Building an Artificial Neural Network from Scratch

Javier Páez Franco

February - March 2023

1 Introduction

This is a report of the work done by Javier Páez Franco for his GitHub repository, "Artificial Neural Network (ANN)". In this document, I discuss the process of the limitations of a single perceptron, developing a multilayer perceptron, how it performs, and comparing its results to the Scikit-learn solution.

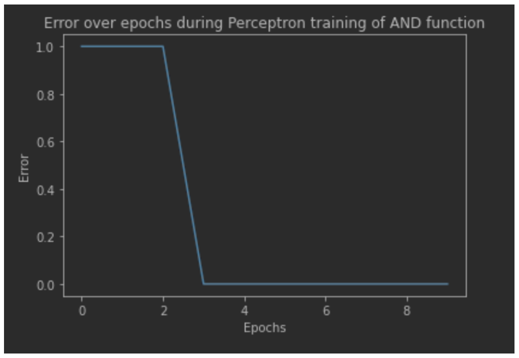
The network comprises multiple interconnected layers of neurons that are trained to classify a 10-feature input into seven distinct classes. I discuss the use of various network architectures and hyperparameters tuning to obtain the best performance on the test data set, which is studied through cross-validation.

The purpose of this work was learning how neural networks work, by building one completely from scratch, without using any libraries or frameworks (such as PyTorch or Scikit-learn), except for NumPy (as we can really speed up calculations with this library). Additionally, more advanced concepts such as L2 regularization or the Adam Optimization Algorithm were implemented too. The final purpose was to test whether my work can be compared to other solutions, and more specifically against Scikit-learn.

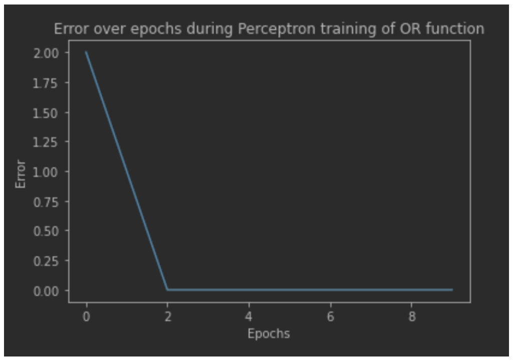
2 Architecture

2.1 A single perceptron

First, I will develop (program) a single perceptron. You can find the code in the file "NeuralNetwork.py", the class `Perceptron`. I show with three graphs (one per logic function: AND, OR, and XOR) that present the error over epochs. Thanks to this, I see that the perceptron is able to learn the OR and the AND functions, but not the XOR function.



(a) Error over 10 epochs for AND gate

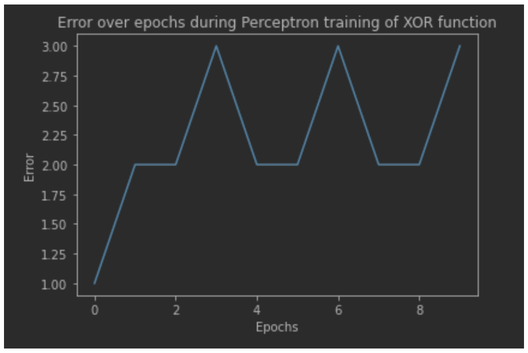


(b) Error over 10 epochs for OR gate

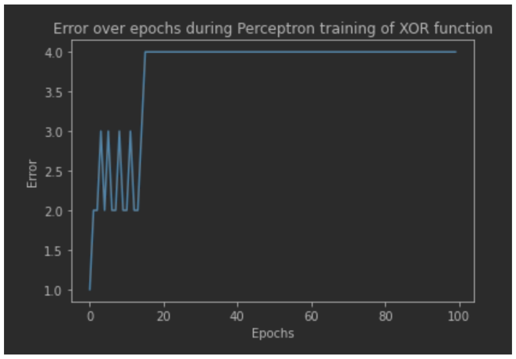
Figure 1: Error over 10 epochs for AND and OR gates

As we can see from the graphs above, the perceptron is able to learn both the AND and OR gates, as the error after a certain number of epochs is constantly zero.

However, for the XOR gate this doesn't seem to be the case. The error after ten epochs is still not zero and, to prove that the gate can't be learned even if I add more epochs, I run the same simulation with 100 epochs. After a certain number of epochs, the error seems to constantly stay at four, which means the perceptron is not able to learn the XOR gate. This is expected, as the XOR gate needs non-linear separability to correctly classify the points, and the perceptron is linear.



