NEURAL NETWORK

Multi-layer Perceptron

BY Abba Umar

Hassan Jaber

1. Problem description

In this project, a Multilayered Perceptron is implemented and evaluated using two data sets from the UCI Machine Learning repository. It was trained using Backpropagation. Additionally, the neural network design and the backpropagation mechanism are being implemented at a low level in this project.

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
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?
- Dependency on the batch size.
- The impact of selecting of learning rate and momentum.
- The impact of weight updating methods.
- The application should plot the training and test error.

2. Theoretical Introduction

The project includes some neural network concepts. These include feedforward, backpropagation, datasets, multi-layered perceptron.

2.1 Perceptrons

A Perceptron is an algorithm for supervised learning of binary classifiers. Neurons can learn and process each component of the training set individually thanks to this approach. It consists of inputs X1, X2, X3,..., X m and their corresponding weights w1, w 2, w3, wm. These inputs are multiplied by the respective weights using a function called the activation function, which yields the output y.

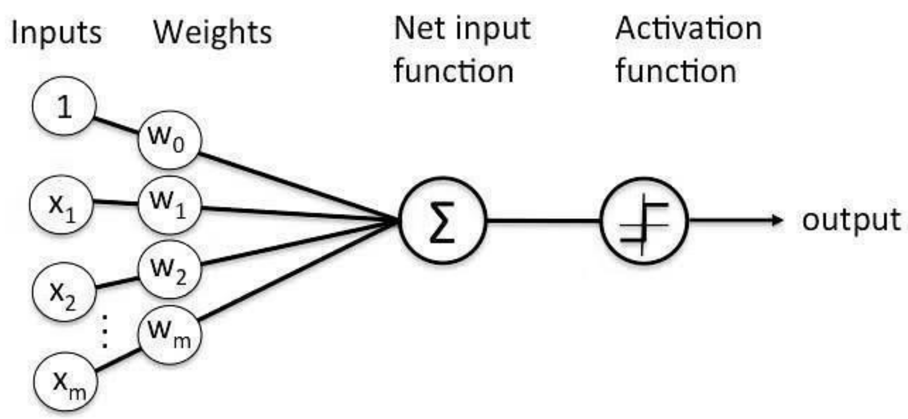


Figure 1: *Perceptron*

2.2 Multilayer Perceptrons

Similar to the human brain, a multi-layered perceptron is made up of linked neurons that communicate with one another. A multilayer perceptron (MLP), which creates a set of outputs from a collection of inputs, can alternatively be thought of as a feedforward artificial neural network. There are at least three nodes in it. The multilayer perceptron's nodes are organized in layers.

- The input layers
- Hidden layers
- The output layers

2.3 Feed Forward

Since input is only processed in one direction, the feed forward model is the simplest type of neural network. Although the data may flow via several buried nodes, it always proceeds forward and never backward. It is employed to discover the association between independent variables—used as the network's inputs—and dependent variables—used as the network's outputs.

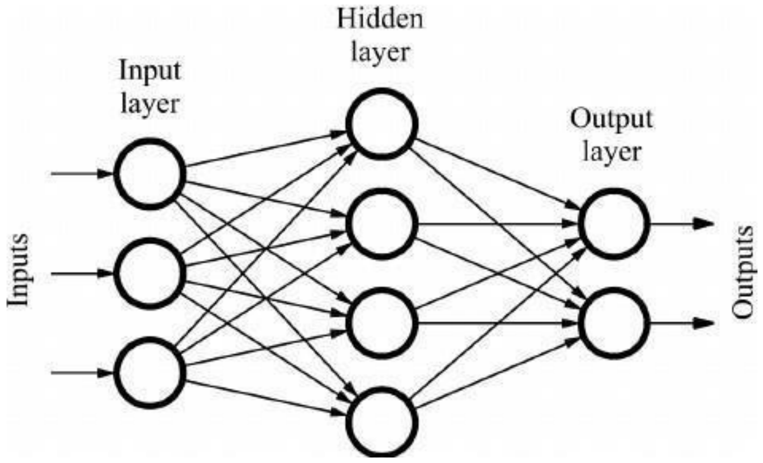


Figure 2: *Feed Forward*

A number of inputs are introduced into the layer in this model and multiplied by the weights. The weighted input values are then summed together to produce a total. The value produced is frequently 1, and if the sum of the values is below the threshold, the output value is -1. The threshold is typically set at zero.

