Neural Networks Implementing a Neural Network from scratch

Meaza Eyakem Gebreamlak, Sebastián Cajas Ordoñez,
*University of Bordeaux.

I Introduction

During the present work, two different neural networks are presented, the first one performing binary classification between the number zero and different numbers, and secondly, a multi-classification task for classifying 10 different types of numbers using the MNIST dataset [1]. For this task, some processing has been performed, such as normalizing the dataset and in the multi-classification task, hot-encoding format has been implemented to adapt the same code into a multi-class problem. Similarly, the equations for gradient descent have been employed, as well as cross-entropy to validate the loss during training.

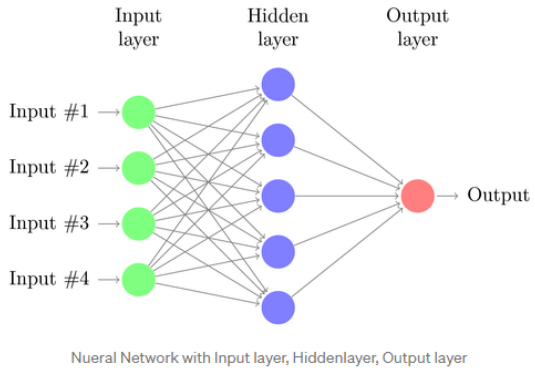

Fig. 1. Neural network structure)

II Literature

A. Neural Networks

Neural Networks are a set of approaches that were first designed to mimic the functions of neurons in the human brain. The human brain has a complex way of communicating with each other that allows us to process the data we obtain. This is the function that Artificial neural networks are designed to copy and try to make machines as intelligent as possible like human beings.

Neural networks are a combination of an input layer, a different number of hidden layers, and an output layer. The layers are composed of the basic functional unit Node. A node is a unit that receives an input and processes the output using an activation function. It can be observed from figure 1 that the network is built by connecting the nodes of the previous layer with the nodes of the current layer up to the output layer. Each node has its associated weights and bias that determines the learning process.

B. Forward Propagation

The network learns the important details of the input data by sequential training. First, the output is computed by propagating the outputs through the layers up to the output layer.

As we can see from figure 3 Individual inputs have their associated weights. In the first layer, the output is computed by summing up the multiplication of the weight and input as seen in equation 1. This result then passes through the activation function to yield the output. An activation function is a nonlinear function that determines the output from the weighted sum of inputs. We have different activation functions, such as sigmoid, tanh, relu, rectified relu. In this assignment, the sigmoid activation function is used.

$$Z = \sum_{n=1}^{M} WnXn + b \tag{1}$$

where w is the wights of the nodes, x is the input and b is the bias.

Sigmoid Function

The network will learn gradually by updating the associated weights and bias values.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

This activation function will be useful during the multi-classification tasks. However we need to consider that it also creates a vanishing gradient problem. Known as a smoothed version of the Heaviside step function. When x=0, f(x) is 0.5, and therefore when x takes a value too small or too big, then the function tends to get zero, in particular for negative values of x. Then, small gradient descents are always zero

when x is too big, which is a problem

During the MNIST dataset, each neuron will have different values, which will represent the possible outcomes, so the higher/lower the value for an activation zone will mean the representation of some part of the data (edges or corners for example). So since this values can be positive and negative, we need to normalize them, for than we can use the Sigmoid Squishification function.

C. Back Propagation

For the network to learn the weights and biases it should optimize its cost function. The cost function is the function that calculated how much the predicted value has deviated from the true value. In this assignment, cross-entropy is used. To optimize the cost function Gradient descent algorithm is used. Gradient descent algorithm uses the derivative of the parameters to find the optimal value. It is done backwardly starting from the output layer. The algorithm aims to update the values of weight and bias to the optimal value by calculating the derivative of the cost function with respect to each variable.

Binary Classification

The steps are taken to classify the input are. First, the output is computed using forward propagation. After computing the predicted value the cost function is calculated using true and predicted values using cross-entropy loss. Second, the derivatives of the cost function with respect to the individual variable are computed using a gradient descent algorithm. As shown below. After having the gradients the weights and biases are updated using the gradients. Finally, the updated weights are you to predicts the input image.

The gradient descent algorithm is shown in the following. we will treat it with a single Sigmoid activation function and the implementation of the loss function.

