Perceptrons

A multi-layered perceptron consists of interconnected neurons transferring information to each other, much like the human brain. Also, a multilayer perceptron (MLP) can also be seen as a feedforward artificial neural network that generates a set of outputs from a set of inputs. It is made up of at least 3 nodes. The nodes of the multilayer perceptron are arranged in layers.

- The input layers
- Hidden layers
- The output layers

2.3 Feed Forward

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. It is used to learn the relationship between independent variables, which serve as inputs to the network, and dependent variables that are designated as outputs of the network.

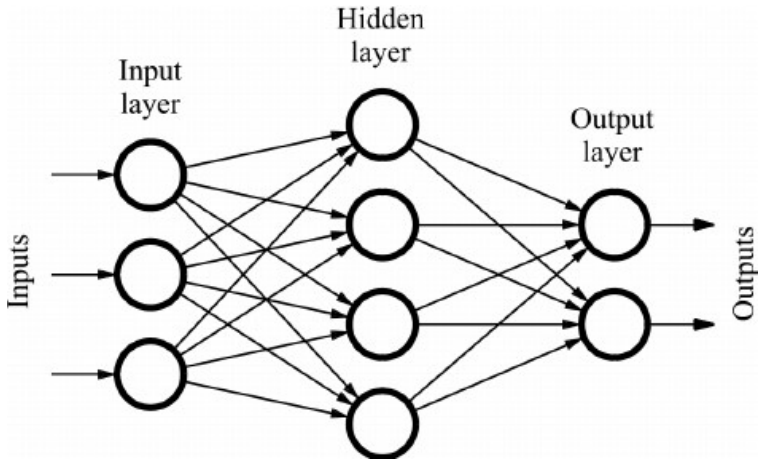

Figure 2: *Feed Forward*

In this model, a series of inputs enter the layer and are multiplied by the weights. Each value is then added together to get a sum of the weighted input values. If the sum of the values is above a specific threshold, usually set at zero, the value produced is often 1, whereas if the sum falls below the threshold, the output value is -1.

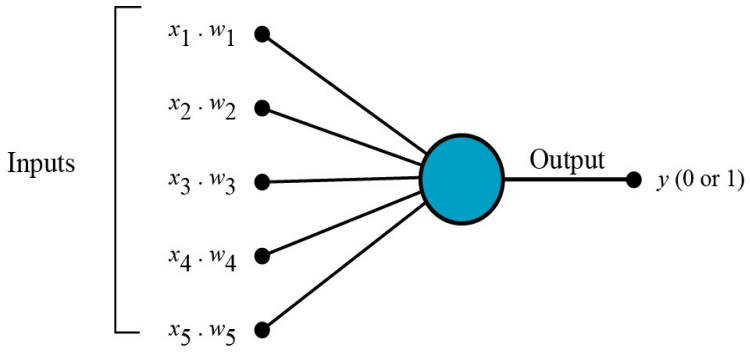

Figure 3: *Feedforward 2*

2.4 Backpropagation

The Backpropagation neural network is a multilayered, feedforward neural network and is by far the most extensively used. It is also considered one of the simplest and most general methods used for supervised training of multilayered neural networks. Backpropagation works by approximating the non-linear relationship between the input and the output by adjusting the weight values internally. It can further be generalized for the input that is not included in the training patterns. Generally, the

Backpropagation network has two stages, training and testing. The figure below shows a typical neural network backpropagation.

