Image Recognition using Neural Networks in Machine Learning

Student: Yi Qiang Ji Zhang
Professor: Dr. Alex Ferrer Ferré
Enginyeria en Tecnologies Aeroespacials
Universitat Politècnica de Catalunya

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

10 June 2021

1 Introduction

A neuron is the basic unit of the neural network. It takes inputs parameters, then does some computation with them, and produces one output. Here's what a 2-input neuron looks like:

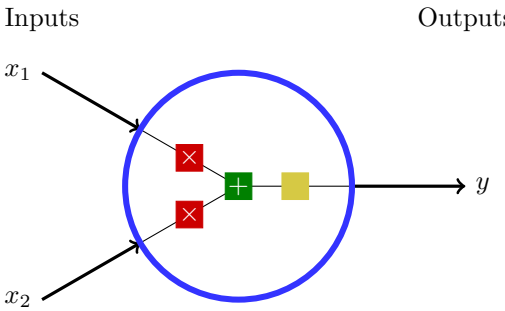


Figure 1. Schematic of a neuron. Source: Victor Zhou [1].

Each input is multiplied by a weight

$$x_1 \longmapsto \omega_1 \cdot x_1 \tag{1}$$

$$x_2 \longmapsto \omega_2 \cdot x_2 \tag{2}$$

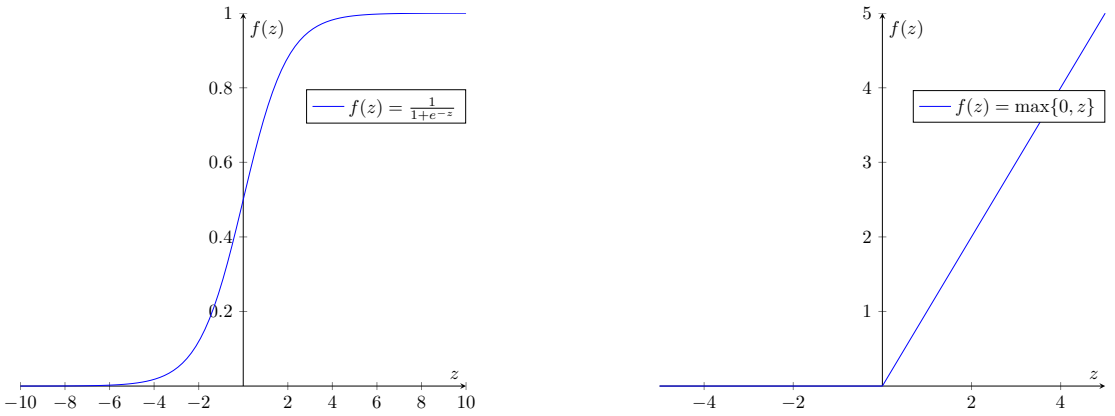
all the weight inputs are added with some bias

$$(\omega_1 \cdot x_1) + (\omega_2 \cdot x_2) + b \tag{3}$$

and finally, the sum is passed with an activation function

$$y = (\omega_1 \cdot x_1 + \omega_2 \cdot x_2 + b) \tag{4}$$

Two commonly used activation functions are the sigmoid function and the ReLu, shown in figure 2. The sigmoid function only outputs numbers in the range $(0, 1)$. You can think of it as compressing $(-\infty, +\infty)$ to $(0, 1)$, big negative numbers become ≈ 0, and big positive numbers become ≈ 1.



(a) Sigmoid function. Source: Own.

(b) ReLU function. Source: Own.

Figure 2. Activation functions

2 Model of Neural Network

A neural network is built by connecting several of our basic "neurons" so that the output of one can be the input of another. Here's an example of a little neural network:

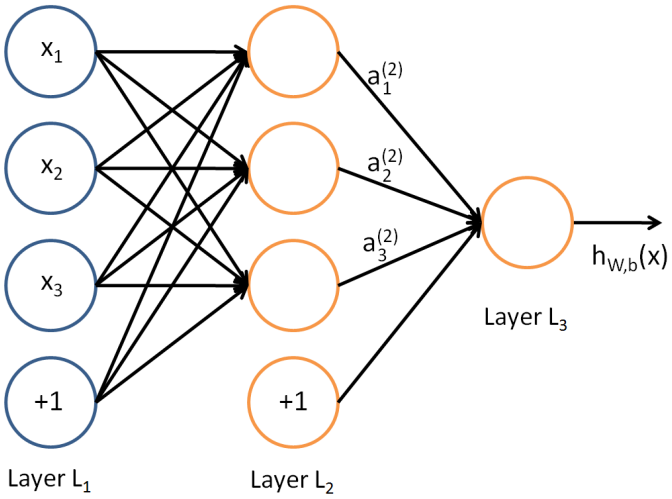


Figure 3. Neural Nerwork

In this diagram, circles represent the network's inputs. Bias units are the circles labeled " +1 " that relate to the intercept term. The network's leftmost layer is known as the input layer, while the network's rightmost layer is known as the output layer (which, in this example, has only one node). Because its values are not noticed in the training set, the intermediate layer of nodes is referred to as the hidden layer.

3 Problem statement

The dataset we're using is the well-known MNIST handwritten digit dataset, which is frequently used in both ML and computer vision applications. It includes 28×28 grayscale photos of handwritten numerals that resemble as follows:

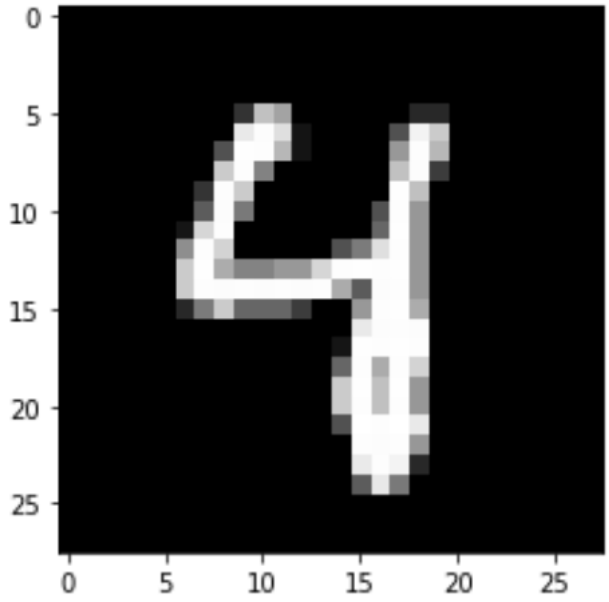


Figure 4. MNIST number example. Source: [2]

What we are seeking is to generate a neural network that is capable of recognizing the numbers from the dataset.