Gradient Descent algorithm:
$j = 0,\ dW_1,\ dW_2,\ db = 0$
$For\ i = 1\ \text{to}\ M$
$z_i = W.T * X_i + b$
$a_i = sigmoid(z_i)$
$J+ = -[y^i * Loga_i + (1 - y^i)Log(1 - y^i)]$
$dZ^i = a^i - y^i$
$dW_1+ = x_1^i$
$dW_2+ = x_2^i$
$db+ = dz^i$
$J/= m$
$dW_1 = \frac{dJ}{dW_1}$
$W_1 = W_1 - alpha * dW_1$
$W_2 = W_2 - alpha * dW_2$
$b = b - alpha * db$

Multi-class classification. The main difference will be set on the loss function, as we will be considering multiple types of outputs in the range 0 to 9. On this regard, we will be creating a multi-classification task as shown on the figure 2 we will be using categorical cross-entropy loss, given by the following:

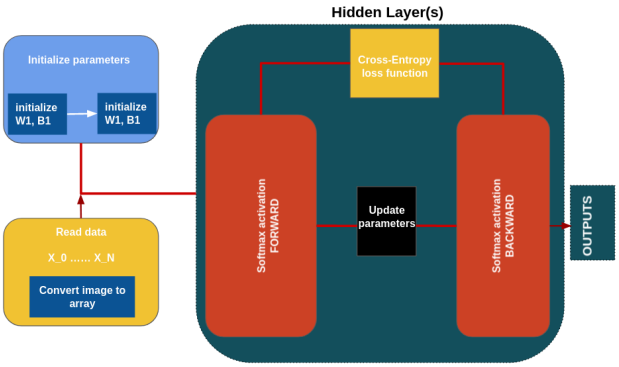

Fig. 2. **Training diagram** for multi-classification using 1 hidden layer with 64 neurons, cross-entropy loss function over *MNIST* dataset.

$$Loss = -\sum y_i * Logy_i^{'} \tag{3}$$

The equation calculates the loss of an example by comparing the models output versus the labeled sample. It will have a negative sign since we are intending to decrease as we perform the training. This is also a more convenient way to work all along with the *Softmax* function, since in the case of multi-classification problems it is known for outperforming gradient descent results.

Sigmoid function

This activation function will be useful during the multi-classification tasks. However we need to consider that it also creates a vanishing gradient problem. Known as a smoothed

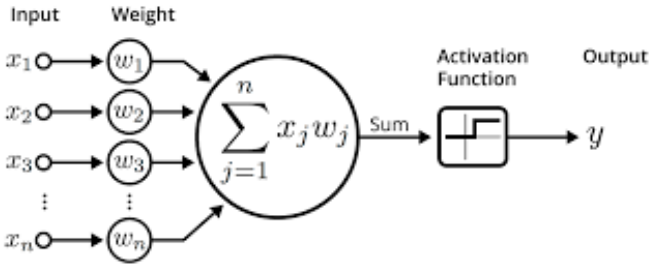

Fig. 3. Single Layer Neural network structure)

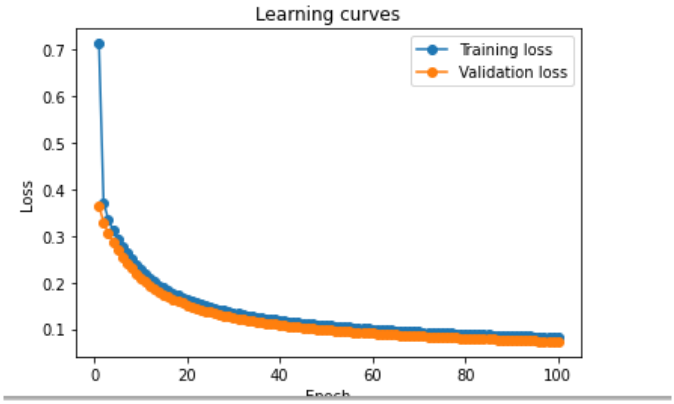

Fig. 4. Learning curve for 100 epochs using 0.001 learning rate)

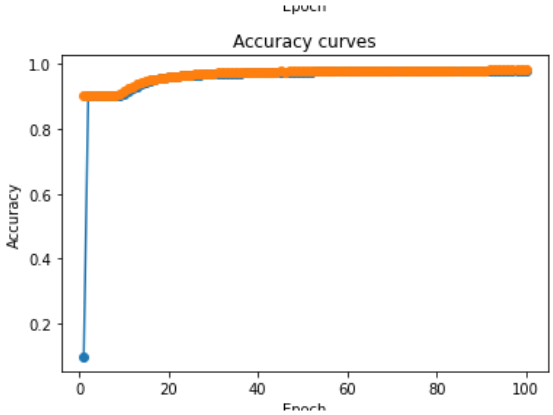

Fig. 5. Accuracy for 100 epochs using 0.001 learning rate)

version of the Heaviside step function. When x=0, f(x) is 0.5, and therefore when x takes a value too small or too big, then the function tends to get zero, in particular for negative values of x. Then, small gradient descents are always zero when x is too big, which is a problem

During the MNIST dataset, each neuron will have different values, which will represent the possible outcomes, so the higher/lower the value for an activation zone will mean the representation of some part of the data (edges or corners for example). So since this values can be positive and negative, we need to normalize them, for than we can use the Sigmoid Squishification function.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

III Analysis and Results

A. Binary classification

This section focuses on the analysis of the first part of the assignment which is the single neural network. Before starting training different preprocessing steps are applied to the dataset.The input features are reshaped and normalized before passing the network.