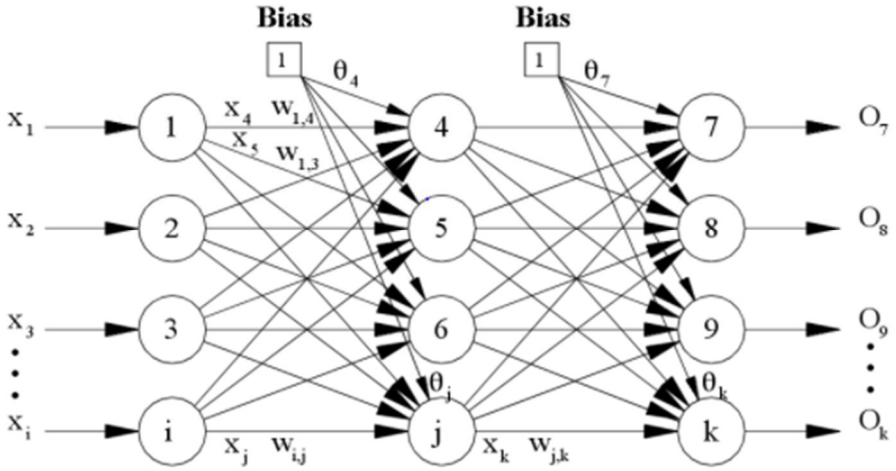

Figure 4: *Backpropagation Neural Network with one hidden layer*

2.5 Activation Functions

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model.

2.5.1 Activation Functions used for our implementation

1. Sigmoid:

The Sigmoid Function curve looks like a S-shape. The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

The function is differentiable. That means, we can find the slope of the sigmoid curve at any two points.

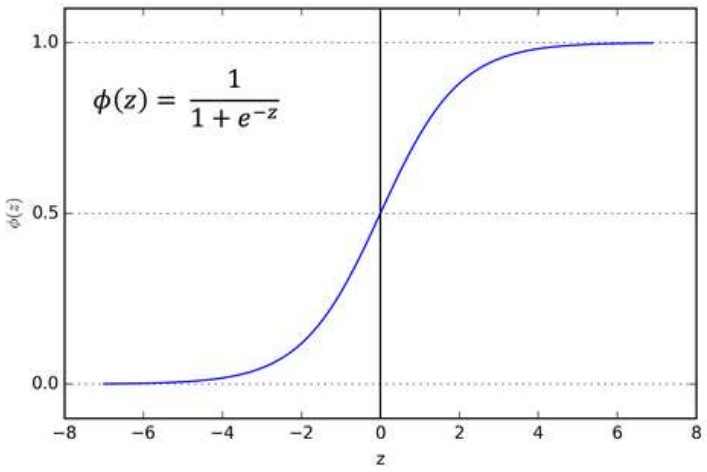

Figure 5: *Sigmoid Function*

2. Hyperbolic tangent

The hyperbolic tangent activation function is also referred to simply as the Tanh (also "tanh" and "TanH") function. It is very similar to the sigmoid activation function and even has the same S-shape. The function takes any real value as input and outputs values in the range -1 to 1

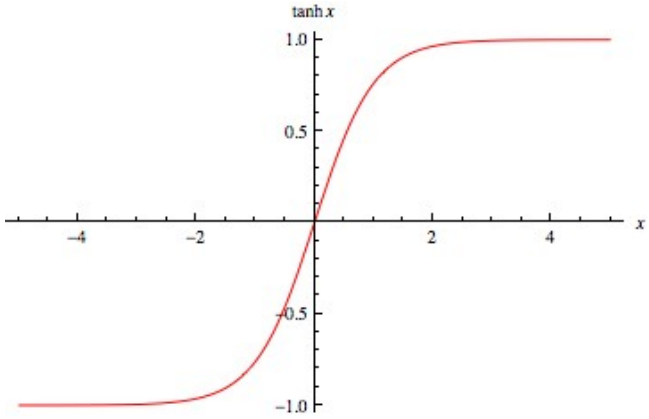

Figure 6: *Hyperbolic tangent function*

3. 3. Dataset Description

3.1 Dataset 1: Image Recognition Task Execution Times in Mobile Edge Computing

This dataset contains the turnaround execution times (in seconds) for offloaded image recognition tasks when executed in different edge servers. The edge servers are MacBook Pro Processor: 1.4 GHz Quad-Core Intel Core i5 RAM: 8 GB 2133 MHz LPDDR3. The turnaround execution time is the time duration once the connection is established (when the edge node starts sending the image) until it receives the recognition result from the edge server. The execution time is recorded for each edge server. There are 1000 set of data in this dataset. This dataset is of Univariate, Sequential, Time-Series data type from its original state as downloaded from UCI machine learning data repository.