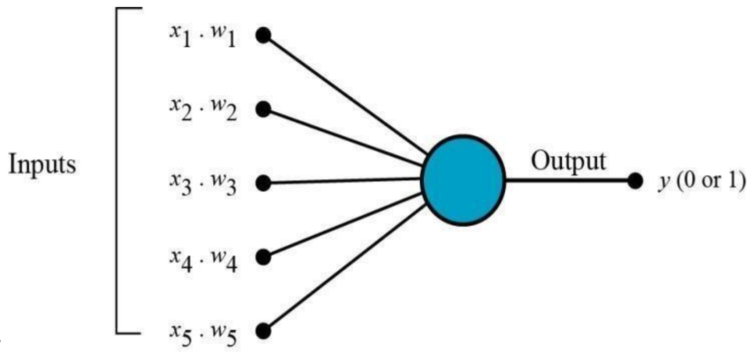


Figure 3: *Feedforward 2*

2.4 Backpropagation

The most often used neural network type is the backpropagation neural network, which has multiple layers and is feedforward. It is also regarded as one of the most straightforward and versatile techniques for supervising the training of multilayered neural networks. Backpropagation functions by internally modifying the weight values to approximate the non-linear relationship between the input and the output. For input not covered by the training patterns, it can be further generalized. The Backpropagation network typically goes through two stages: training and testing. A typical neural network backpropagation is shown in the figure below.

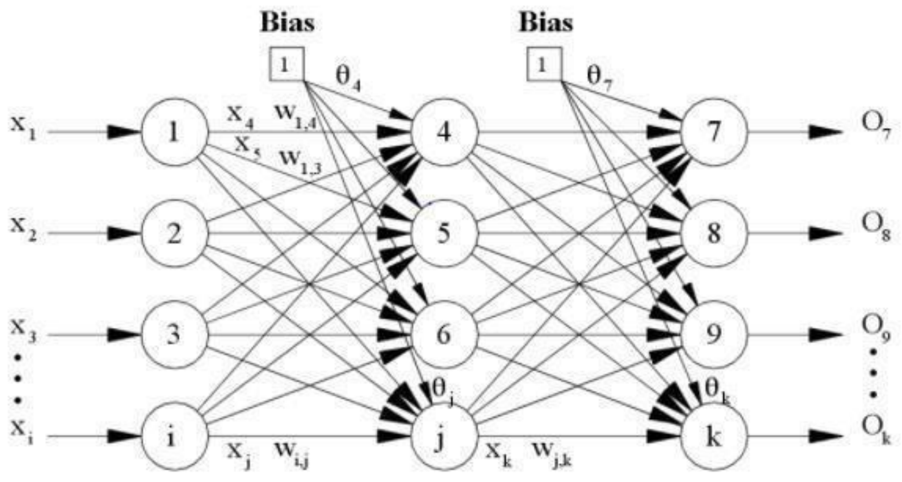


Figure 4: *Backpropagation Neural Network with one hidden layer*

2.5 Activation Functions

An activation function in a neural network describes how a node or nodes in a layer of the network translate the weighted sum of the input into an output. Different activation functions may be used in different regions of the model, and the choice of activation function has a significant impact on the neural network's capacity and performance.

2.5.1 Activation Functions used for our implementation

1. Sigmoid:

The Sigmoid Function curve has an S-shaped appearance. We chose the sigmoid function primarily because it can be found between 0 and 1. As a result, it is particularly used for models whose output is a probability prediction. The sigmoid is the best option because anything has a probability that only occurs between 0 and 1.

The function might take numerous forms. Therefore, we can determine the sigmoid curve's slope between any two points.

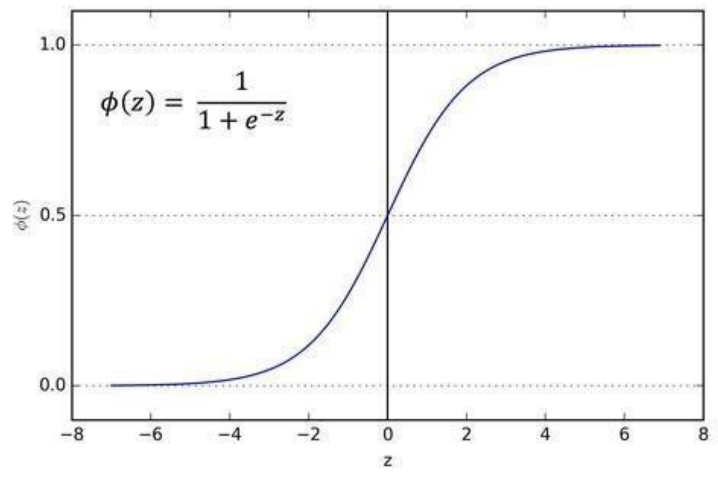


Figure 5: *Sigmoid Function*

2. Hyperbolic tangent

The Tanh function (also "tanh" and "TanH") is another name for the hyperbolic tangent activation function. It even shares the same S-shape with the sigmoid activation function.

The function outputs numbers between -1 and 1 and accepts any real value as input.