(a) Error over 10 epochs for XOR gate



(b) Error over 100 epochs for XOR gate

Figure 2: Error over 10 and 100 epochs for XOR gate

Before looking at the solution, let's summarize the architecture of the single layer perceptron:

1. As the input has 10 features, the input layer should consist of 10 neurons.
2. There are seven possible classes (i.e. outputs), so I will need seven neurons in the output layer.

2.2 A multilayer perceptron

The nonlinearity problem can be fixed by using a multilayer perceptron.

The network architecture comprises three layers: the input layer, one hidden layer, and the output layer. Based on empirical evidence, it was determined that the inclusion of additional hidden layers would not yield substantial performance improvements [3]. I will begin by using the mean number of hidden neurons, which is calculated as follows: $int((7+10)\ /\ 2) = 8$. Given the simplicity of the network, I will only utilize one hidden layer (for a total of three layers). This will be sufficient, as further layers are unlikely to provide substantial improvements.

The Rectified Linear Unit (ReLU) activation function will be used in the hidden layers, as it enables more efficient computation, resulting in faster convergence and improved optimization using Stochastic Gradient Descent.

The Softmax function is the activation function used in the output layer, due to its ability to facilitate multi-class classification, required for the 7 possible output classes. Furthermore, it is highly compatible with the cross-entropy loss function, which I will make use of. Implementation of the loss function is based on [5].

To sum up, a schematic diagram of the complete network is shown in Figure 3.

3 Training

Prior to the training of the neural network, the data is randomly shuffled and split into three separate sets: a training set, a validation set, and a test set, comprising 70%, 15%, and 15% of the total dataset, respectively. The network training is done on the training set. In every epoch checkpoint during training, the validation set is utilized to evaluate performance, and parameter adjustment is done based on the optimal outcome.

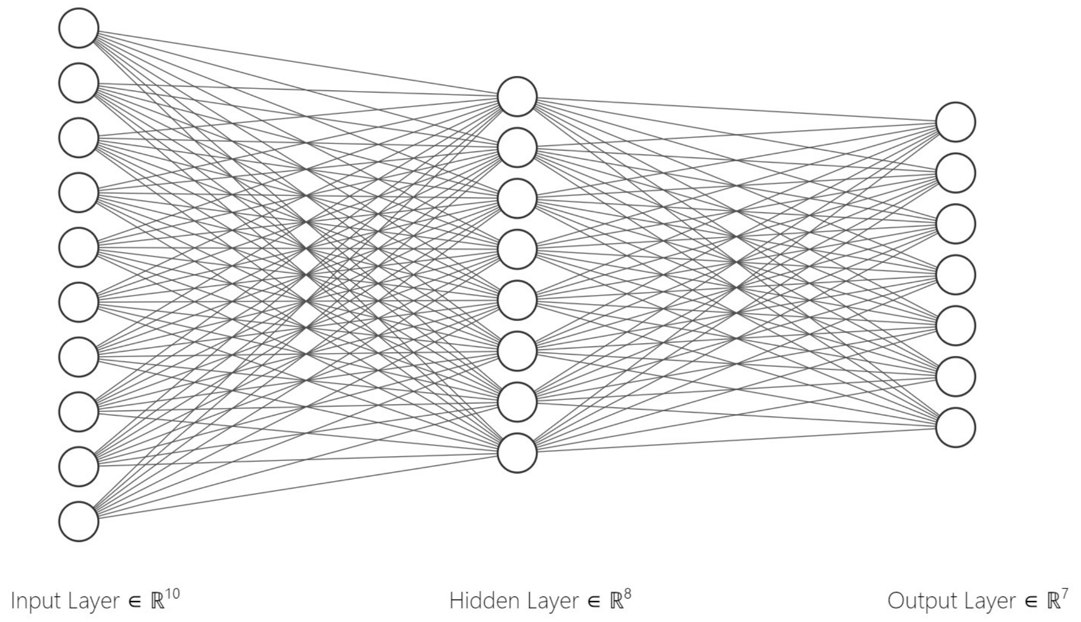


Figure 3: Network visualization diagram

To ensure unbiased results, the final execution of the network is evaluated solely on the test set, composed of unseen data to the network and resulting on neutral performance assessment.