Attribute Information:

[1] Time: in seconds.

[2] Turnaround Task Execution time: in seconds.

3.2 Dataset 2: RasberryPi

This is second dataset, we code named it “RasberryPi in order to differentiate it from the first dataset. There are 1000 set of data in this dataset also. It has the following information: the turnaround execution times (in seconds) for offloaded image recognition tasks when run in different edge servers are contained in this dataset. Raspberry Pi 4B are used as edge servers. Processor: ARM Cortex-A72 quad-core 64-bit CPU RAM: 4GB The turnaround execution time is the time it takes for the edge node to receive the recognition result from the edge server after the connection is established (when the edge node starts sending the image). Each edge server's execution time is logged. This dataset is of Univariate data type from its original state as downloaded from UCI machine learning data repository. The distinguishing feature between this dataset and the previous one is the one state nature of the datatype for this second dataset - univariate.

Attribute Information:

[1] Time: in seconds.

[2] Turnaround Task Execution time: in seconds.

Both of these datasets were downloaded from the UCI Machine Learning Data Repository.

4. Application Manual

This application was developed using python 3. It can be used adding the project folder to any Python IDE or Jupyter Notebook.

- By clicking the run all button (depending on the python application available to the user).
- The application will prompt the user to input the following: hidden layer, number of epochs, batch size, learning rate and the momentum.
- The user is then prompted to select the activation function i.e., using either
  1 = Sigmoid Function and
  2 = Hyperbolic Tangent function.
- After entering these: the application trains the data, test the data and print out the output (expected outcome, predicted outcome, mean error, train data set Loss value calculation, test error and some plots).

5. Experimental Setup

The application was implemented using python 3. The libraries used are:

1. NumPy

   This is a python library used for working with arrays, it also has functions for working in domain of linear algebra, Fourier transform and matrices.

2. **Pandas**

   This is also another python library used for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series.

3. Matplotlib.pyplot

   Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. In summary, it is used in plotting graphs in python programming.

4. **Seaborn**

   Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

In summary, these libraries are prerequisite for running the application. Since this application uses graphs and visualizations, the running platform should be able to handle these. Jupyter notebooks are highly recommended.

5.1Data Training, Testing and visualization

Different parameters that can be modified are:

- **Training data**:

  The first training dataset was stored in a CSV file called: **MacbookProo.csv** whereas for the second dataset is named: **RasberryPi.csv**. It reads any csv data and visualizes the dataset. The regression dataset is set to accept a csv file with 1 feature and 1 output. The application uses 70% of the dataset for training.

- **Batch size:**

  This is the number of samples processed before the model is updated. The size of a must be more than or equal to one and less than or equal to the number of samples in the training dataset. For this project, the batch size is dynamically entered by the user; for the output given in this report, a **batch size of 100** was selected to be used on both datasets.

- **Epoch:**

  The epoch is the number of complete passes through the training dataset. The epoch for this project is also entered by the user at runtime. The epochs selected for the result presented below is: **100.**

- **Learning Rate**:

  The learning rate controls how quickly the model is adapted to the problem. It is a positive value in the range between 0.0 and 1.0. For this report's output, a learning rate of **0.001** was selected.

- **Test data:**

  The test set is a set of data that is used to test the model after the model has already been trained. The test set is separate from both the training set and validation set. Since 70% of the dataset was for training, the remaining **30% was used for testing.**

- **Hidden Layer:**

  The hidden layer selected for the results presented in this report on both datasets used was 3.

- **Data cleaning and organization:**

  The input data obtained from both datasets used were in string datatype. In order to perform the needing regression analysis operation, we performed data cleaning operation in which we converted the date and the time given as string into its corresponding milliseconds which is of floating-point data type. By this, the data was ready for the regression analysis.