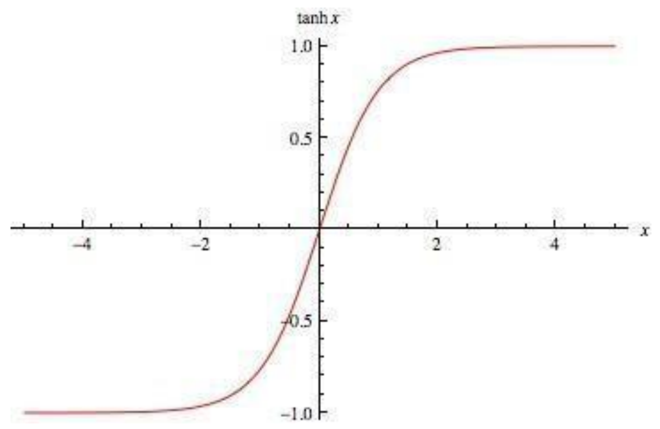


Figure 6: *Hyperbolic tangent function*

3. Dataset Description

3.1 Dataset 1: Iris

The Iris dataset was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems, and can also be found on the UCI Machine Learning Repository.

It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.

Attribute Information:

[1] SepalLengthCm

[2] SepalWidthCm

[3] PetalLengthCm

[4] PetalWidthCm

[5] Classes (Iris-Setosa, Iris –Versicolour, Iris-Virginica)

3.2 Dataset 2: Breast Cancer Wisconsin

This breast cancer database was obtained from the University of Wisconsin

Hospitals, Madison from Dr. William H. Wolberg. Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe the characteristics of the cell nuclei present in the image. Number of instances are 569 and there are 32 attributes

Attribute Information:

[1] Diagnosis (M: malignant, B: benign)

[2] 30 real-valued input features (for example: radius, texture, parameter, area, smoothness, compactness…)

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 neurons in each 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
  “sigmoid” = Sigmoid Function and
  “tanh” = Hyperbolic Tangent function.
- After entering these: the application trains the data, test the data, and predict.

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 base on matplotlib. It provides a highlevel interface of drawing attractive and informative statistical graphics.

5. **Sklearn**

   The Sklearn train_test_split function helps us create our training data and test data. This is because typically, the training data and test data come from the same original dataset.

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: **Iris.data** whereas for the second dataset is named: **wdbc.data**. It reads any data and visualizes the dataset. The classification dataset is set to accept any data file. The application uses 80% of the dataset for training.

- **Batch size:**

  This is the number of samples processed before the model is updated. The size of it 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 10** 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 epoch selected for the result presented below is: **1000.**

- **Learning_Rate**:

  The learning rate controls how quickly the model update the weight. It is a positive value in the range between 0.0 and 1.0. For this report's output, a learning rate of **0.05** was selected

- **Momentum**:

  The Momentum rate is used to increase the size of steps taken toward the minimum by trying to jump from a local minima. It is a positive value in the range between 0.0 and 1.0. For this report's output, a momentum rate of **0.5 and 1** 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 80% of the dataset was for training, the remaining **20% was used for testing.**

- **Hidden Layer:**

  The hidden layer selected for the results presented in this report on Iris dataset is 1, with 5 neurons in it and for the breast cancer dataset the number of hidden layer used is 2, with 15 and 7 neurons respectively.

- **Data cleaning and organization:**

  The input data obtained from both datasets used are integer datatype. In order to perform the needing classification, we performed data cleaning operation in which we converted the classes to binary. By this, the data was ready for the classification.

6. Experimental Results, Error analysis and discussions

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

6.1 Activation function selection:

We trained and tested the results using sigmoid and tanh activation functions, below are the result obtained for breast cancer:

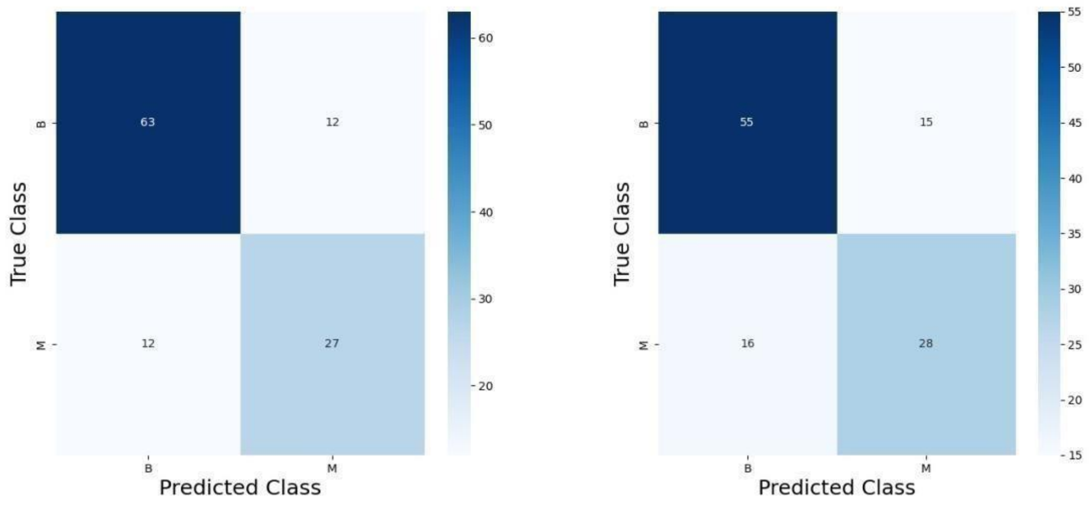
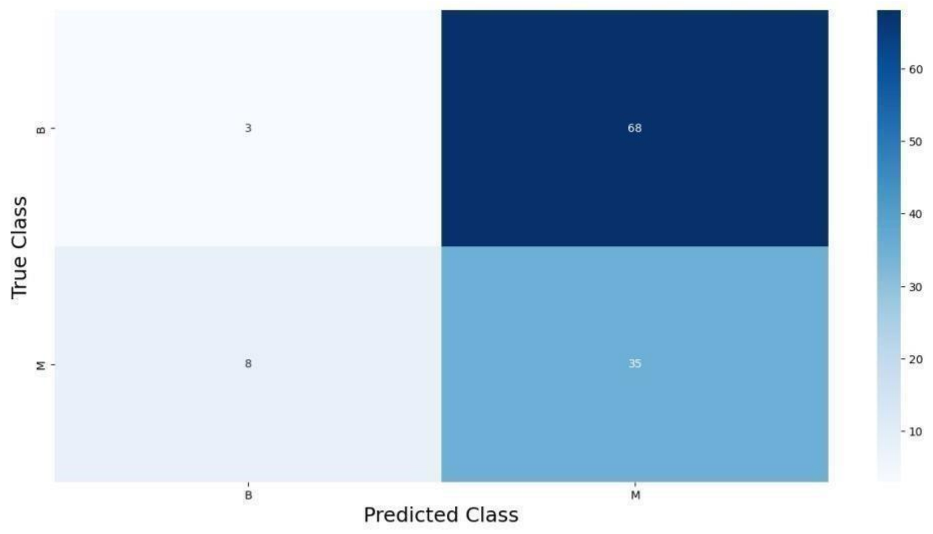


