Grado en Ingeniería Informática

Lab assignment 2: Multilayer perceptron for classification problems

Universidad de Córdoba

Fourth year of "Grado en Ingeniería Informática"
Dpto. de Informática y análisis Numérico
Introduction to computational models
Course 2023-2024

Autor:

Antonio Llamas García i92llgaa@uco.es
31886320V

Docentes:

Pedro A. Gutiérrez Peña	pagutierrez@uco.es
Javier Sánchez Monedero	jsanchezm@uco.es
César Hervás Martínez	chervas@uco.es

Córdoba, 5 de febrero de 2024

Índice

1. Description of Neural Network Models

In the initial phase of our experimentation, we conducted the first experiment to identify optimal parameters for subsequent experiments. This involved testing various neural network architectures on different datasets. The experiment is divided into two parts:

1.1. Experiment 1

The primary objective of Experiment 1 is to identify the most effective parameters, which will be subsequently employed in the following experiments. The results and metrics are stored in the `results1.out` file for further analysis.

1.1.1. XOR Problem:

- **Architecture:** Utilizing the architecture that exhibited the best performance in the previous lab assignment.
- **Common Parameters:**
  - Learning Rate (η): 0.7
  - Momentum (μ): 1.0
  - Maximum Iterations: 1000
  - Training Algorithm: Offline (Batch)

1.1.2. ProPublica COMPAS and noMNIST Datasets:

Testing eight distinct architectures, each with one or two hidden layers having 4, 8, 16, or 64 neurons.

- **Common Parameters:**
  - Learning Rate (η): 0.7
  - Momentum (μ): 1.0
  - Maximum Iterations: 1000
  - Training Algorithm: Offline (Batch)
  - Error Function: Cross-Entropy
  - Output Layer Activation: Softmax

1.2. Experiment 2

For this experiment, we have used the best configuration obtained in the first experiment for each dataset, and we have done some experiments in them:

- MSE error function and sigmoidal activation function in the output layer.
- MSE error function and softmax activation function in the output layer.
- Cross-entropy error function and softmax activation function in the output layer.

1.3. Experiment 3

For this experiment, we have used the best the best configuration obtained in the second experiment for each dataset, and we have trained them with the online version.

2. Algorithms description

In this section, we will learn about the most important functions/algorithms of our program. We are going to know how they work and why are they so important.

2.1. ForwardPropagation Algorithm

The forwardPropagate() function performs forward propagation in a Multilayer Perceptron (MLP). It calculates the output of each neuron in the network based on the input data and learned weights. The activation function used depends on whether the current layer is the output layer or a hidden layer.

Algorithm 1 Forward Propagate

```
for $i \leftarrow 1$ to $nOfLayers - 1$ do    ▷ For each layer
    for each neuron $j$ in the current layer ($this \rightarrow layers[i]$) do
        $net \leftarrow 0{,}0$    ▷ Initialize the weighted sum
        for $k \leftarrow 1$ to $layers[i-1].nOfNeurons$ do    ▷ For each input connection
            $net \mathrel{+}= this \rightarrow layers[i].neurons[j].w[k] \cdot this \rightarrow layers[i-1].neurons[k-1].out$    ▷ Calculate the weighted sum
        end for
        $net \mathrel{+}= this \rightarrow layers[i].neurons[j].w[0]$    ▷ Add the bias term
        if $(i = nOfLayers - 1)$ and $(outputFunction = 1)$ then
            $this \rightarrow layers[i].neurons[j].out \leftarrow \exp(net)$    ▷ Softmax for output layer
            $sumNet \mathrel{+}= \exp(net)$    ▷ Sum for Softmax normalization
        else
            $this \rightarrow layers[i].neurons[j].out \leftarrow \frac{1}{1+\exp(-net)}$    ▷ Sigmoid for hidden layers
        end if
    end for
    if $(i = nOfLayers - 1)$ and $(outputFunction = 1)$ then
        for each neuron $j$ in the output layer do
            $this \rightarrow layers[i].neurons[j].out \mathrel{/}= sumNet$    ▷ Softmax normalization
        end for
    end if
end for
```

2.2. Obtain Error Function

The obtainError() function in the Multilayer Perceptron (MLP) calculates the error between the network's output and the target values, based on the specified error function. It supports Mean Squared Error (MSE) and Cross-Entropy (CE) as two different error functions.