6. Experimental Results, Error analysis and discussions

This section will present the performance and results of the experiments conducted for the project.

6.1 Regression

6.1.1 Model Training

We train models with different data sets and find out Loss values and error values. The model trains 1000 times and calculates loss value. It calculates loss value up to 100 times and reaches a constant value.

```
Loss: Execution Time    0.004511
dtype: float64
Loss: Execution Time    0.003157
dtype: float64
Loss: Execution Time    0.002418
dtype: float64
Loss: Execution Time    0.001954
dtype: float64
Loss: Execution Time    0.001637
dtype: float64
Loss: Execution Time    0.001407
dtype: float64
Loss: Execution Time    0.001232
dtype: float64
Loss: Execution Time    0.001096
dtype: float64
Loss: Execution Time    0.000986
dtype: float64
Loss: Execution Time    0.000896
dtype: float64
Input:
```

Figure 7: *Regression train data set Loss value calculation*

6.1.2 Finding Mean Error

Find out error values, mean error, variance and standard deviation from the data sets with the help of NumPy functions and print out the output results

```
Mean Errorvalue  : -0.06528149914538478

Variance : 0.0001793692584085854

Standard Deviation : 0.013392880885328048
```

Figure 8: *Regression Mean error*

6.1.3 Input error graph

Find out error values, mean error, variance and standard deviation from the data sets with the help of NumPy functions and print out the output results

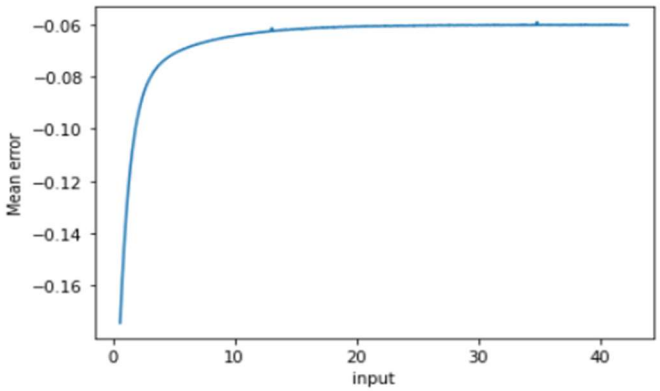

Figure 9: *Error graph with input Dataset 1*

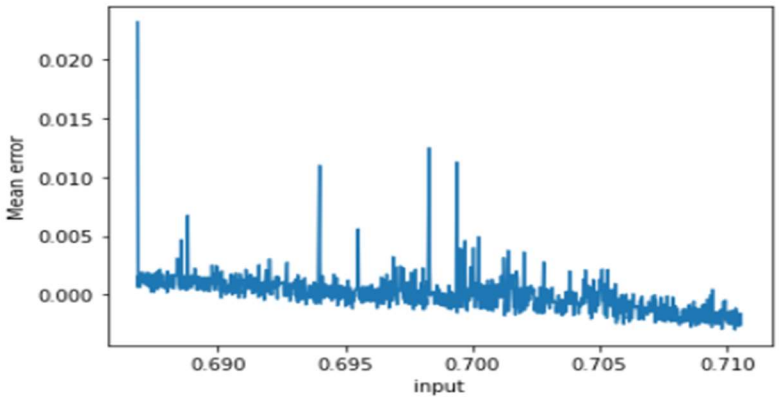

Figure 10: *Error graph with input dataset 2*

6.2 Regression

6.2.1 Data set visualization

In seaborn, two major functions are used to display a linear relationship obtained by regression. Regplot ( ) and lmplot ( ) are tightly correlated functions that share a lot of the same functionality.

However, it's critical to understand how they differ so that we can quickly select the right tool for the job. In the simplest invocation, both functions draw a scatter plot of two variables, x and y, and then fit the regression model y, x and plot the resulting regression line.