4 Methodology

Each image is labeled with the digit it relates to, ranging from 0 to 9. Our aim is to create a network that can take a picture like this and determine which digit is written in it.

Forward propagation refers to the act of taking an image input and putting it through a neural network to produce a prediction. The prediction generated from a particular picture is determined by the network's weights and biases, or parameters.

To train a neural network, we must update these weights and biases in order to make correct predictions. This is accomplished by a technique known as gradient descent. The main principle behind gradient descent is to work out which direction each parameter may go in to reduce error the most, then push each parameter in that direction again and over until the values with the least error and best accuracy are determined.

Gradient descent in a neural network is accomplished by a method known as backward propagation, or backprop. Instead of running an input image forwards through the network to get a prediction, backprop takes the previously made prediction, calculates the error of how far it was off from the actual value, and then runs this error backwards through the network to determine how much each weight and bias parameter contributed to the error. We may enhance our model by adjusting our weights and biases based on the error derivative terms. If we repeat this process enough times, we will create a neural network that can properly detect handwritten digits.

Softmax takes a column of data at a time, taking each element in the column and outputting the exponential of that element divided by the sum of the exponentials of each of the elements in the input column. The end result is a column of probabilities between 0 and 1.

The Neural Network will feature a straightforward two-layer design. The input layer $a[0]$ will have 784 units, which match to the 784 pixels in each 28x28 input picture. A hidden layer will have 10 units with ReLU activation, and the output layer will have 10 units with softmax activation corresponding to the ten digit classes.

4.1 Forward propagation

Forward propagation will be computed in the following section,

First, let's compute the unactivated values of the nodes in the first hidden layer by applying $W^{[1]}$ and $b^{[1]}$ to the input layer. We'll call the output of this operation $\boldsymbol{Z}^{[1]}$.

$$Z^{[1]} = W^{[1]}X + b^{[1]} \tag{5}$$

The bias matrix has 10×1 dimensions and in column so it is applied to all m columns of the training examples. ReLU is used for this non-linearity. It is a non-linear function which outputs the input parameter if it is above 0 or 0 if the input parameter is below 0.

$$A^{[1]} = g_{\text{ReLU}}(Z^{[1]})) \tag{6}$$

Moreover, a non-linear activation function is needed in order to build the regression model. Otherwise, the sum of the weights multiplied by the base will result in a linear combination when moving from layer to layer. Finally, the last layer yields,

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]} \tag{7}$$

Finally. the last activation function chosen is not the ReLU but a Sigmoid and a softmax since the output shall be from [0, 1] as a probability. Softmax works with a column of data at a time, taking each element in the column and dividing the exponential of that element by the total of the exponentials of all the components in the input column. The ultimate result is a column of probabilities ranging from 0 to 1:

$$A^{[2]} = g_{\text{softmax}}(Z^{[2]}) \tag{8}$$

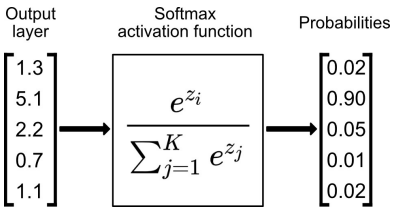


Figure 5. Softmax scheme. Source: [3].

4.2 Backward propagation

In forward propagation we compute a prediction of the result using a given set of weights and biases. However, in backward propagation we will be using the error and update the coefficients according to the labels. This is by using the following cross-entropy loss function:

$$J(\hat{y}, y) = -\sum_{i=0}^{c} y_i \log(\hat{y}_i) \tag{9}$$

Here, $\hat{y}$ is the prediction vector. It might look like this:

$$[0.01\ 0.02\ 0.03\ 0.06\ 0.80\ 0.12\ 0.45\ 0.03\ 0.06\ 0.09]^T \tag{10}$$

And y is the one-hot encoding of the correct label for the training example. If the result is 5, the one-hot encoding of y would look like this:

$$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T \tag{11}$$

It's important to notice that the sum $\sum_{i=0}^{c} y_i \log\left(\hat{y}_i\right), y_i = 0$ for all i except the correct label. It's worth noting that the closer the prediction probability is to one, the closer the loss is to O. The loss approaches $+\infty$ as the likelihood approaches zero. We increase the accuracy of the model by reducing the cost function. Over several rounds of gradient descent, we subtract the derivative of the loss function with respect to each parameter from that parameter.

$$\begin{aligned} W^{[1]} &:= W^{[1]} - \alpha \frac{\delta J}{\delta W^{[1]}} \\ b^{[1]} &:= b^{[1]} - \alpha \frac{\delta J}{\delta b^{[1]}} \\ W^{[2]} &:= W^{[2]} - \alpha \frac{\delta J}{\delta W^{[2]}} \\ b^{[2]} &:= b^{[2]} - \alpha \frac{\delta J}{\delta b^{[2]}} \end{aligned} \tag{12}$$

The objective in backward propagation is to find $\frac{\delta J}{\delta W^{[1]}}$, $\frac{\delta J}{\delta b^{[1]}}$, $\frac{\delta J}{\delta W^{[2]}}$, and $\frac{\delta J}{\delta b^{[2]}}$.