2.3. BackPropagation Algorithm

The backpropagateError() function in the Multilayer Perceptron (MLP) is responsible for updating the delta values for each neuron during the backpropagation phase. It considers different cases for the output layer activation function, Softmax or Sigmoid, and the specified error function (Mean Squared Error or Cross-Entropy).

2.4. Weight Adjustment Function

The weightAdjustment() function in a Multilayer Perceptron (MLP) is responsible for updating the weights of the neurons based on the calculated weight changes (deltaW). It incorporates both online and offline learning modes, allowing for updates after each training pattern in online mode or after processing all training patterns in offline mode. The function uses the Backpropagation through Time (BPTT) algorithm for weight adjustment, including momentum for faster convergence.

Algorithm 2 Obtain Error

```
Require: $target$: Array representing the desired output for the current input pattern
Require: $errorFunction$: Integer specifying the error function to use (0 for MSE, 1 for Cross-Entropy)
 1: if $errorFunction = 0$ then
 2:     $mse \leftarrow 0{,}0$
 3:     for $i \leftarrow 0$ to $layers[nOfLayers-1].nOfNeurons-1$ do
 4:         $mse \mathrel{+}= (target[i] - layers[nOfLayers-1].neurons[i].out)^2$   ▷ Calculate the squared error
 5:     end for
 6:     return $mse/layers[nOfLayers-1].nOfNeurons$   ▷ Calculate the mean squared error
 7: else
 8:     $ce \leftarrow 0{,}0$
 9:     for $i \leftarrow 0$ to $layers[nOfLayers-1].nOfNeurons-1$ do
10:         $ce \mathrel{+}= target[i] \cdot \log(layers[nOfLayers-1].neurons[i].out)$   ▷ Calculate the cross-entropy error
11:     end for
12:     return $-ce/layers[nOfLayers-1].nOfNeurons$   ▷ Calculate the mean cross-entropy error
13: end if
```

Algorithm 3 Backpropagate Error

```
Require: $target$: Array representing the desired output for the current input pattern
Require: $errorFunction$: Integer specifying the error function to use (0 for MSE, 1 for Cross-Entropy)
 1: for $i \leftarrow 0$ to $layers[nOfLayers-1].nOfNeurons-1$ do
 2:     $out \leftarrow layers[nOfLayers-1].neurons[i].out$
 3:     $layers[nOfLayers-1].neurons[i].delta \leftarrow 0{,}0$
 4:     if $outputFunction = 1$ then   ▷ Softmax activation in the output layer
 5:         for $j \leftarrow 0$ to $layers[nOfLayers-1].nOfNeurons-1$ do
 6:             $conditionSoftmax \leftarrow (i == j)$
 7:             if $errorFunction = 0$ then
 8:                 $layers[nOfLayers-1].neurons[i].delta \mathrel{+}= -(target[j] - layers[nOfLayers-1].neurons[j].out) \cdot out \cdot (conditionSoftmax - layers[nOfLayers-1].neurons[j].out)$
 9:             else
10:                 $layers[nOfLayers-1].neurons[i].delta \mathrel{+}= -(target[j]/layers[nOfLayers-1].neurons[j].out) \cdot out \cdot (conditionSoftmax - layers[nOfLayers-1].neurons[j].out)$
11:             end if
12:         end for
13:     else   ▷ Sigmoid activation in the output layer
14:         if $errorFunction = 0$ then
15:             $layers[nOfLayers-1].neurons[i].delta \leftarrow -(target[i] - out) \cdot out \cdot (1 - out)$
16:         else
17:             $layers[nOfLayers-1].neurons[i].delta \leftarrow -(target[i]/out) \cdot out \cdot (1 - out)$
18:         end if
19:     end if
20: end for
21: for $i \leftarrow nOfLayers-2$ down to $1$ do
22:     for $j \leftarrow 0$ to $layers[i].nOfNeurons-1$ do
23:         $out \leftarrow layers[i].neurons[j].out$
24:         $aux \leftarrow 0{,}0$
25:         for $k \leftarrow 0$ to $layers[i+1].nOfNeurons-1$ do
26:             $aux \mathrel{+}= layers[i+1].neurons[k].w[j+1] \cdot layers[i+1].neurons[k].delta$
27:         end for
28:         $layers[i].neurons[j].delta \leftarrow aux \cdot out \cdot (1 - out)$
29:     end for
30: end for
```