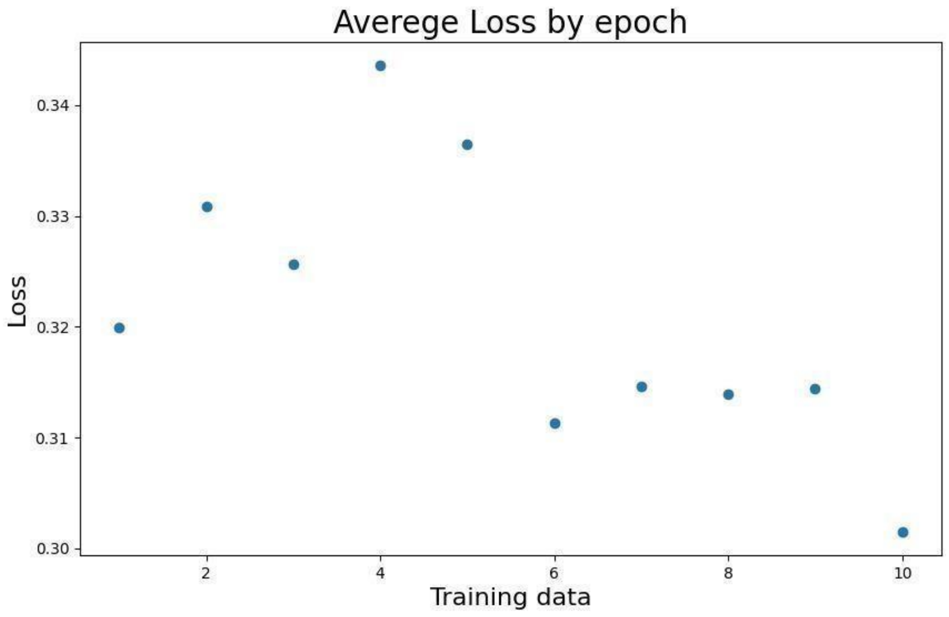
Figure 7: *The left confusion matrix represent the result after using sigmoid function, and the right after using tanh function.*

6.2 Number of hidden layers, and neurons in each of them:

We tested same number of hidden layers and different number of neurons using the breast cancer dataset, below are the results obtained:



True Class
Predicted Class
B
M
3
68
8
35
60
50
40
30
20
10



Averege Loss by epoch
Loss
Training data
0.34
0.33
0.32
0.31
0.30
2
4
6
8
10

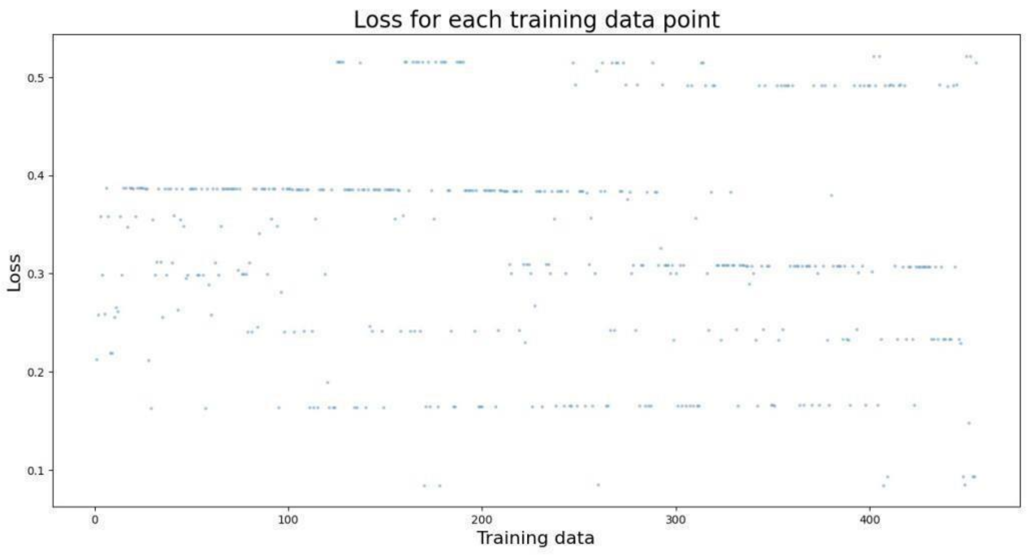
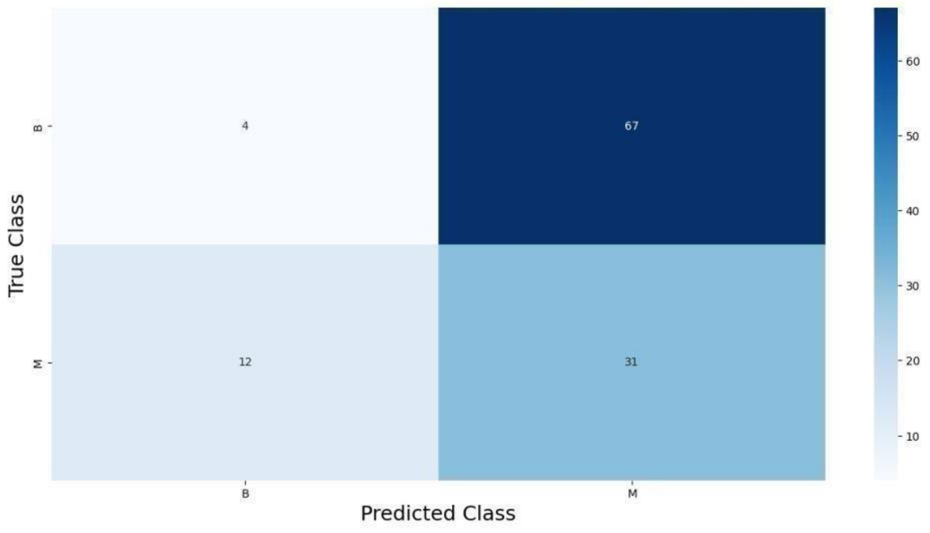
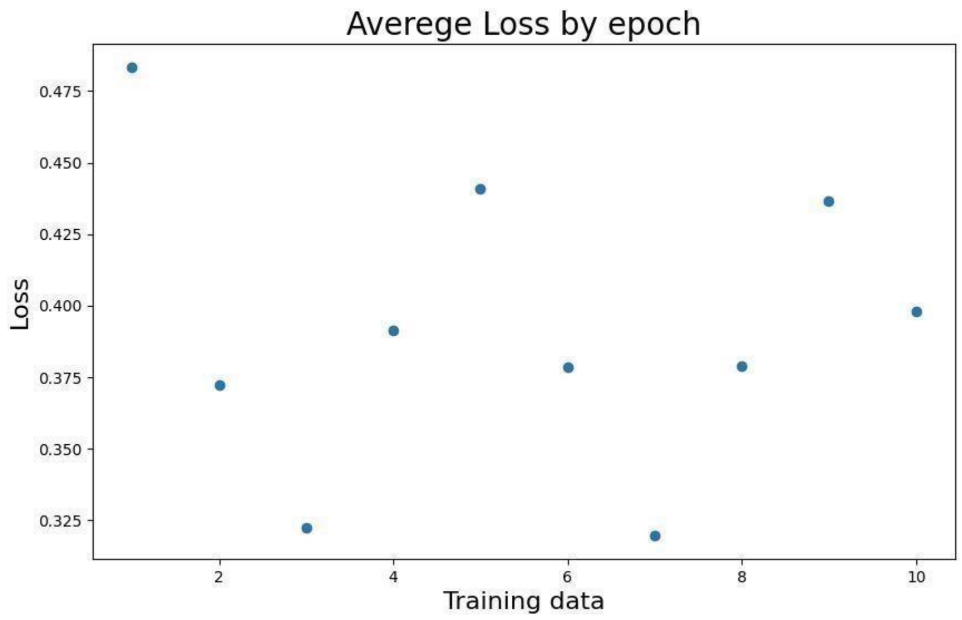


Figure 8: In the above figures, we can see the result after using 2 hidden layers, and 10 and 5 neurons respectively.