By using the chain rule, it is possible to obtain the cost function derivatives

$$\begin{aligned} \mathrm{d}Z^{[2]} &= A^{[2]} - Y \\ \mathrm{d}W^{[2]} &= \frac{1}{m}\, \mathrm{d}Z^{[2]}\, A^{[1]T} \\ \mathrm{d}B^{[2]} &= \frac{1}{m} \Sigma\, \mathrm{d}Z^{[2]} \\ \mathrm{d}Z^{[1]} &= W^{[2]T}\, \mathrm{d}Z^{[2]} * g^{[1],}(z^{[1]}) \\ \mathrm{d}W^{[1]} &= \frac{1}{m}\, \mathrm{d}Z^{[1]}\, A^{[0]T} \\ \mathrm{d}B^{[1]} &= \frac{1}{m} \Sigma\, \mathrm{d}Z^{[1]} \end{aligned}$$

4.3 Parameter updates

After finding the correpondand derivatives, we shall update the weights and biases:

$$\begin{aligned} W^{[2]} &:= W^{[2]} - \alpha\, \mathrm{d}W^{[2]} \\ b^{[2]} &:= b^{[2]} - \alpha\, \mathrm{d}b^{[2]} \\ W^{[1]} &:= W^{[1]} - \alpha\, \mathrm{d}W^{[1]} \\ b^{[1]} &:= b^{[1]} - \alpha\, \mathrm{d}b^{[1]} \end{aligned}$$

Where α is the hyper parameter which is arbitrary. The user adjusts the parameter.

Finally, this process is done several times as for getting better more accurate version of the parameters.

5 Results

5.1 1 layer

First, we shall analyse the case for 1 layer, as seen in the following figures.

Two activation functions were analysed as explained previously. In Figures6 and 7, it shows how the accuracy for the Sigmoid activation function is way higher and faster than ReLU in this particular example, and set of images. The main reason behind this may reside in the fact that the main difference between ReLU and Sigmoid is that ReLu is faster to compute than the sigmoid function, and its derivative is faster to compute. This makes a significant difference to training and inference time for neural networks: only a constant factor, but constants can matter.

NOTE: X-axis named "layer" are "Epoch"

5.1.1 3 epoch

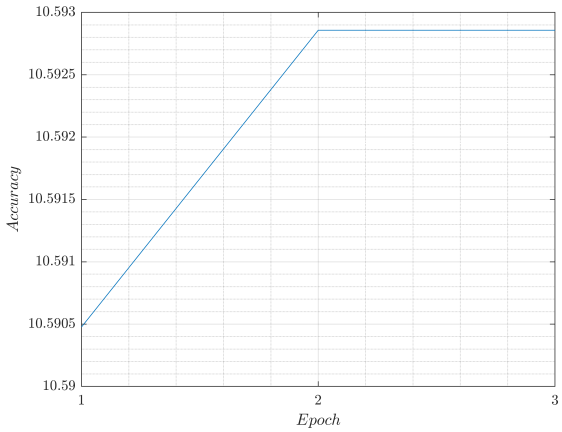


Figure 6. Accuracy for 10 nodes with 3 epoch using RELU. Source: Own.

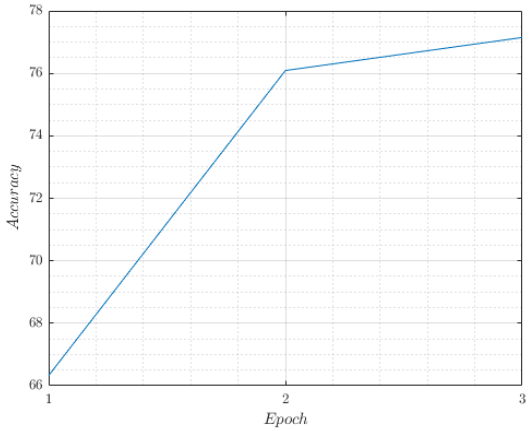


Figure 7. Accuracy for 10 nodes with 3 epoch using Sigmoid. Source: Own.

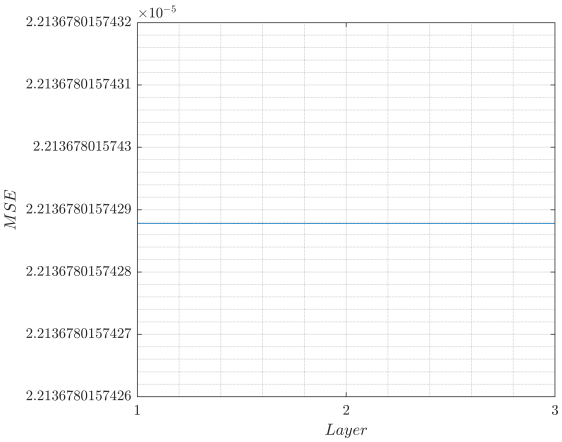


Figure 8. MSE for 10 nodes with 3 epoch using RELU. Source: Own.

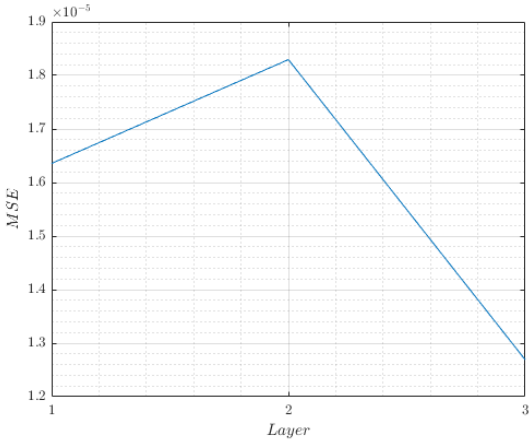


Figure 9. MSE for 10 nodes with 3 epoch using Sigmoid. Source: Own.

In FIgure 6, the accuracy using ReLU is significantly smaller than Sigmoid activation function, in which at epoch 3, it almost reaches 78% accuracy. The fact that ReLU activation's accuracy is so small may happen the

neuron outputs might become very large, and the network may suffer from numerical stability in the operations. It also slows down the network by the way even if it learns.

By taking a look at the MSE for each each, a strange behaviour happens for ReLU. The error seems to become nearly constant, this is due to the fact that the accurracy has not even increased from one epoch to the other. Whereas for the Sigmoid's MSE, as the Neural Network increases the acurracy, the the MSE also reduces itself.