Algorithm 4 Weight Adjustment

```
1: if online = true then                                                    ▷ Online mode
2:     for $i \leftarrow 1$ to $nOfLayers - 1$ do                             ▷ For every layer
3:         for $j \leftarrow 1$ to $layers[i].nOfNeurons$ do                  ▷ For every neuron
4:             for $k \leftarrow 1$ to $layers[i-1].nOfNeurons + 1$ do         ▷ For every weight
5:                 $layers[i].neurons[j].w[k] \mathrel{-}= (\eta \cdot layers[i].neurons[j].deltaW[k]) - (\mu \cdot \eta \cdot layers[i].neurons[j].lastDeltaW[k])$   ▷ Update the weight
6:             end for
7:             $layers[i].neurons[j].w[0] \mathrel{-}= (\eta \cdot layers[i].neurons[j].deltaW[0]) - (\mu \cdot \eta \cdot layers[i].neurons[j].lastDeltaW[0])$   ▷ Update the bias
8:         end for
9:     end for
10: else                                                                    ▷ Offline mode
11:     for $h \leftarrow 1$ to $nOfLayers - 1$ do                            ▷ For every layer
12:         for $j \leftarrow 1$ to $layers[h].nOfNeurons$ do                 ▷ For every neuron
13:             for $i \leftarrow 1$ to $layers[h-1].nOfNeurons + 1$ do        ▷ For every weight
14:                 $layers[h].neurons[j].w[i] \mathrel{-}= (\eta \cdot layers[h].neurons[j].deltaW[i]/nOfTrainingPatterns) - (\mu \cdot \eta \cdot layers[h].neurons[j].lastDeltaW[i]/nOfTrainingPatterns)$   ▷ Update the weight
15:             end for
16:             $layers[h].neurons[j].w[0] \mathrel{-}= (\eta \cdot layers[h].neurons[j].deltaW[0]/nOfTrainingPatterns) - (\mu \cdot \eta \cdot layers[h].neurons[j].lastDeltaW[0]/nOfTrainingPatterns)$   ▷ Update the bias
17:         end for
18:     end for
19: end if
```

2.5. Perform Epoch Algorithm

The performEpoch() function in a Multilayer Perceptron (MLP) is designed to execute one epoch of the training process. An epoch involves processing each training pattern through the network, computing the error, and adjusting the weights based on the backpropagation algorithm. The function accommodates both online and offline learning modes, allowing for weight updates after each training pattern in online mode or after processing all patterns in offline mode.

Algorithm 5 Perform Epoch

```
1: if online = 1 then                                                       ▷ Online mode
2:     for $i \leftarrow 1$ to $nOfLayers - 1$ do
3:         for $j \leftarrow 0$ to $layers[i].nOfNeurons$ do
4:             for $k \leftarrow 0$ to $layers[i-1].nOfNeurons + 1$ do
5:                 $layers[i].neurons[j].deltaW[k] \leftarrow 0{,}0$                 ▷ Reset deltaW
6:             end for
7:         end for
8:     end for
9: end if
10: feedInputs($input$)
11: forwardPropagate()
12: backpropagateError($target, errorFunction$)
13: accumulateChange()
14: if online = 1 then                                                      ▷ Online mode
15:     weightAdjustment()
16: end if
```

3. Experiments

In this section, we will see the datasets and parameters that we have used, and the results obtained with a discussion about them.

