Neural Networks: Exploring Multi-Layer Perceptron Architecture

Camilo Oberndorfer Mejía
Universidad EAFIT
Medellin, Colombia
coberndorm@eafit.edu.co

Olga Lucía Quintero Montoya
Universidad EAFIT
Medellin, Colombia
oquinte1@eafit.edu.co

***Abstract*—This paper presents a comprehensive analysis of the training of Multi-Layer Perceptron (MLP) neural networks, a feed-forward algorithm based on the working process of a brain under the perception of 1950s. The study showcases the MLP's utility in learning tasks and the impact of architectural choices within the context of a specific dataset. The architectures that had the best, average, and worst performance are reported, through detailed graphs of the data. The relationship between the architecture of neural networks and progress in training and learning is discussed. The results provide valuable insight for optimizing the design and configuration of neural networks, which may have significant implications for the field of artificial intelligence. This deliverable provides the mathematical foundation of MLP's, serving as a valuable resource for students entering the realm of neural networks.**

I. Introduction

Neural networks are a type of machine learning algorithm that are inspired by the human brain. They are composed of interconnected nodes, called neurons, that learn to perform tasks by adjusting the weights of their connections. Neural networks have been used to solve a wide variety of problems, including image classification, natural language processing, and speech recognition.

The MLP neural network is a feedforward neural network that consists of multiple layers of neurons. The input layer receives the data, the hidden layers perform the computation, and the output layer produces the prediction. The number of layers and the number of neurons per layer can vary depending on the problem being solved.

The architecture of an MLP is defined by the problem the number of layers, the number of neurons per layer, the activation function, and the learning rate are all essential components of the architecture. The number of layers and the number of neurons per layer can affect the complexity of the model and the amount of data needed to train it. The activation function determines how the output of each neuron is processed. And, The learning rate controls how quickly the weights of the network are adjusted during training.

This paper presents a comprehensive analysis of the training of MLP neural networks. The study investigates the impact of architectural choices on the performance of MLP neural networks on a specific dataset. The results provide valuable insight for optimizing the design and configuration of neural networks.

II. Methodology

A. Data

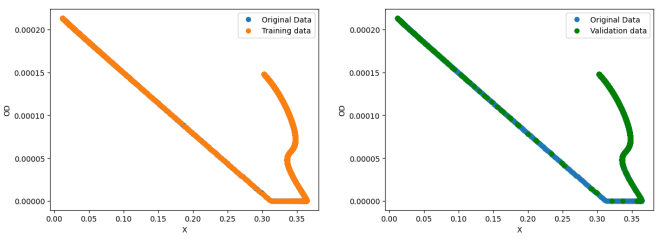
The dataset utilized in this study was obtained from a dynamic model provided by the course professor. To ensure a robust assessment of the Multi-Layer Perceptron (MLP) neural networks, the dataset was divided into three distinct subsets, each serving a unique purpose in our experimentation.

- **Training Set:** encompasses 60% of the data. Used to facilitate the learning process.
- **Testing set:** encompasses 20% of the dataset. Dedicated to evaluating the performance of the MLP models during the training process.
- **Validation Set:** encompasses the last 20% of the data. Used to determine the MLP performance once trained, used in the selection of the best-performing, worst-performing, and mean architectures based on a comprehensive evaluation.

In order to ensure that all inputs features are treated equally and to deal with the different scale of both the inputs and outputs, we perform a range normalization to scale the data to $[0, 1]$.

$$X_{\text{norm}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$$

Where X corresponds to the feature or output to be normalized. The data iytself has 3 columns, X, OD and S. The values that will be taken as inputs will be X and OD while the desired value to predict will be S. A comparison of the normalized data separation through the plot of X against OD was made



B. Algorithm

Each neuron is characterized by a set of weights, which serve as learnable parameters, and is associated with an

activation function. These weights play a pivotal role in capturing the relationships between input features, while the activation function introduces non-linearity into the neuron's computations.

The algorithm itself is based in two main parts in order to emulate a neuron as to how it learns and predicts.

1) Forward Propagation: This emulates when a neuron fires. In general, this process can be explained by , where w_i represents the i^{th} weight of the neuron, x_i the i^{th} input, ϕ the activation function of the neuron and b_i will be called a bias term which, for the scope of this research, will not be included in the implementation of the MLPs for this project.

$$y = \phi(v) = \phi(\sum_{i}^{n} w_i x_i) \tag{1}$$

2) Back propagation: This emulates how a neuron learns.