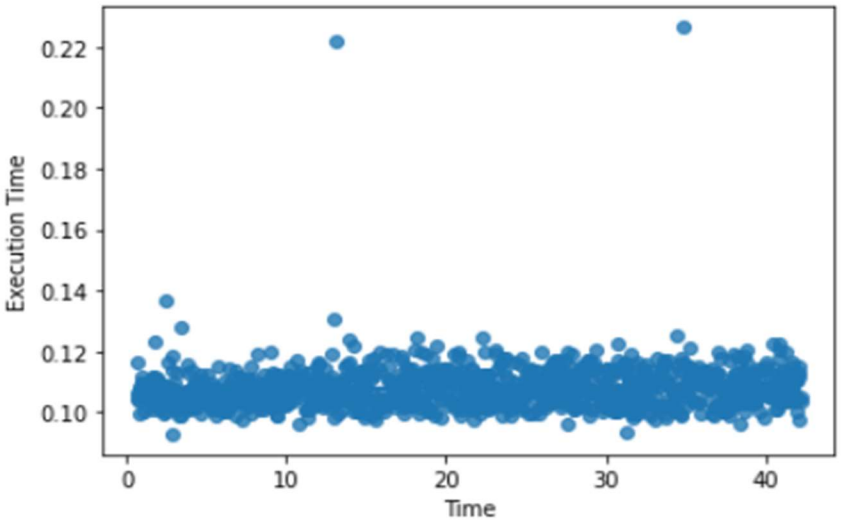

Figure 11: *Regression train data set Visualization for Dataset 1*

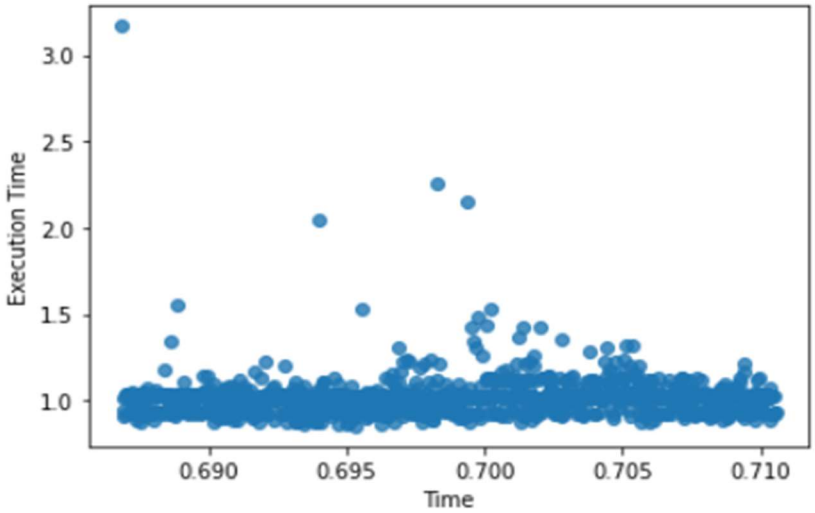

Figure 12: *Regression train data set Visualization for dataset 2*

6.2.2 Output prediction

Calculated prediction values from data models with mean loss value result. We got Loss value result with 0.004441043389077461 and prediction outputs shown in the figure given below; the predicted output displayed in an array

```
Input:
          Time
521  22.324745
737  31.324745
740  31.449745
660  28.116412
411  17.741412
..         ...
468  20.116412
935  39.574745
428  18.449745
7     0.908079
155   7.074745

[300 rows x 1 columns]
Actual Output:
     Execution Time
521         0.001243
737         0.000939
740         0.001076
660         0.001030
411         0.001123
..               ...
468         0.001079
935         0.001096
428         0.001117
7           0.001095
155         0.001067

[300 rows x 1 columns]
```

Figure 14: *Regression train dataset 1 values*

```
Input:
          Time
0     0.686840
1     0.686863
2     0.686887
3     0.686910
4     0.686933
..         ...
995   0.710463
996   0.710486
997   0.710509
998   0.710532
999   0.710556

[1000 rows x 1 columns]
Actual Output:
     Execution Time
0           0.031679
1           0.009125
2           0.010139
3           0.009442
4           0.009166
..               ...
995         0.009254
996         0.009168
997         0.009292
998         0.010243
999         0.009311

[1000 rows x 1 columns]
```