3.1. Datasets Description

In this section, we provide an overview of the datasets utilized to assess the performance of the implemented algorithm. Three distinct datasets, each representing a unique problem, have been selected for evaluation.

3.1.1. XOR Problem

The XOR problem dataset is designed to address the non-linear classification challenge posed by the XOR logical operation. The same file is used for both training and testing. Notably, the dataset has been adapted to a 1-to-k codification, resulting in two outputs instead of one.

3.1.2. ProPublica COMPAS Dataset

The ProPublica COMPAS dataset focuses on evaluating the performance of the COMPAS algorithm, a statistical method employed within the United States criminal justice system for risk assessment. Originally published by ProPublica in 2016, the dataset investigates the risk scores assigned by COMPAS for the risk of recidivism..^{Ea}ch individual in the dataset is associated with a binary outcome, indicating whether they were rearrested within two years after the initial arrest. The attributes have been reduced from 52 to 9, including variables such as sex, age, race, juvenile felony count, juvenile misdemeanor count, juvenile other count, priors count, and charge degree. The prediction variable is whether the individual will be rearrested in two years or not.

3.1.3. noMNIST Dataset

The noMNIST dataset was initially composed of 200,000 training patterns and 10,000 test patterns, featuring 10 classes of letters (from a to f). However, for the purpose of this lab assignment, the dataset size has been reduced to 900 training patterns and 300 test patterns to manage computational costs. The dataset includes letters written in various typologies or symbols, adjusted to a squared grid of 28 $\times$ 28 pixels. The images are grayscale, ranging from -1.0 to +1.0. Each pixel serves as an input variable, resulting in a total of 784 input variables. The classes correspond to the written letters (a, b, c, d, e, and f).

3.2. Parameter Values

In our experiments, various parameters play a crucial role in configuring the neural network models. Below, we outline the different parameters along with their default values:

- **Argument `t`:** Indicates the name of the file containing the training data. This argument is mandatory, and the program cannot function without it.
- **Argument `T`:** Indicates the name of the file containing the testing data. If not specified, the training data will be used for testing as well.
- **Argument `i`:** Specifies the number of iterations for the outer loop. If not specified, the default is 1000 iterations.
- **Argument `l`:** Specifies the number of hidden layers in the neural network. If not specified, the default is 1 hidden layer.
- **Argument `h`:** Specifies the number of neurons in each hidden layer. If not specified, the default is 5 neurons.
- **Argument `e`:** Specifies the value for the eta (η) parameter. The default is $\eta = 0{,}1$.
- **Argument `m`:** Specifies the value for the mu (μ) parameter. The default is $\mu = 0{,}9$.

- **Argument `o`:** A Boolean indicating whether the on-line version is applied. By default, the off-line version is used.
- **Argument `f`:** Specifies the error function to be used (0 for MSE and 1 for cross-entropy). The default is MSE.
- **Argument `s`:** A Boolean indicating whether the softmax function is used for the output layer. By default, the sigmoidal function is used.
- **Argument `n`:** A Boolean indicating that the data (training and test) will be normalized after reading. Inputs will be normalized to the interval $[-1, 1]$, and outputs are not normalized in classification.

These parameters provide flexibility and control over the neural network configurations during the experiments.

3.3. Experiment 1

This section is thought with the purpose of show the results obtain in each experiment done with each dataset about the first experiment.

3.3.1. Results for XOR Dataset

First, we have the XOR dataset. This dataset is the simplest of the 3 used, in addition to being the one for which the fewest implementations have been asked of us. For this set of data, two executions have been carried out, the first with the best version obtained in the previous practice, but with the values of the eta and mu variables adapted to the requirements of the practice. The second test has been carried out with the values of the best version obtained in the previous practice without any modification.