5.1.2 10 epoch

Now, changing the number of iterations to 10 epochs, we still see how ReLU is behaving anormally, wich the accuracy decreases. However, the accuracy for the Sigmoid grows from around 78% to above 80%.

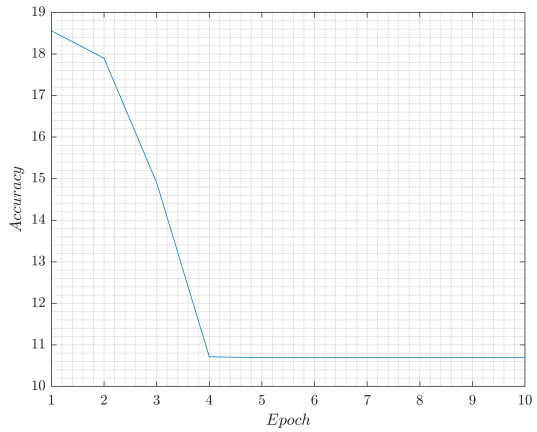


Figure 10. Accuracy for 10 nodes with 10 epoch using RELU. Source: Own.

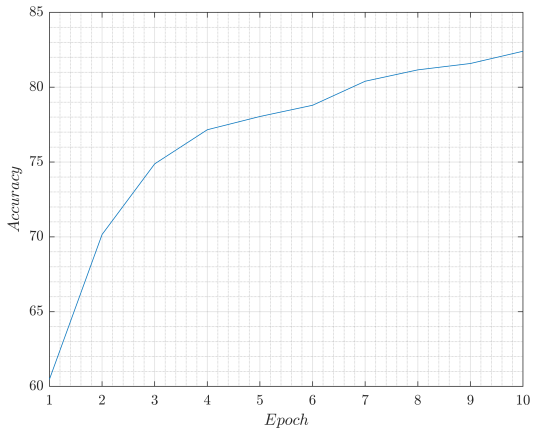


Figure 11. Accuracy for 10 nodes with 3 epoch using Sigmoid. Source: Own.

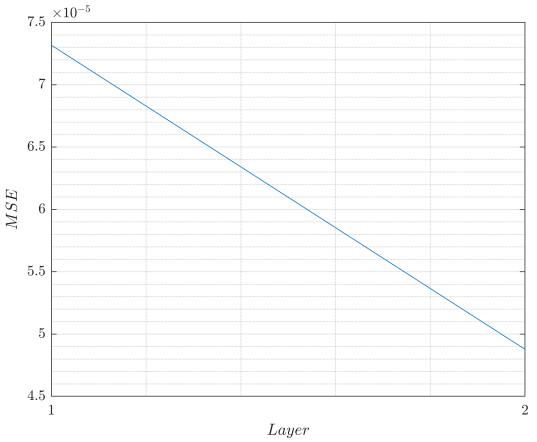


Figure 12. MSE for 10 nodes with 10 epoch using RELU. Source: Own.

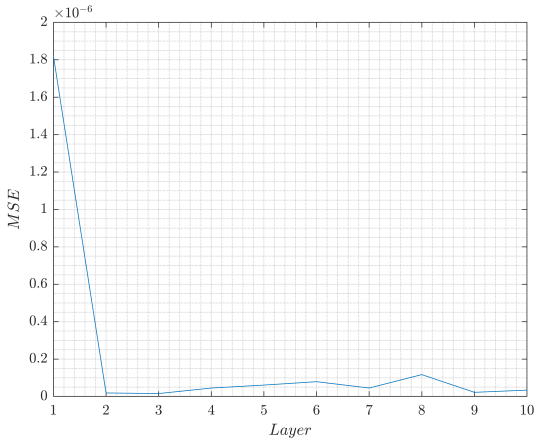


Figure 13. MSE for 10 nodes with 10 epoch using Sigmoid. Source: Own.

The same reasoning happens again, the MSE for the Sigmoid activation has reduced even more. Notice how there is a slight spike in layer

5.1.3 100 epoch

Finally, by doing 100 epochs

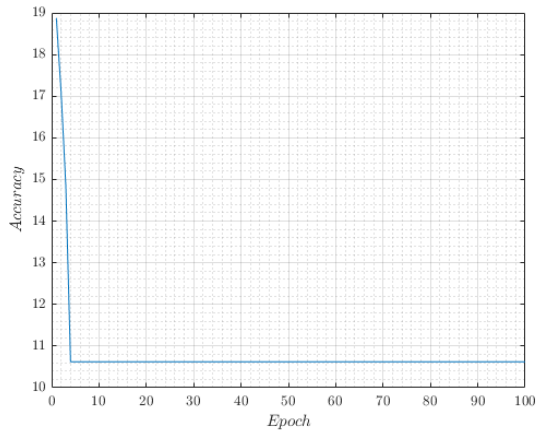


Figure 14. Accuracy for 10 nodes with 100 epoch using RELU. Source: Own.

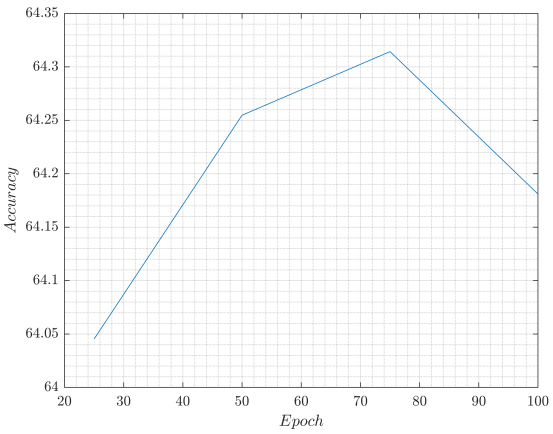


Figure 15. Accuracy for 10 nodes with 100 epoch using Sigmoid. Source: Own.