3.1 Evaluating the performance of the network

Network performance is evaluated via a graph plotting the accuracy of the validation set. This allows to visualize results using different parameter values and to choose the network architecture which will optimize the classification outcome. Furthermore, accuracy reflects the proportion of correctly classified inputs, thereby providing a direct measure of performance evaluation.
Moreover, I plot the graphs of the loss for both the training and validation sets across epochs, comparing them to avoid under-fitting or over-fitting. The loss function used is cross-entropy; because it can be applied to multi-class classification and helps efficiently compute the gradient during backpropagation. In addition, to reduce over-fitting, I introduce L^2 regularization [2], [4]. This technique incorporates a penalty term into the loss function that is proportional to the square of the weights, thereby driving the weights towards zero. The results demonstrate that this combination improves the overall performance of my model.

3.2 Ending the training

Given the limitations of computational resources and the time-consuming nature of training neural networks, I employ early stopping as a strategy to mitigate these challenges. Specifically, I monitor the performance of the validation set across epochs and halt the training process when the performance plateaus for a certain number of epochs. This approach not only saves computational resources, but also prevents over-fitting. My implementation is based on [1]. Based on the analysis of the graphs of the loss for both the training and validation sets across epochs, I determined that the optimal performance is achieved after approximately 30 epochs.

3.3 Evaluating the importance of the initial weights

Initializing the weights in different ways can affect the performance of the network and, therefore, the accuracy of results. This can be observed in **Figure 4** representing 10 different random weights initialization using He Initialization formula. It can be concluded that depending on how weights are first created, accuracy can range from around 0.93 to 0.87.

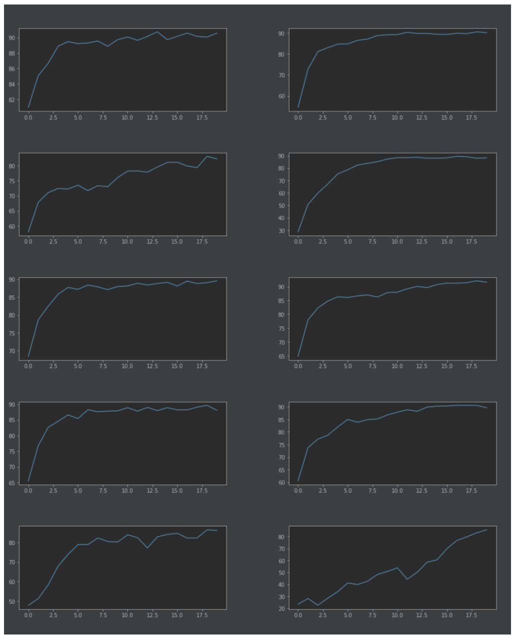


Figure 4: 10 graphs of accuracy against weights initialization

4 Optimization

4.1 The amount of hidden neurons

In **Figure 5**, I use 10-fold cross-validation to observe how the number of hidden neurons affects the training behaviour and the performance of the neural network. I trained a network with 4, 7, 8, 9, 10, 11, 15, and 30 neurons and analysed the resulting accuracy on both the training and validation sets. My findings indicate that increasing the number of neurons in the hidden layer leads to higher accuracy. However, this increased accuracy does not necessarily correspond to improved performance and can result in over-fitting of the model. Furthermore, adding more neurons to the network increases its complexity, which can negatively impact its performance. To determine the optimal number of neurons in the hidden layer, I employ the "elbow method." Through this method, I determined that the optimal number of neurons is nine.

4.2 The best architecture

It has already been decided that the network will use 9 neurons in the hidden layer, but how about the learning rate and the regularization parameters? To find the best values for those hyperparameters, again, 10-fold cross-validation is used.

In **Figure 6**, I experiment with different values for the learning rate, 0.001, 0.01, 0.05, 0.1 and 0.15, as well as for the regularization parameter, 0, 0.001, 0.01, and 0.1. My assessment involves the plotting of the performance of the network over 100 epochs. Through this process, I identified the best learning rate to be 0.01, and the optimal regularization parameter to be 0.001.