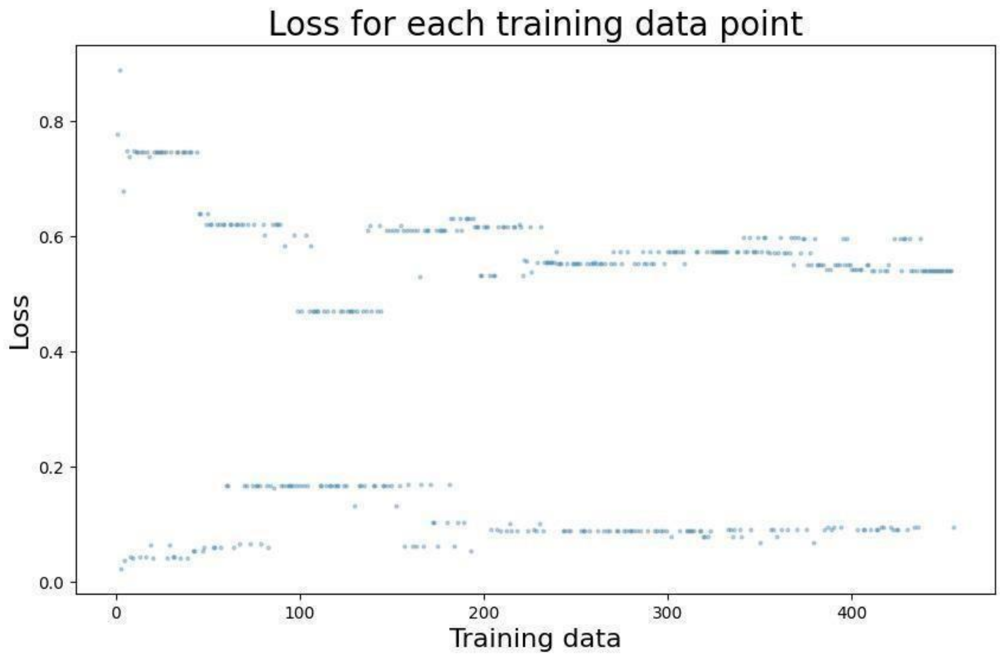


Figure 9: In the above figures, we can see the result after using 2 hidden layers, 15 and 7 neurons respectively.

6.3 Batch size dependency

We tested our program using different batch size to see how it can affect the results

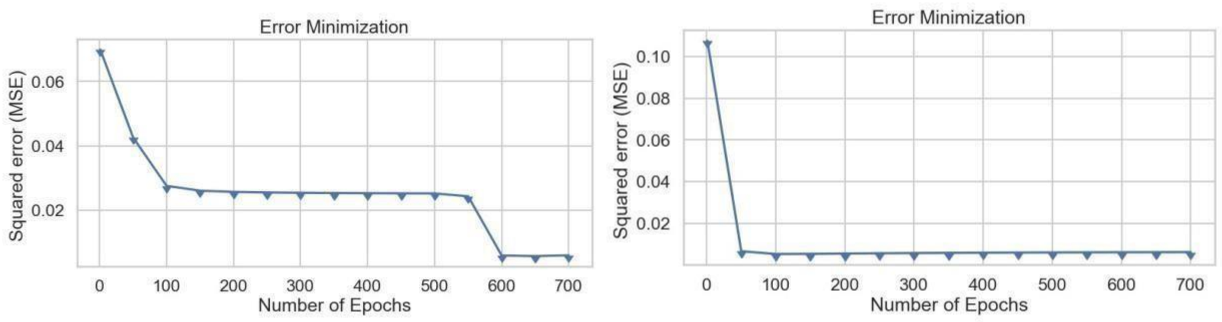


Figure 10*: In the above figures, we can see the error result after using batch size equal to 20(left), and batch size equal to 10(right).*