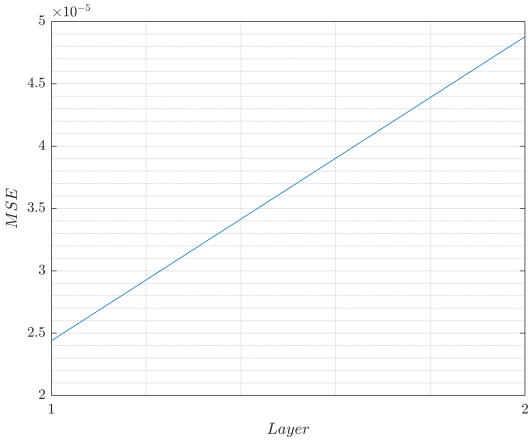


Figure 16. MSE for 10 nodes with 100 epoch using RELU. Source: Own.

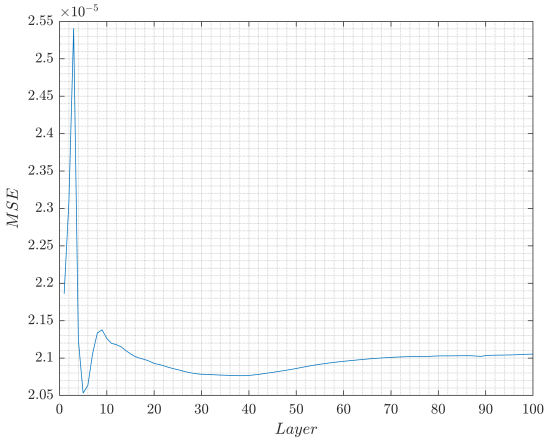


Figure 17. MSE for 10 nodes with 100 epoch using Sigmoid. Source: Own.

Figure 17 shows the impact of changing the epochs in the performance of the neural network once optimised. Notice how the MSE reduces as the number of iterations reduces as well.

Note how as it approaches zero its effect gets neutralised, whilst for slightly greater values the test MSE gets reduced. The optimal value is found to be around $1.5 \cdot 10^{-5}$.

The percentage value is that of the test set, and the results are consistent with the fact that the fewer the training data, the greater the variance in the results obtained by the neural network.

5.2 2 layers

Once we have seen the results using one single layer, the next step is to increase the number of layers and see how the Neural Network performs.

5.2.1 3 epoch

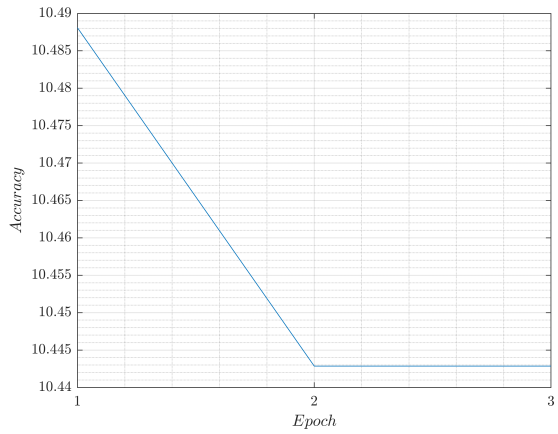


Figure 18. Accuracy for 10 nodes with 3 epoch using RELU and learning rate $\alpha = 0.01$ (2 layers). Source: Own.

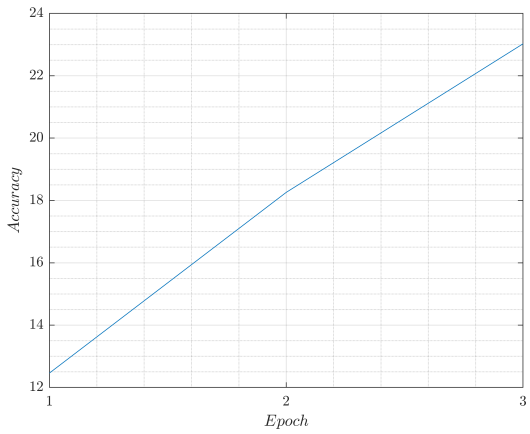


Figure 19. Accuracy for 10 nodes with 3 epoch using Sigmoid and learning rate $\alpha = 0.01$ (2 layers). Source: Own.

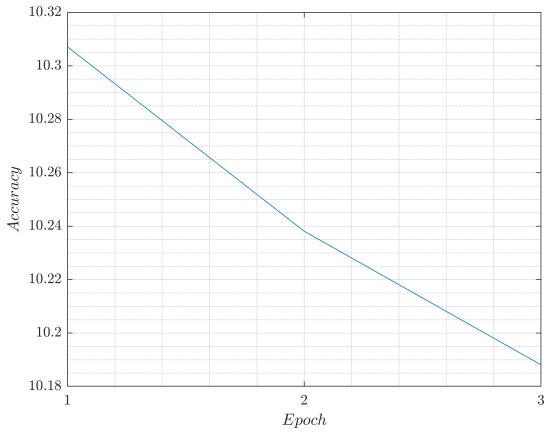


Figure 20. Accuracy for 10 nodes with 3 epoch using ReLU (2 layers) with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

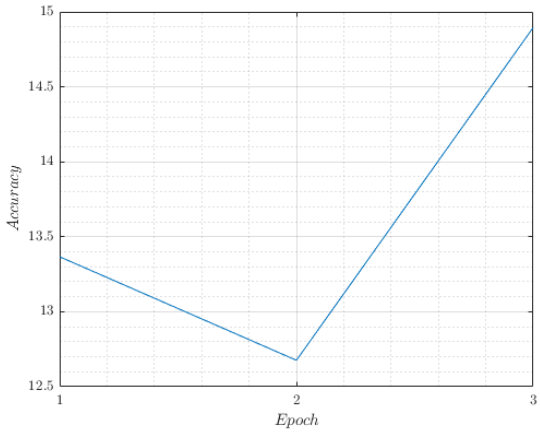


Figure 21. Accuracy for 10 nodes with 3 epoch using Sigmoid (2 layers) with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

The accuracy of the ReLU activation function still continues to decrease,
As seen in the prior figures, the hyper parameter learning rate α does not change the behaviour that much.