Run 1:

- Hidden Layers (l): 2
- Neurons per Hidden Layer (h): 100
- Learning Rate (e): 0.7
- Momentum (m): 1.0
- Error Function (f): 1 (cross-entropy)
- Activation Function in Output Layer (s): true
- Data Normalization (n): false
- Online Version (o): false

Run 2:

- Hidden Layers (l): 2
- Neurons per Hidden Layer (h): 100
- Learning Rate (e): 0.1
- Momentum (m): 0.9
- Error Function (f): 1 (cross-entropy)
- Activation Function in Output Layer (s): true
- Data Normalization (n): false
- Online Version (o): false

Figura 1: Parameter Configurations for XOR Dataset

Run	**Train MSE**	**Test MSE**	**Train CCR (%)**	**Test CCR (%)**	**Execution Time (s)**
1	0.000599249 +- 5.03008e-05	0.000599249 +- 5.03008e-05	100 +- 0	100 +- 0	4.226142
2	0.00548247 +- 0.000596968	0.00548247 +- 0.000596968	100 +- 0	100 +- 0	4.129812

Tabla 1: Performance Metrics for XOR Dataset

As we can see, both executions have a very similar time, but the algorithm performs better with the suggested data as input, since it obtains a much lower error and standard deviation than the default configuration of the previous practice.

A lower error suggests that the predictions made are more accurate and the model fits the training data well. Furthermore, the low value of the standard deviation suggests that the errors are close and introduce less variability into the model.

There is not much to talk about the CCR obtained because in both cases it is 100 %, which indicates that the model correctly learns to classify in both cases. This is mainly because the test set used is the same as the training set, and because this set is very small in size.

3.3.2. Results for ProPublica COMPAS Dataset

This is the second largest dataset. For it, we have used the configurations that were required of us in the lab assignment. There are two main architectures, one with 1 hidden layer and another with 2 hidden layers, for which the number of layer neurons is varied, taking the values 4, 8, 16 and 64. For these configurations, the data obtained is shown and subsequently analyzed.

Run 1:

- Hidden layers: 1
- Neurons in hidden layer: 4, 8, 16, 64
- Learning rate (eta): 0.7
- Momentum (mu): 1.0
- Error function (f): 1 (cross-entropy)
- Activation function in output layer (s): true (softmax)
- Data normalization (n): false
- Execution mode (o): false (offline)

Run 2:

- Hidden layers: 2
- Neurons in hidden layer: 4, 8, 16, 64
- Learning rate (eta): 0.7
- Momentum (mu): 1.0
- Error function (f): 1 (cross-entropy)
- Activation function in output layer (s): true (softmax)
- Data normalization (n): false
- Execution mode (o): false (offline)

Figura 2: Parameter Configurations for ProPublica COMPAS Dataset

N	**Train MSE**	**Test MSE**	**Train CCR (%)**	**Test CCR (%)**	**Execution Time (s)**
4	0.300841 +- 0.00126283	0.306178 +- 0.00121779	100 +- 0	99.9446 +- 0	20.135917
8	0.299046 +- 0.000441879	0.30598 +- 0.000724105	100 +- 0	99.8448 +- 0.199557	36.797665
16	0.296824 +- 0.000508963	0.305179 +- 0.000309314	100 +- 0	99.9446 +- 0	65.819942
64	0.313066 +- 0.0357167	0.326045 +- 0.0323222	99.9951 +- 0.0098595	99.9446 +- 0	199.429132

Tabla 2: Performance Metrics for ProPublica COMPAS Dataset with 1 hidden layer

N	**Train MSE**	**Test MSE**	**Train CCR (%)**	**Test CCR (%)**	**Execution Time (s)**
4	0.30057 +- 0.000417457	0.306172 +- 0.000545525	100 +- 0	99.9446 +- 0	29.707909
8	0.299767 +- 0.000877069	0.306011 +- 0.000880332	100 +- 0	99.9446 +- 0	64.751357
16	0.297814 +- 0.00102205	0.305392 +- 0.000404637	100 +- 0	99.8448 +- 0.199557	154.644078
64	0.297824 +- 0.00350595	0.310618 +- 0.00269111	100 +- 0	99.9446 +- 0	1228.244275

Tabla 3: Performance Metrics for ProPublica COMPAS Dataset with 2 hidden layers

By analyzing the data obtained after the executions of the different required configurations, we can draw several general conclusions about the behavior of the algorithm on this data set.