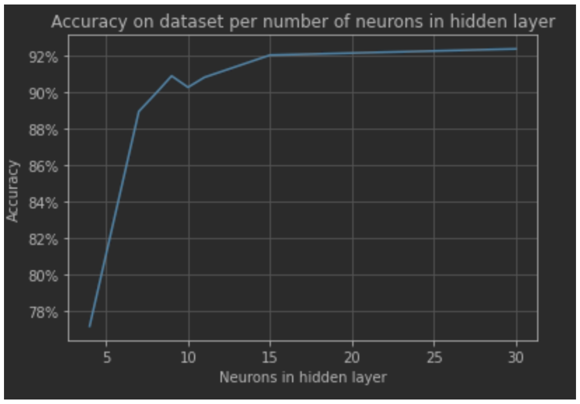
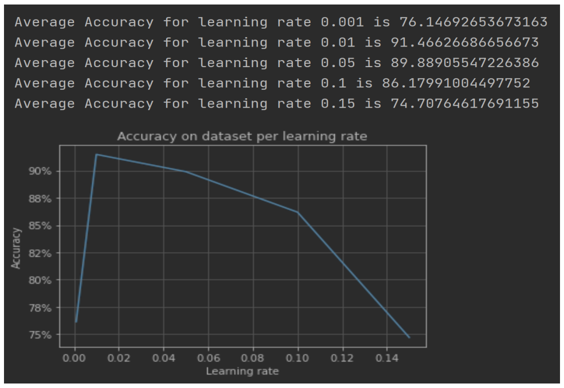


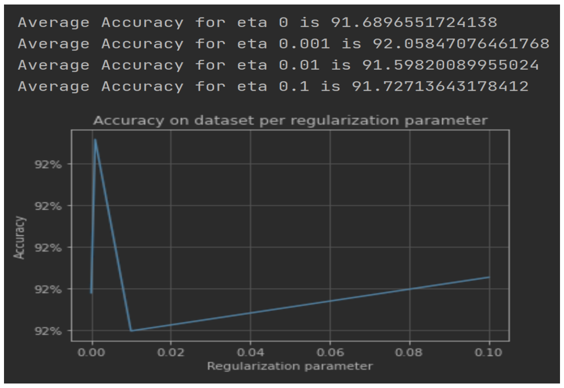
Figure 5: Final performance versus the number of hidden neurons

However, due to the utilization of Adam algorithm, I selected a learning rate of 0.001, as suggested by [2]. This choice is better for the plotting of the performance on the training and validation sets, since in Epoch 1 we start at a lower accuracy and the curve is more feasible. Nonetheless, both 0.01 and 0.001 converge in a similar fashion, given a sufficient number of epochs, thereby resulting in comparable performance. My results underscore the importance of carefully selecting hyperparameters to optimize the neural network's performance.



```
Average Accuracy for learning rate 0.001 is 76.14692653673163
Average Accuracy for learning rate 0.01 is 91.46626686656673
Average Accuracy for learning rate 0.05 is 89.88905547226386
Average Accuracy for learning rate 0.1 is 86.17991004497752
Average Accuracy for learning rate 0.15 is 74.70764617691155
```

(a) The learning rate



```
Average Accuracy for eta 0 is 91.6896551724138
Average Accuracy for eta 0.001 is 92.05847076461768
Average Accuracy for eta 0.01 is 91.59820089955024
Average Accuracy for eta 0.1 is 91.72713643178412
```

(b) The regularization parameter

Figure 6: Final performance versus learning rate and regularization parameter

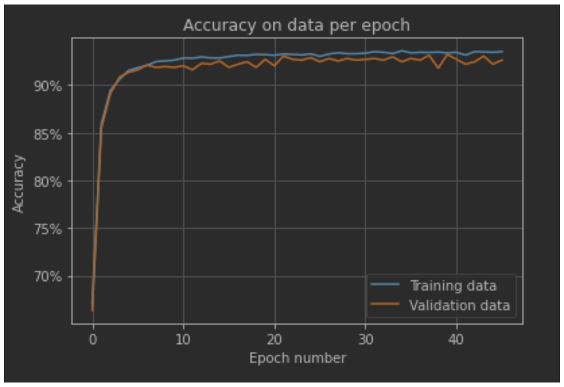
Now that I am confident that all the optimal hyperparameters were picked, I plot in **Figure 7** the performance of the network on the training set and validation set across 100 epochs. I show both the accuracy and loss on each set.

The accuracy of the validation set is very similar to the one of the training set, so the neural network seems to predict new data with a high accuracy. The graphs of the losses also confirm that we are neither under-fitting nor over-fitting while training. Thus, this combination of hyperparameters proves to be optimal for my network.

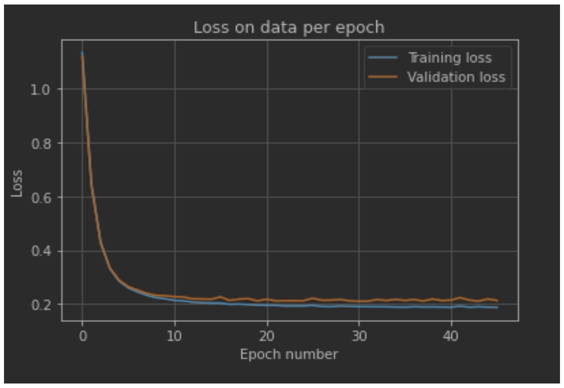
5 Evaluation

On average, the success rate of the **test set** is 93%, while the success rate of the **validation set** is approximately 92%.