6.4 Learning rate and momentum impact:

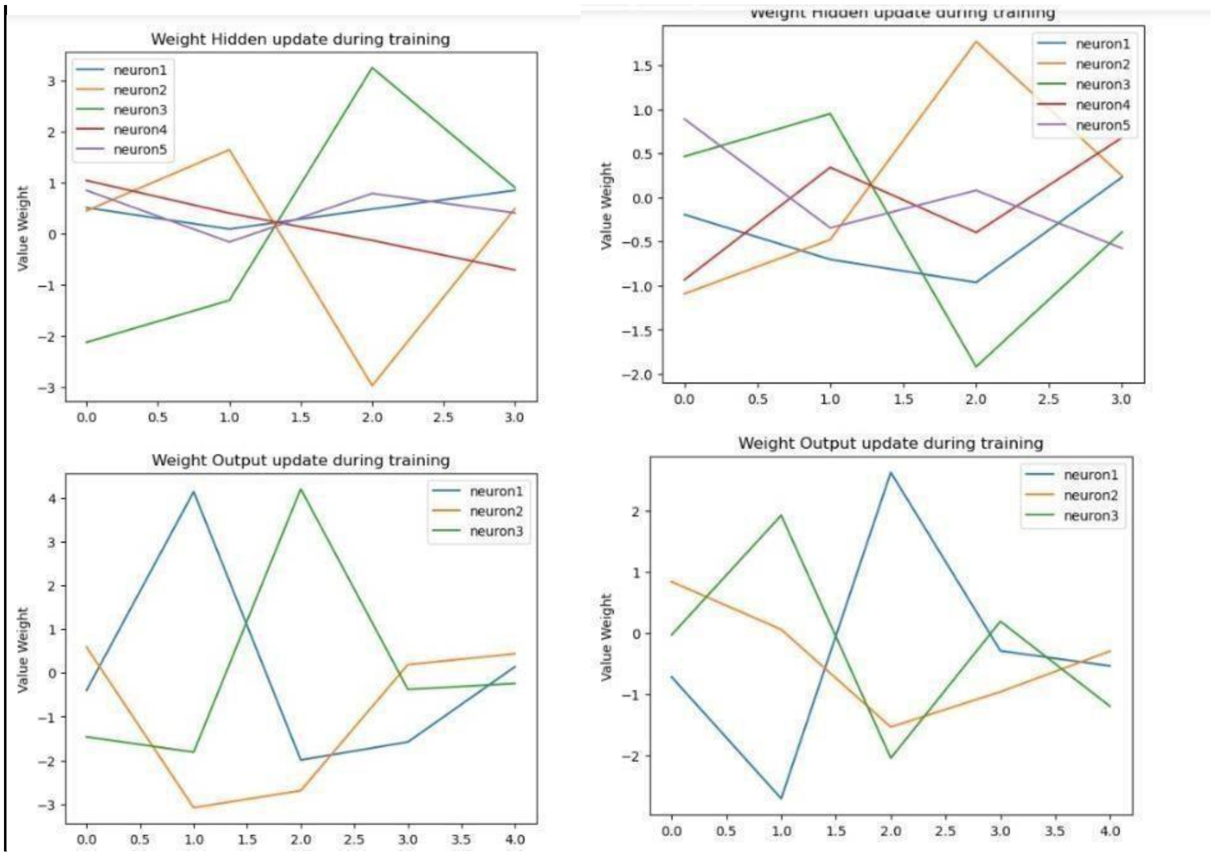


***Figure 8:** In the figure above we can see the result after using different momentum and learning rate.*

6.5 Weight updating methods impact:

We tested our program using 2 different weight updating methods: the first method is using Gaussian distribution, but the second one is symmetric distribution and reshaping the weights.