Firstly, we can observe that there is hardly any tendency for the MSE to decrease/grow as the number of neurons in the hidden layers increases. In fact, in the models with more neurons a non-significant increase in the MSE can be observed, which may suggest that the models with more neurons do not necessarily fit better to these training and test data.

We can also see how the CCR for both configurations in the training set is always 100 %, while for the test sets it sometimes decreases with the increase in neurons, so it could indicate an overfitting to the training data. Even so, the decrease is minimal, and analyzing the confusion matrices of the seeds, it may seem that the error/errors committed in the classification are minimal and always similar or equal, which strengthens this aforementioned hypothesis.

Another metric to take into account is the variance of each result. The variance tells us the consistency of the results throughout the executions. It stands out that for simpler configurations the variance is similar or even lower than for more complex configurations, which tells us that these implementations show results with greater consistency than complex ones, although the difference is minimal.

And lastly, it is worth mentioning the execution time of each configuration, which, as expected, increases significantly as the number of neurons increases. A better number of neurons implies a better number of operations and therefore a better training time.

Once the general behaviors of the algorithm applied to this particular data set have been shown, the user must choose the configuration that they consider optimal, adapted to the balance between the performance of the model and the computational efficiency that they desire.

We should also note that these conclusions are based on an initial observation of the results and that more detailed analysis, such as cross-validation and hyperparameter tuning, could provide a more complete and robust understanding of model performance.

Even so, we will go a little deeper into the results we have obtained so far:

As we have mentioned before, in most of the configurations used for this dataset, a greater number of neurons does not imply obtaining better results, but it does greatly increase the execution time and computational cost.

That is why, after analyzing all the metrics obtained, it is easy to come to the conclusion that the best configurations obtained are those with 4 neurons per hidden layer, giving very good results, being similar or better than the rest of the configurations.

Going deeper into these two configurations, both obtain very similar MSE and CCR results, although it is true that the one that contains 2 hidden layers shows a lower variance, which implies that it offers more consistent results, although the difference is very small. It is also true that this small improvement requires 1/2 more of the required execution time, but since the times are small, this increase is not very large either.

3.3.3. Results for noMNIST Dataset

This is the largest of the datasets, for which the same configurations have been made as for the previous dataset, in order to be able to see and compare the behavior of the algorithm under the same conditions but different datasets.

Run 1:

- Hidden layers: 1
- Neurons in hidden layer: 4, 8, 16, 64
- Learning rate (eta): 0.7
- Momentum (mu): 1.0
- Error function (f): 1 (cross-entropy)
- Activation function in output layer (s): true (softmax)
- Data normalization (n): false
- Execution mode (o): false (offline)

Run 2:

- Hidden layers: 2
- Neurons in hidden layer: 4, 8, 16, 64
- Learning rate (eta): 0.7
- Momentum (mu): 1.0
- Error function (f): 1 (cross-entropy)
- Activation function in output layer (s): true (softmax)
- Data normalization (n): false
- Execution mode (o): false (offline)

Figura 3: Parameter Configurations for noMNIST Dataset

N	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
4	0.119049 +- 0.00456289	0.152547 +- 0.00656416	69.6667 +- 3.38333	62.3333 +- 2.59058	180.291908
8	0.0471212 +- 0.003087	0.0965052 +- 0.010324	80.3333 +- 1.79574	70.8 +- 3.40327	336.421636
16	0.0275593 +- 0.00257001	0.103177 +- 0.00493731	82.3778 +- 1.02463	71 +- 1.92065	647.377504
64	0.00363359 +- 0.000402459	0.111041 +- 0.00421485	100 +- 0	72.8 +- 0.541603	2484.416950