Figure 13: *Regression train dataset 2*

```
Mean Loss value : 0.004441043389077461


Predicted Output:
[[0.17527989]
 [0.17101437]
 [0.1669071 ]
 [0.1629556 ]
 [0.15915704]
 [0.1555082 ]
 [0.15200562]
 [0.14864555]
 [0.14542407]
 [0.1423371 ]
 [0.13938043]
 [0.13654977]
 [0.13384079]
```

Figure 16: *Regression train data set1 print section*

```
Mean Loss value : 2.5252139805765007e-06


Predicted Output:
[[0.0085015 ]
 [0.00850471]
 [0.00850793]
 [0.00851115]
 [0.00851436]
 [0.00851758]
 [0.00852079]
 [0.00852723]
 [0.00853044]
 [0.00853366]
 [0.00853688]
 [0.00854009]
 [0.00854331]
 [0.00854653]
```

Figure 15: *Regression train data set2 print section*

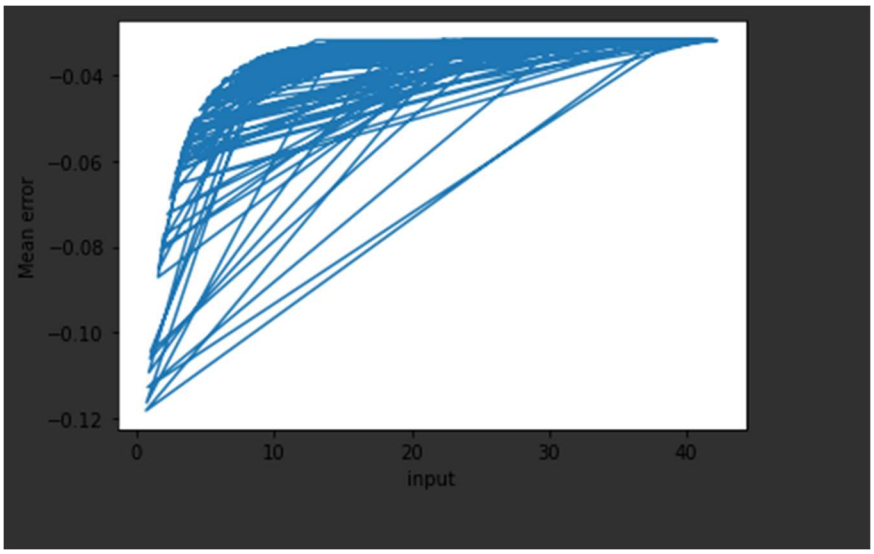

Figure 17: *Test - Graph Input value and Mean Error for dataset 1*

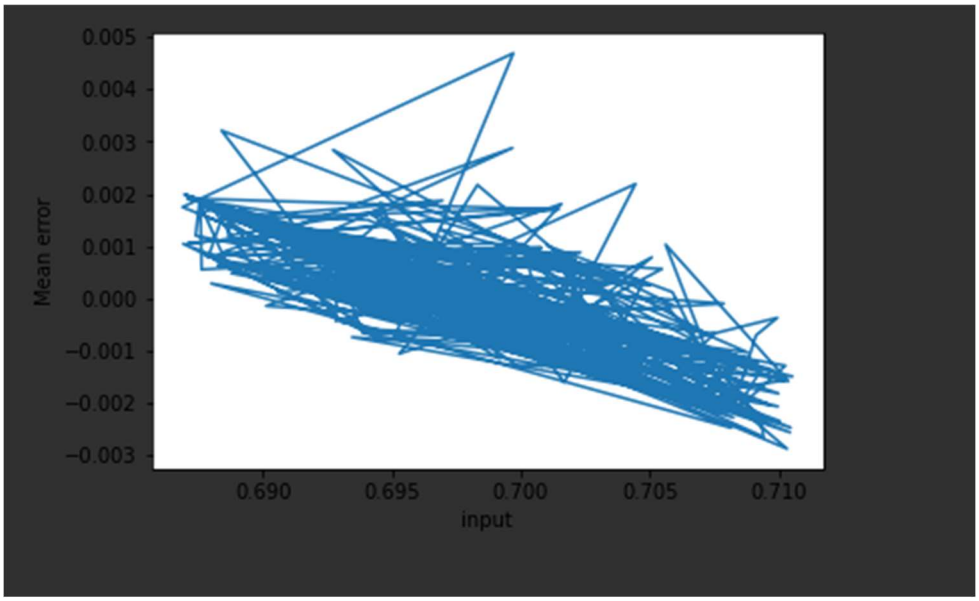

Figure 18: *Test - Graph Input value and Mean Error for dataset 2*

6.2.3 Result Summary

If we take a close look on the predicted output in figure 15 and 16, we will notice that: there was a mean loss value of 0.004441043389077461 (in figure 16 predicted using sigmoid function) as against 2.5252139805765007e-06 (in figure 15 predicted using hyperbolic tangent). This indicates that the error loss value is less in results obtained using the hyperbolic tangent as against the results obtained using the sigmoid function.

The following considerations were made:

i. The hyperbolic tangent function recorded less mean error when compared to that of the sigmoid function. This means that the choice of an activation has effects on the output of the model; in a nutshell, a good choice of activation function will help our model to predict accurately.

ii. The choice of the learning rate also contributes greatly to the accuracy of the results obtained. This is evidenced by about 90% prediction accuracy obtained from the model. Tweaking the learning rate, the number of the hidden layers and the momentum will offer us a great chance of obtaining a more accurate predictions that can be relied upon.

7. Conclusion