The problem a neural network is trying to solve can be explaines as a minimization error, where the objective function is:

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{p=1}^{N}\mathcal{E}(p)$$

This is generally called the instant average or global energy of the network. In the minimization of this function several other equations will be present

$$\mathcal{E}_{av} = \frac{1}{N}\sum_{p=1}^{N}\mathcal{E}(p) \tag{2}$$

$$\mathcal{E}(p) = \frac{1}{2}\sum^{k} e^2(p) \tag{3}$$

$$e(p) = y_d(p) - y(p) \tag{4}$$

$$y(p) = \phi(v) \tag{5}$$

$$v = \sum \omega x \tag{6}$$

(3) is the formula for instant energy. (4) is the formula for the error. (5) is the formula for the stimuli or output of the neuron. And, (6) is the formula for the field. Which is just was is fed into the activation function to see if the neuron fires.

To minimize the instant energy, we need the weights need to be adjusted after each iteration by a δw_{ij}.

$$\Delta w_{ij}(n) = -\eta\delta_j(n)y_i(n)$$

where δ_j is the local gradient of the layer j. This is defined by:

The local gradient of the output layer k can be calculated as:

$$\delta_k(n) = e_k(n)\phi'_k(v_k)$$

The local gradient of a hidden layer j can be calculated as

$$\delta_j(n) = \phi'_j(v_j)\sum_k \delta_k(n)\omega_{kj}$$

Finally, it is important to clarify that the training was done batch wise and not sequentially.

C. Architectures

We employ fully connected architectures with varying hidden layers ($L = 1, 2, 3$) and neurons in each hidden layer ($l = 1, 2, 3, 4, 5$), where we specifically set k = 1 to match our single output requirement and n = 2 to accommodate our two features. Furthermore, our approach incorporates three different learning rates ($\alpha = 0.2, 0.5, 0.9$), resulting in a total of 15 distinct architectures and 45 individual models.

III. Results

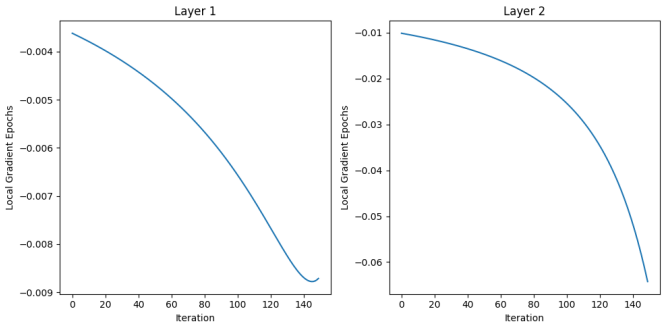
For the sake of consistency every activation function in every layer was the sigmoid function. Which later on in discussion will be explained as to why this was a mistake. The validation error was calculated with the formula:

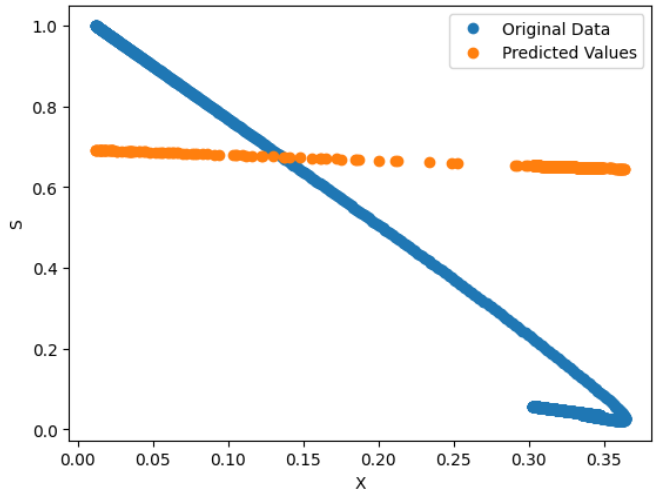
$$\mathrm{err}_S(h) = \frac{|S \cap (h \triangle C_i)|}{|S|}$$

Furthermore, each architecture was tested 50 times, as to ensure statistical significance in the results, and each time it was tested it trained with 50 epochs through 3 iterations.

A. Best

With this specific set of $L = 1$, $l = 3$ and $\alpha = 0.5$ the validation error in average was of 0.168 meaning that 83.2% of the time the error of classification was less than 0.01.