Figure 4 shows the learning curve of the network using 100 epochs and 0.01 value of the learning rate. As the learning rate determines the how fast the network learns the higher the faster, the lower the slower. However, faster can mean it may lose details.In this particular case using when we increase the learning rate in decimal value like from 0.001 to 0.1 the accuracy increases, but in the integer value like from 1 to 5 the accuracy decreases. If we see the curves of different learning rates we can obseve the curve decreases fast at the begning for larger learning rates.

TABLE I
RESULTS OF ACCURACIES FOR DIFFERENT LEARNING RATES

Learning rate	Acc - Epoch: 100	Acc - Epoch: 200
0.001	90.2	90.2
0.01	91.35	95.84
0.1	98.19	98.69
1	899.1	99.21
5	899.1	99.21

When we see the effect of the epoch, in the beginning the loss decreases when the number of epochs is increased.

Figure 5 shows the accuracy of the network with respect to the number of epochs. The accuracy increases as the number of epochs are increased. After some level it becomes nearly constant.

B. Multi-classification

On this case, the binary code will be re-adapted to perform the multi-classification using Fully Convolutional Layers [2]. As specified at the beginning, the loss function will be changed. While for binary classification we were using gradient descent, on this case we will be adapting it for cross-entropy loss. A general diagram composing the system for the code *lab2_3_skelethon* is depicted in

Hyper-tuning at different learning rates have been tested, including the range $2, 1, 0.5, 0.2, 0.1, 0.01, 0.001, 0.0001$ on a for loop, accomplishing the results shown on table II. From this table, it is important to highlight that in order to obtain the best results, only the learning rate can be customized, obtaining the best results when it is set on the range between 0.1 and 0.6, getting a maximum accuracy of 90.31% when learning rate is 0.6. This value was determined empirically using a for-loop. If more hidden layers had been proposed, results could have improved even more.

From the table it is also possible to observe an interesting behaviour when learning rate is too high (above 0.5) or when it is too small (below 0.1), this can be explained because the model will converge too quickly to the optimal solution in less steps, or in the contrary, when it is too small: it might not converge at all as it can get stuck in a middle point as it

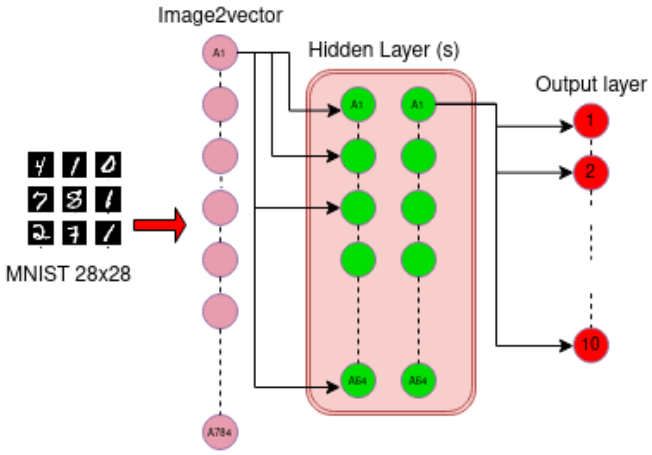

Fig. 6. **Neural network architecture** for multi-classification using 1 hidden layer with 64 neurons, cross-entropy loss function over *MNIST* dataset. Output layer defined to fit 10 numbers.

TABLE II
RESULTS OF ACCURACIES FOR DIFFERENT LEARNING RATES

Learning rate	Acc - Epoch: 100	Acc - Epoch: 290
2	9.871	19.255
1	59.733	89.185
0.6	83.968	90.31
0.5	83.501	89.565
0.4	80.71	88.696
0.3	74.578	87.0
0.2	57.988	84.666
0.1	18.726	73.186
0.01	9.871	9.871
0.001	9.871	9.871
0.0001	9.871	9.871

would be advancing too slowly.