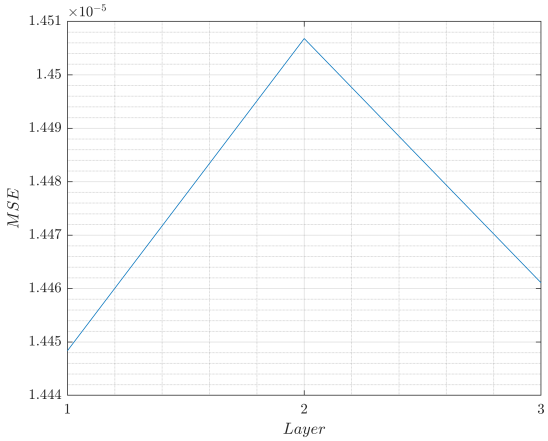


Figure 22. MSE for 10 nodes with 3 epoch using ReLU with learning rate $alpha = 0.01$ (2 layers). Source: Own.

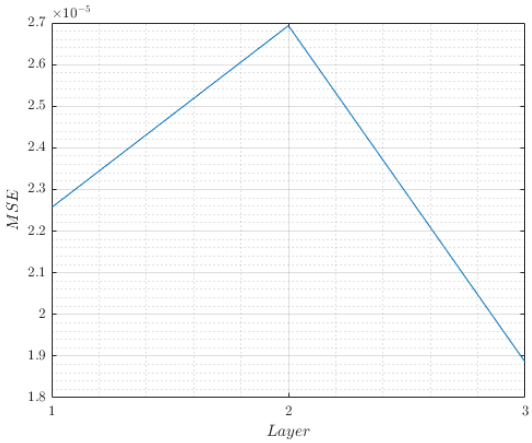


Figure 23. MSE for 10 nodes with 3 epoch using Sigmoid with learning rate $alpha = 0.01$ (2 layers). Source: Own.

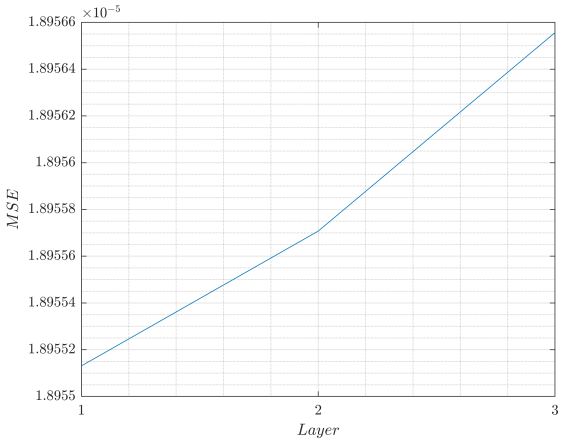


Figure 24. MSE for 10 nodes with 3 epoch using ReLU with learning rate $alpha = 0.1$ (2 layers). Source: Own.

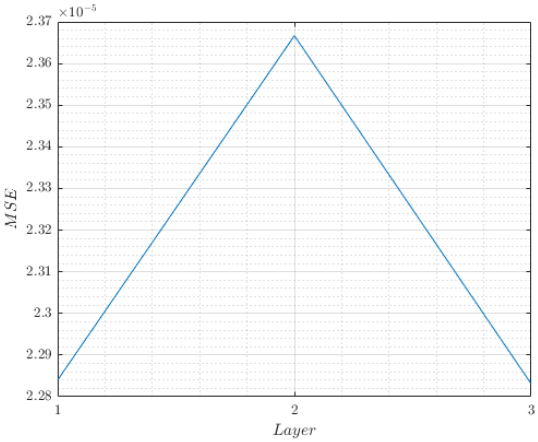


Figure 25. MSE for 10 nodes with 3 epoch using Sigmoid with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

5.2.2 10 epoch

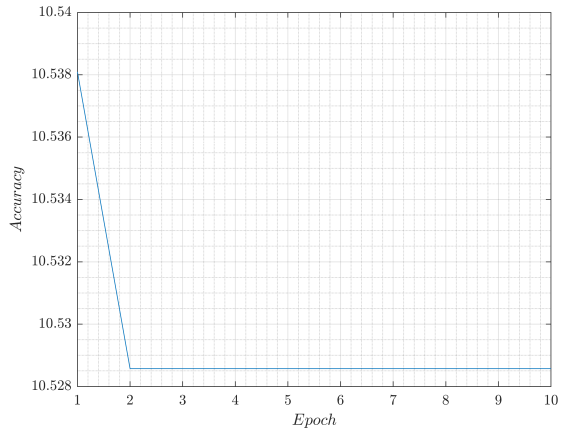


Figure 26. Accuracy for 10 nodes with 10 epoch using RELU (2 layers). Source: Own.

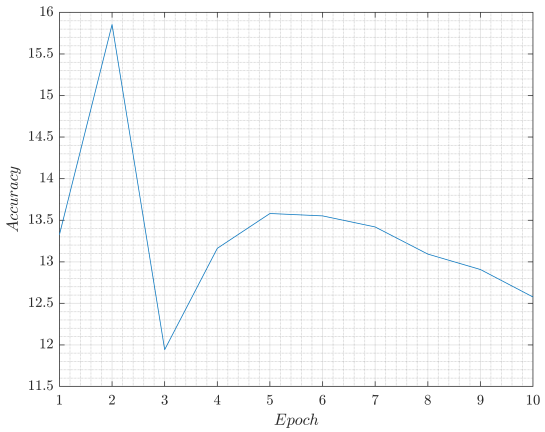


Figure 27. Accuracy for 10 nodes with 10 epoch using Sigmoid (2 layers). Source: Own.