Tabla 4: Performance Metrics for noMNIST Dataset with 1 hidden layer

N	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
4	0.153974 +- 0.0114853	0.17774 +- 0.0152266	60.7111 +- 7.34363	52.7333 +- 8.24729	180.455714
8	0.0576614 +- 0.00577476	0.110856 +- 0.012179	78.3333 +- 1.44188	69.3333 +- 1.38243	340.460270
16	0.0216922 +- 0.00167471	0.107539 +- 0.00294805	85.4667 +- 2.08629	71 +- 1.36626	641.561138
64	0.0022593 +- 0.000128988	0.116318 +- 0.0143732	100 +- 0	71.2667 +- 1.42049	2824.103310

Tabla 5: Performance Metrics for noMNIST Dataset with 2 hidden layers

Once the corresponding metrics have been obtained, we move on to analyze these results. As before, first we can extract some general behaviors of the algorithm against this dataset, and then delve into some particularities.

In both tables it is seen how the MSE decreases as we increase the number of neurons in the algorithm, especially in the training set. This metric suggests that models with more neurons fit the training and test data better.

Furthermore, we can see an upward trend in the CCR for both sets in the same case, that is, as the number of neurons increases. The results obtained are good, but it would be necessary to study whether the decrease, in this metric, of the test sets compared to the training sets, is due to overfitting of the algorithm with the training data.

Regarding the standard deviation of each measurement, it can be seen how it decreases, for both measurements and sets, as the number of neurons in the hidden layers of the models increases. Therefore, we can affirm that as the number of neurons in the models increases, the consistency of the results obtained by this model increases.

As expected, although the results increase as the neurons in the models increase, the execution time and computational cost also increase considerably, something to take into account.

Once all this data is known, the user is responsible for choosing the configuration that best suits the requirements necessary for their project.

Finally, we will analyze some notable particular cases:

First of all, it should be noted that the models implemented with 64 neurons, in both cases, obtain very good metrics for the training sets, although subsequently the data from the test set do not differ much from the model implemented with 16 neurons. This could be a clear example that from a certain number of neurons onwards the algorithm tends to overfit with the training data, which is why it is subsequently not able to improve in the classification of another set.

It should also be said that for these latest models, the execution time increases greatly, so perhaps it would not be the best in the relationship between results and execution time.

For all this, and after analyzing all these patterns, we could conclude that the models that may be most interesting to us are those implemented with 16 neurons per hidden layer, and between the two with these characteristics, the one that has 2 hidden layers, due to Because, although the error and success metrics are similar, the standard deviations of these are somewhat less in the more complex case, so we can ensure that we obtain more consistent results than with the simplest model.

3.4. Experiment 2

In this section, we are going to use the best models achieved at the experiment that we have done before, to apply some changes in the functions that we use at each layer.

The best models that we have achieved previously are:

XOR Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 100
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 1 (Cross-Entropy)
- Softmax in Output Layer: True
- Data Normalization: False
- Online Version: False

COMPAS Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 4
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 1 (Cross-Entropy)
- Softmax in Output Layer: True
- Data Normalization: False
- Online Version: False

noMNIST Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 16
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 1 (Cross-Entropy)
- Softmax in Output Layer: True
- Data Normalization: False
- Online Version: False

Changes to Apply for All Datasets: Once the best architecture is decided, test the following combinations using the off-line algorithm:

- MSE error function and sigmoidal activation function in the output layer.
- MSE error function and softmax activation function in the output layer.
- Cross-entropy error function and softmax activation function in the output layer.

For each best model of its dataset, we are going to prove these 3 changes about the functions applied to the layers.