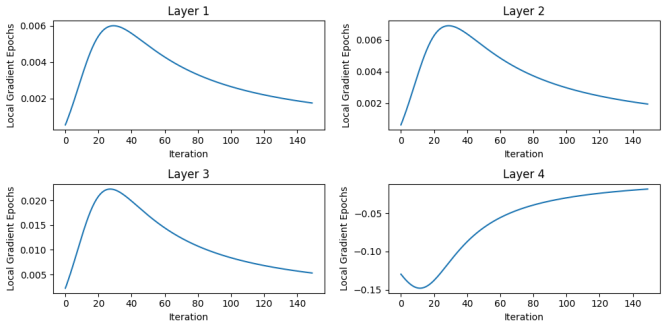


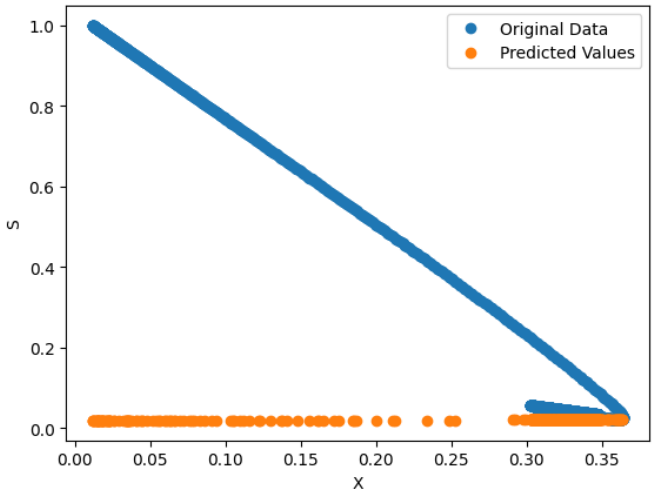


We can see how the local gradients start in extremely low values. This probably means that since the architecture is so small, then it may be that the initial weights were already close to the value that minimized the energy, thus there were no incentives to try and minimize the values.

B. *Worst*

With the set of $L = 3$, $l = 1$ and $\alpha = 0.2$ the validation error obtained was in average 0.698. Which means that in average only 30.2% of the validation set had a classification error of less than 0.01



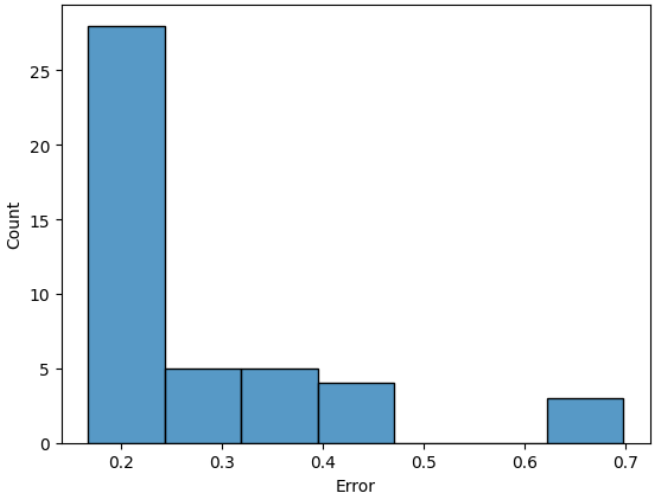


We can see how it just classified evreything as 0.

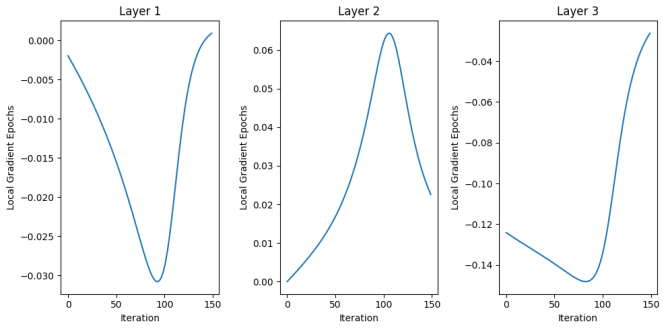
C. *Mean*

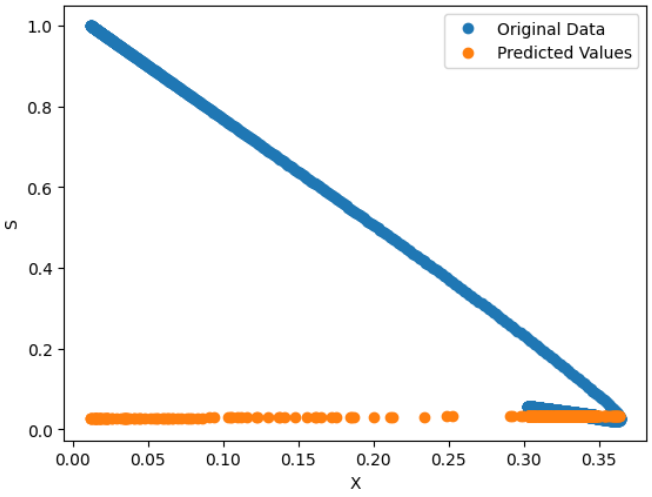
Generally a lot of the architectures tended to give values quite similar, as almost all of the architectures had around a 0.2 validation error as seen in the

Fig. 1. Histogram of all of the architectures validation errors



But the architecture that was closest to the mean was with $L = 2$, $l = 2$ and $\alpha = 0.9$ the validation error in average was of 0.2 meaning that 80% of the time the error of classification was less than 0.01.