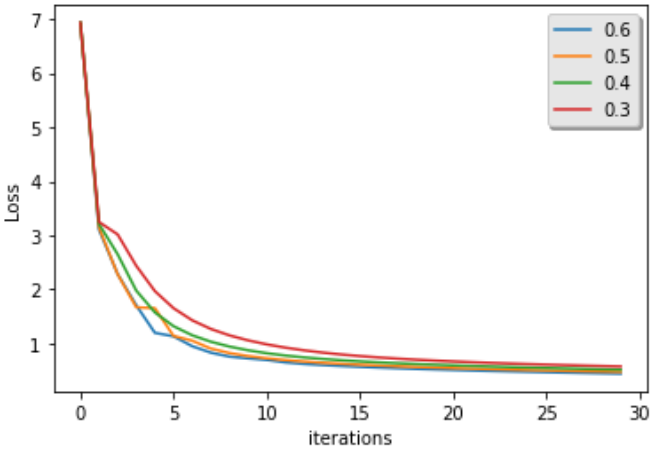

Fig. 7. **Loss curve** for multi-classification using 1 hidden layer with 64 neurons, cross-entropy loss function over *MNIST* dataset while varying multiple learning rates.

From Figure 8 we can observe that the differences for the chosen values do not change aggressively. Other values were tested as depicted in table II, however, on this work we focused on obtaining the best possible results using 1 hidden layer and also adding another layer.

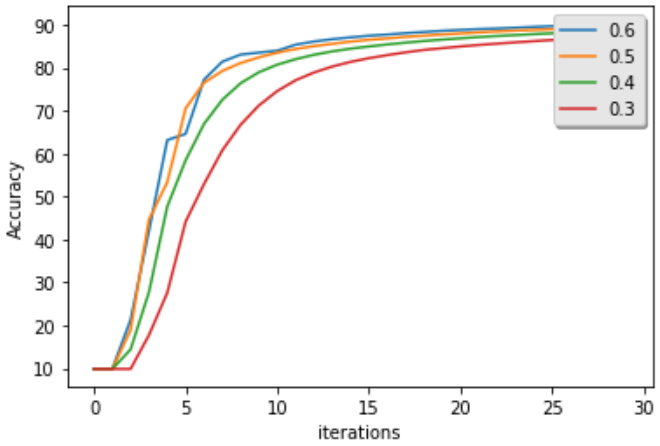

Fig. 8. **Accuracy curves** for multi-classification using 1 hidden layer with 64 neurons, cross-entropy loss function over *MNIST* dataset while varying multiple learning rates.

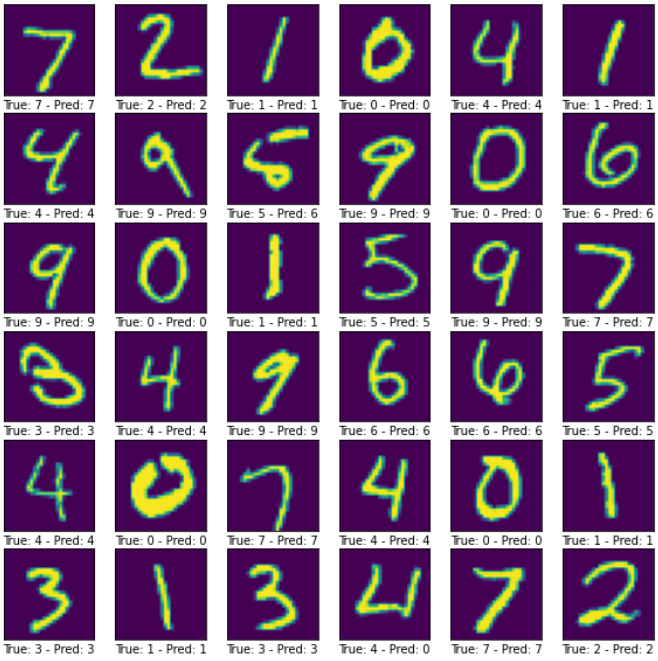

Fig. 9. **Predictions** for multi-classification using 1 hidden layer with 64 neurons, cross-entropy loss function over *MNIST* dataset while varying multiple learning rates.

Finally, the figure 9contains some examples for the best trained model, which is the one with learning rate of 0.6. On it we can see that from 36 different images, only 1 error is detected, where a 4 is predicted as a 0.

IV Conclusions

- For the multi-classification task, it was possible to observe as well that the smaller the learning rate, the more the needed epochs are needed in order to fit the model, and on the contrary, when the learning rate is too large, it will need less epochs in order to converge optimally.

The training time was also proportional to the size of the learning rate, the smaller, the more time of training and viceversa.

References

[1] "THE MNIST DATABASE of handwritten digits". Yann LeCun, Courant Institute, NYU Corinna Cortes, Google Labs, New York Christopher J.C. Burges, Microsoft Research, Redmond.

[2] Liou, D.-R.; Liou, J.-W.; Liou, C.-Y. (2013). Learning Behaviors of Perceptron. iConcept Press. ISBN 978-1-477554-73-9.