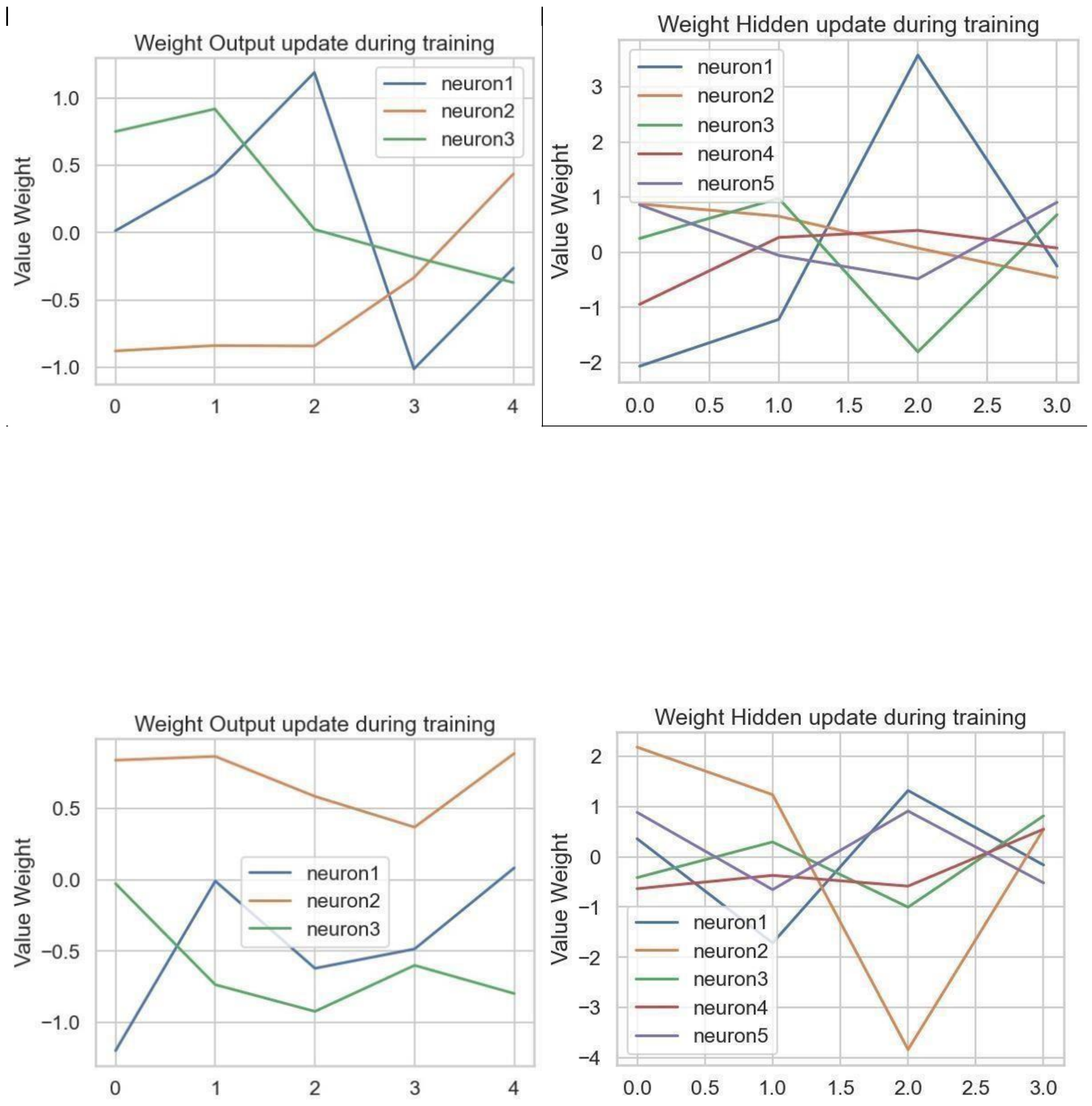


Figure 12: *The above 2 graphs represent result after using the First method, and the below ones are for the second method*

6.6 Training and test plot:

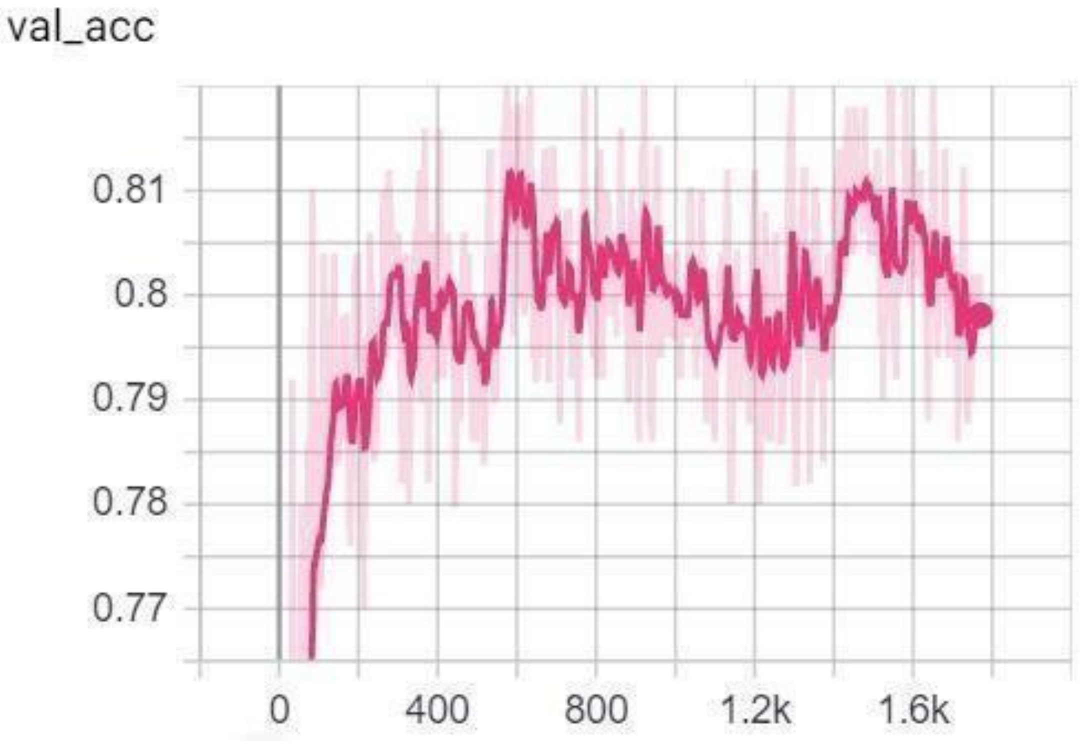


Figure 13: *Training and testing Plot*

6.2.3 Result Summary

i. If you noticed from the figure 7, we can see the sigmoid function performed better than the tanh function

ii. The number of hidden layers and number of neurons in each one, have impact on the both datasets, hidden layers more than 2 /3 of the input cause overfitting iii. The batch size has effect on both the training and testing, we obtained good result with batch size 10 rather than 20.

iv. After using the gaussian method we realize that the more layers we use, the harder it is for the weight to converge, but after using a modified one, we can put up to 30 hidden layers.

v. The lower learning rate and higher momentum, helped to increase the size of steps taking towards the minima.

vi. The plot shows we were able to attend 71% of accuracy on both data sets.

7. Conclusion

We have implemented the MLP with a backpropagation algorithm for classification. 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 (71% 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 include the batch size during training , but only during testing

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 batch size during training .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-inpythonwithscikit-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/glrwithpythonand-scikit-learn-library-67b5b0d418ea vi. www.wikipeadia.com vii. www.geekforgeek.org