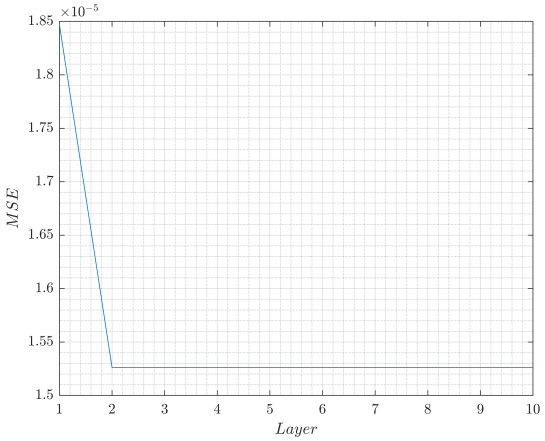


Figure 28. MSE for 10 nodes with 10 epoch using ReLU with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

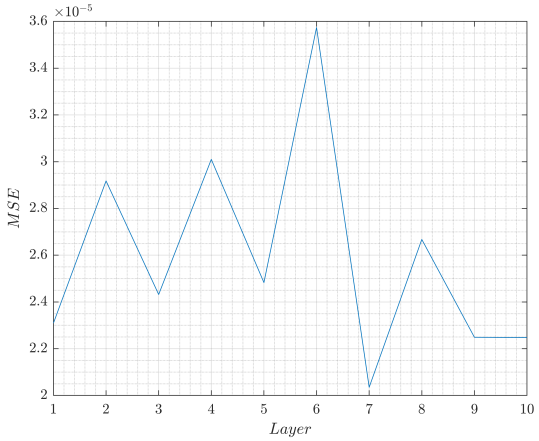


Figure 29. MSE for 10 nodes with 10 epoch using Sigmoid with learning rate $\alpha = 0.1$ (2 layers). Source: Own.

5.2.3 100 epoch

Finally, using 100 epochs, the behaviour for the ReLU remains the same. However, unexpectedly, one would expect the Sigmoid activation plot to increase, increasingng the value of the epoch, the results show how the accuracy increases with the Sigmoid, however, in a point it decreases back, this is problem resides in that the the fact that the set is trained excessively with the amount of iterations. Thus, it produces and over-fitting of the weights for a specific set of image. That is why the accuracy of the Sigmoid does not increase.

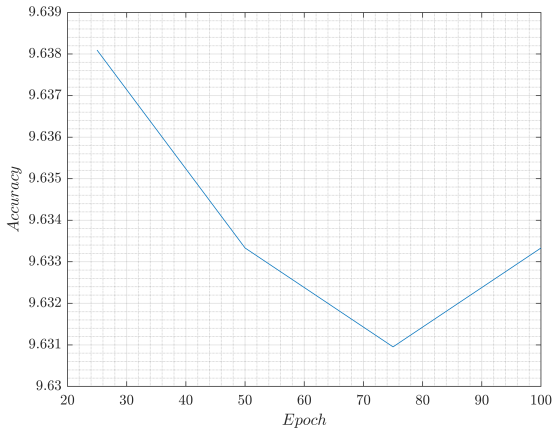


Figure 30. Accuracy for 10 nodes with 100 epoch using RELU (2 layers) $\alpha = 0.1$. Source: Own.

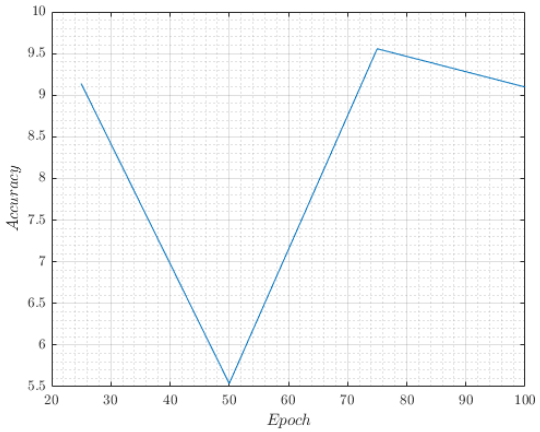


Figure 31. Accuracy for 10 nodes with 100 epoch using Sigmoid (2 layers) $\alpha = 0.1$. Source: Own.

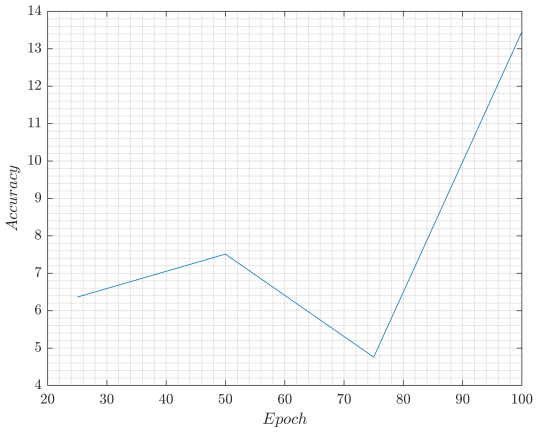


Figure 32. Accuracy for 10 nodes with 100 epoch using Sigmoid with learning rate $\alpha = 0.01$ (2 layers). Source: Own.

When increasing the α, the accuracy seems to recover a bit. Nevertheless, by taking a look at the MSE, it confirms the hyphotesis stated before. In which the MSE oscillates back and forward trying to adjust the weights.