IV. Discussion

Two remarkable things were seen through the testing. First, when the number of iterations was larger than 4 and the number of epochs was higher than 50, all of the architectures tended to converge in a validation error of 0.167. This means that all of the networks were able to learn to predict the output with an accuracy of 83.3%.

In further exploration, it was found that this was because all of the networks learned to predict almost equally. This can be seen in Figure III-C. The cause of this is probably the selection of the activation function in each layer as sigmoid and the fact that the bias was missing. The sigmoid activation function is a non-linear function that has a range of [0, 1]. This means that the output of each neuron is always between 0 and 1. And, the absence of the bias can make the network more conservative, leading to predictions that are closer to 0. This suggests that the choice of activation function is highly dependent in two things, the type of problem and the layer, an input layer with a sigmoid function, might kill the stimuli in a forward pass thus making it harder for the learning further on, this may suggest a linear function might be better in this case or another high energy function. However in a classfification problem, since the should be between [0,1] a sigmoid function in the output would be preferred to a linear, since we want to limit the output. Finally the presence of bias can have a significant impact on the performance of an MLP neural network and can help in the prediction of non linearity.

The other remarkable thing seen was that there was an extremely high variability in between tests, This is why each architecture was trained 50 differrent times and their validation error was averaged. In one instance the validation error may have been as low as 0.16 after training, and in the next instance the error may have been as high as 0.94. This high variability is likely due to a number of factors, including: The choice of activation function, as mentioned early with its problems; the absence of bias, may lead the network to be more sensitive to the initial weights, which can lead to different results in different trials; The random initialization of the weights, this means that different trials will start with different weights, which can lead to different results. The noise in the data, this noise can contribute to the variability of the results.

V. Conclusion

The main findings of this paper are that the choice of activation function can have a significant impact on the performance of an MLP neural network. The sigmoid activation function was found to be the most conservative, leading to networks that tended to predict values close to 0. This is because the sigmoid function is a positive-defined function, which limits the outputs of the network. A mix of high-energy functions, such as linear, tanh, and sigmoid, could have improved the performance of the networks.

The findings of this paper have implications for the design and configuration of MLP neural networks. When choosing an activation function, it is important to consider the nature of the problem being solved and the desired output. For example, if the desired output is a binary classification, then a sigmoid activation function may be appropriate. However, if the desired output is a continuous value, then a linear activation function may be a better choice.

The variability of the results was also a major finding of this study. This variability was due to a number of factors, including the choice of activation function, the experimental design matrix, and the random initialization of the weights.

One architecture could give massively different results depending on these factors. Future research could investigate the use of other activation functions and the use of different experimental design matrices in MLP neural networks. A better experimental design matrix could help to reduce the variability of the results and improve the accuracy of the models.

TABLE I
RESULTS FOR ALL ARCHITECTURES

Hidden Layers L	Neurons Per Hidden Layer l	Learning Rate η	Error ϵ
1	1	0.2	0.6588
1	1	0.5	0.4161
1	1	0.9	0.3246
1	2	0.2	0.4161
1	2	0.5	0.2265
1	2	0.9	0.1748
1	3	0.2	0.2629
1	3	0.5	0.1676
1	3	0.9	0.1823
1	4	0.2	0.2255
1	4	0.5	0.1820
1	4	0.9	0.1682
1	5	0.2	0.1706
1	5	0.5	0.1843
1	5	0.9	0.1824
2	1	0.2	0.6858
2	1	0.5	0.3920
2	1	0.9	0.2401
2	2	0.2	0.4272
2	2	0.5	0.2752
2	2	0.9	0.2006
2	3	0.2	0.3839
2	3	0.5	0.1944
2	3	0.9	0.1828
2	4	0.2	0.3701
2	4	0.5	0.2313
2	4	0.9	0.1844
2	5	0.2	0.3603
2	5	0.5	0.1681
2	5	0.9	0.1683
3	1	0.2	0.6983
3	1	0.5	0.2635
3	1	0.9	0.2982
3	2	0.2	0.4384
3	2	0.5	0.2329
3	2	0.9	0.1844
3	3	0.2	0.2968
3	3	0.5	0.2167
3	3	0.9	0.1847
3	4	0.2	0.1835
3	4	0.5	0.1681
3	4	0.9	0.1683
3	5	0.2	0.1840
3	5	0.5	0.1681
3	5	0.9	0.1827