We have implemented the MLP with a backpropagation algorithm for regression. It is obvious that we could visualize the training progress and could experiment tweaking different hyper parameters. It can be concluded that, building multi-layered perceptron with backpropagation has the potential of given a 100% prediction when a good choice of the number of the hidden layer, learning rate, momentum, the number of epochs, batch size, and activation function is made.

7.1 Presumed reasons for Success/Failure

To a great extent, the implementation of this model is successful. This is evidenced by the predicted outcomes we obtained of about (90% prediction accuracy). This success rate is attributed to the following:

i. Modeling the activation functions (sigmoid and hyperbolic tangent). Our activation functions performed excellently well.

ii. The number of the hidden layers and the number of the neurons in each of them.

iii. Other factors that also contributed to this success rate includes the following: the number of epochs, batch size, and the learning rate.

In the course of implementation, we could not implement adding the momentum into the model. This was as a result of some errors encountered in the course of implementation.

8. Potential Future Research

We hope to develop a system in the future that can handle numerous loss and activation functions as application parameters. Furthermore, with more time in the future, we could add the momentum into the model. This we believe will contribute greatly to improving the percentage of the prediction accuracy.

9. Bibliography

i. Neural Network Lecture notes and class discussions

ii. Linear regression in python with Scikit-Learn – https://stackabuse.com/linear-regression-in-python-with-scikit-learn/

iii. Backpropagation - https://www.cse.unsw.edu.au/~cs9417ml/MLP2/BackPropagation.html#:~:text=Mutli%2DLayer%20Perceptron%20%2D%20Back%20Propagation&text=The%20Backpropagation%20neural%20network%20is,multilayered%20neural%20networks%5B6%5D.

iv. Multilayer Perceptron – Theory and Implementation of the Backpropagation Algorithm - https://pabloinsente.github.io/the-multilayer-perceptron

v. Linear Regression with Python and scikit-learn library - https://towardsdatascience.com/glr-with-python-and-scikit-learn-library-67b5b0d418ea

vi. www.wikipeadia.com

vii. www.geekforgeek.org