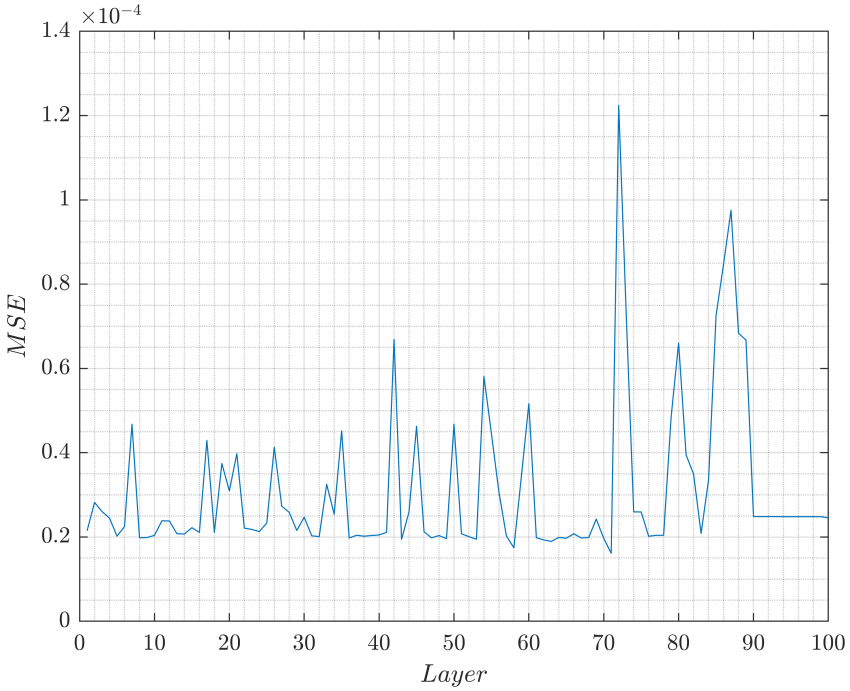


Figure 33. MSE for 10 nodes with 100 epoch using Sigmoid with $\alpha = 0.01$ (2 layers). Source: Own.

Below are some images for the predictions

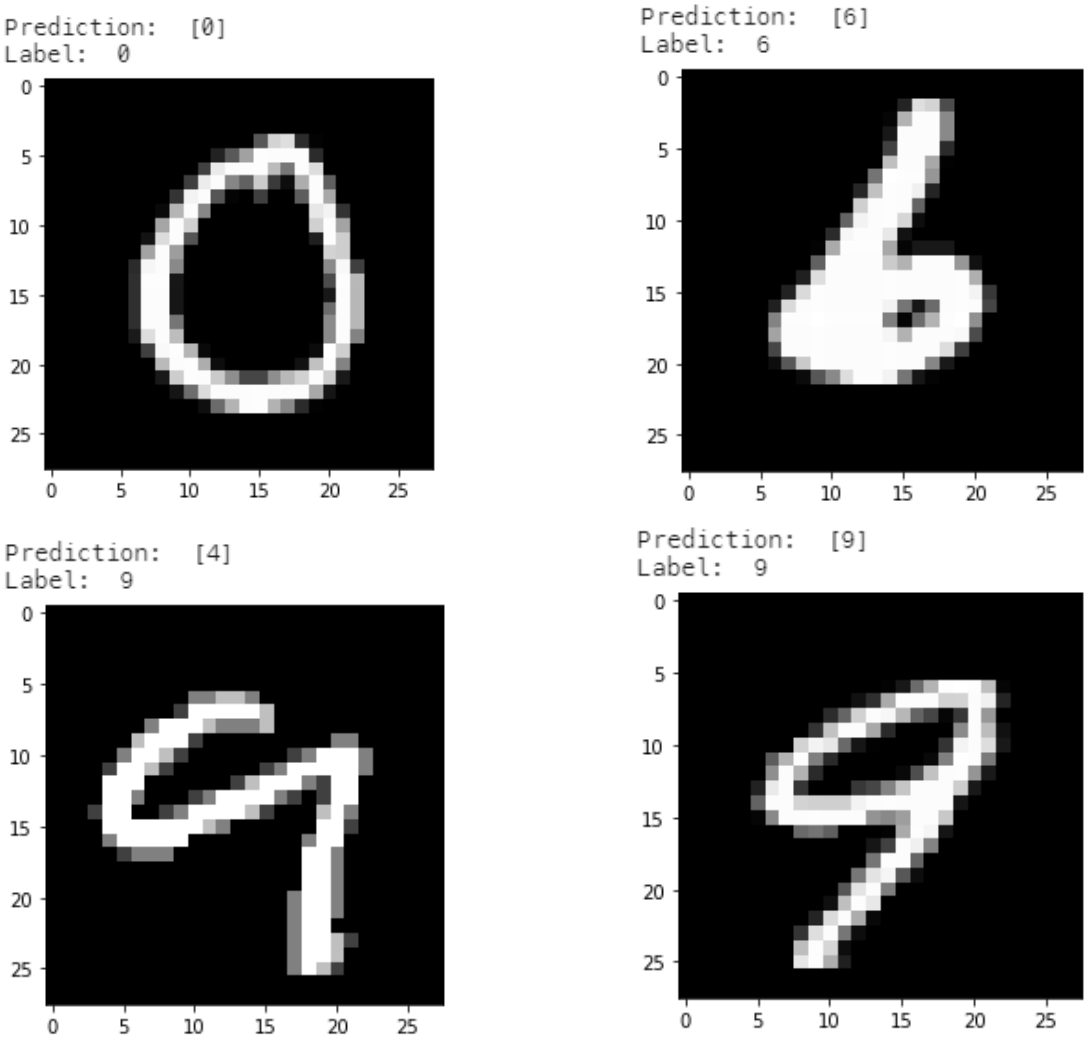


Figure 34. Train test prediction and label. Source: Own.

Also, using the above coefficients for the test set given by the dataset, the accuracy was about 88.1 %. As regards the sigmoid one, the accuracy was lower, 80.4 %. As we can see, the wrong predictions are quite forgiveable since they're in some cases even hard to recognize for the human reader.

References

[1] Victor Zhou. *Machine Learning for Beginners: An Introduction to Neural Networks*. 2021. URL: https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9.

[2] Kaggle. *Digit Recognizer*. 2021. URL: https://www.kaggle.com/c/digit-recognizer/data.

[3] Dario Radečić. *Softmax Activation Function Explained*. 2021. URL: https://towardsdatascience.com/softmax-activation-function-explained-a7e1bc3ad60.

[4] Stanford. *Multi-Layer Neural Network*. 2021. URL: http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks/.

[5] Samson, Zhang. *Understanding the math behind neural networks by building one from scratch (no TF/Keras, just numpy.* 2021. URL: https://www.samsonzhang.com/2020/11/24/understanding-the-math-behind-neural-networks-by-building-one-from-scratch-no-tf-keras-just-numpy.html.