For evaluation, it is a better idea to discuss the confusion matrix constructed from the test set, shown in Figure 8.

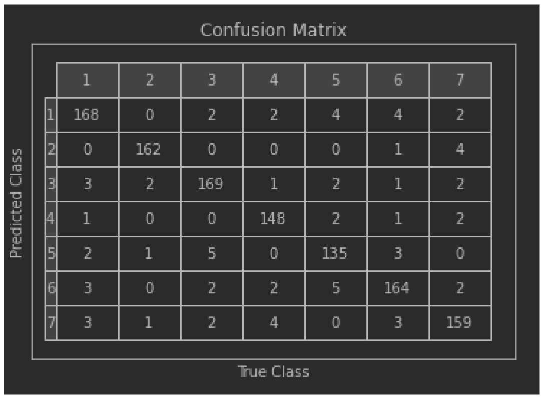


(a) Accuracy on the training set and validation set



(b) Loss on the training set and validation set

Figure 7: Accuracy and loss on the training set and validation set across 100 epochs


Confusion Matrix
Predicted Class
True Class

	1	2	3	4	5	6	7
1	168	0	2	2	4	4	2
2	0	162	0	0	0	1	4
3	3	2	169	1	2	1	2
4	1	0	0	148	2	1	2
5	2	1	5	0	135	3	0
6	3	0	2	2	5	164	2
7	3	1	2	4	0	3	159

Figure 8: Confusion Matrix

The rows of the confusion matrix in **Figure 8** indicate the class the neural network predicts, while the columns show the correct labels. Thus, the diagonal of the matrix represents the correctly classified elements. For example, on the top-left cell (Predicted Class = 1, True Class = 1), the network correctly classifies 168 values for class 1. It can also be derived that the network incorrectly classifies 3 values as class 3 instead of class 1 (Predicted Class = 3, True Class = 1).
The network makes the most mistakes by incorrectly classifying elements that belong to class 6. In total, 13 elements were incorrectly classified for this class. Specifically, it made 4 mistakes by predicting elements of class 6 as class 1.

Finally, I fed the unknown set to the network, and the comma-separated file with the resulting classes is named **"05_classes.txt"**.

6 Scikit-learn

I will now compare my network to the Scikit alternative. When using the grid search provided by Scikit, the following parameters are chosen as optimal for the neural network:

1. *Activation function*: ReLU. I chose the same activation function.
2. *Regularization parameter*: 0.01. This value differs from my choice of 0.001.
3. *Early stopping*: True, because I also use Early Stopping.
4. *Hidden Layer Size*: 30. Scikit chooses the highest number as this yields the largest accuracy. On the other hand, I chose the number of hidden layers with which the network performs best without severely increasing computational time, which is 9.
5. *Learning rate*: 0.01. I also found the same optimal learning rate, but I decided to use 0.001 as explained previously in Section 4.2.
6. *Solver*: Adam. I also chose Adam.

The implementation of Adam algorithm is based on [6] and algorithm 8.7 of [2].

The efficacy of the hyperparameters selected by Scikit varies, owing to the random initialization of weights and biases, leading to either performance improvement or decline. Overall, their success rate seems to be in the 90-92% range.

My belief is that their chosen hyperparameters are comparable in effectiveness to mine. Firstly, the different regularization parameters do not significantly impact the performance. Secondly, the sole point of disagreement lies in the selection of the number of hidden neurons, where Scikit invariably chooses the maximum number to maximize accuracy, whereas I take into account network performance, which tends to deteriorate upon the addition of more hidden neurons.

References

[1] Scikit-Learn Documentation. Early stopping of Stochastic Gradient Descent . https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_early_stopping.html. [Online; accessed 01-March-2023].

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[3] Rob Hyndman. CrossValidated Forum Post. https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw, 2011. [Online; accessed 20-February-2023].

[4] Andrew Ng, Moses Charikar, and Carlos Guestrin. *CS229: Machine Learning Lecture Notes*. Stanford University, 2022.

[5] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[6] Stanford University. CS231n Lecture Notes. https://cs231n.github.io/neural-networks-3/#ada, 2022. [Online; accessed 25-February-2023].