3.4.1. Results for XOR Dataset

Fixed Model Parameters:

- Hidden Layers: 2
- Neurons in Hidden Layer: 100
- Eta (η): 0.7
- Mu (μ): 1.0
- Data Normalization: False
- Online Version: False

Execution Schema:

1. MSE and Sigmoidal
2. MSE and Softmax
3. Cross-entropy and Softmax

Run	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
1	7.39833e-05 +- 1.49847e-05	7.39833e-05 +- 1.49847e-05	75 +- 0	75 +- 0	4.158103
2	0.000599249 +- 5.03008e-05	0.000599249 +- 5.03008e-05	100 +- 0	100 +- 0	3.830993
3	0.000599249 +- 5.03008e-05	0.000599249 +- 5.03008e-05	100 +- 0	100 +- 0	3.846533

Tabla 6: Functions Metrics for XOR Dataset

In the analysis of the XOR dataset, it is observed that the variation from the MSE to Cross-Entropy loss function does not significantly influence the results, suggesting that both functions are suitable for this specific binary classification problem. However, the transition from the sigmoidal to the softmax activation function shows a substantial improvement in model performance.

The shift to the softmax activation function is particularly relevant in classification problems, as it provides normalized outputs as probabilities, thus favoring a better representation of relationships between classes. This improvement can be attributed to the softmax function's ability to handle multiclass classification problems more effectively.

In summary, the choice of loss and activation functions in binary classification problems like XOR may not be critical, but the selection of the softmax activation function appears to be key for enhancing model generalization ability and representation in more complex classification cases.

3.4.2. Results for ProPublica COMPAS Dataset

Fixed Model Parameters:

- Hidden Layers: 2
- Neurons in Hidden Layer: 4
- Eta (η): 0.7
- Mu (μ): 1.0
- Data Normalization: False
- Online Version: False

Execution Schema:

1. MSE and Sigmoidal
2. MSE and Softmax
3. Cross-entropy and Softmax

Run	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
1	0.0647387 +- 0.0335123	0.0672062 +- 0.0348046	100 +- 0	100 +- 0	27.937801
2	0.299767 +- 0.000877069	0.306011 +- 0.000880332	100 +- 0	99.9446 +- 0	62.88565
3	0.297814 +- 0.00102205	0.305392 +- 0.000404637	100 +- 0	99.8448 +- 0.199557	156.266904

Tabla 7: Functions Metrics for ProPublica COMPAS Dataset

After analyzing the results obtained, we can observe notable differences regarding the behavior of the algorithm compared to the previous dataset.

For this dataset, although the CCR metrics do not vary much, the MSE metrics do for both sets. It can be seen that in the first execution the MSE value obtained is much lower, in addition to obtaining the results in a much shorter time compared to the rest of the executions.

With this information, we can conclude that the variation between the MSE and Cross-entropy error functions is not significant, but the variation of the activation function from softmax to sigmoidal is what represents the improvement mentioned above.

This can be attributed to its ability to handle binary classification problems, the sigmoidal function compresses the output between 0 and 1, which can be beneficial in binary classification problems or when the outputs must be interpreted as probabilities.

3.4.3. Results for noMNIST Dataset

Fixed Model Parameters:

- Hidden Layers: 2
- Neurons in Hidden Layer: 16
- Eta (η): 0.7
- Mu (μ): 1.0
- Data Normalization: False
- Online Version: False

Execution Schema:

1. MSE and Sigmoidal
2. MSE and Softmax
3. Cross-entropy and Softmax

Run	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
1	0.00440824 +- 0.00248628	0.00436263 +- 0.00247505	19.7111 +- 3.35739	19.0667 +- 5.02615	352.467235
2	0.0216922 +- 0.00167471	0.107539 +- 0.00294805	85.4667 +- 2.08629	71 +- 1.36626	673.197247
3	0.0022593 +- 0.000128988	0.116318 +- 0.0143732	100 +- 0	71.2667 +- 1.42049	2846.071890

Tabla 8: Functions Metrics for noMNIST Dataset

As we can see after studying the results obtained, the variation between the error function used is not very significant, although the variation of the activation function is.

Although both activation functions seem to give a good result in terms of error, when analyzing the correct classification rates, we see that for a sigmoidal activation function the result is much worse than for a softmax activation function.

An overfitting of the algorithm on the training data can also be intuited because we can see that despite the increase in the CCR for run 3 on the training data, the results on the validation data do not seem to exceed a certain limit.

Between the two implementations used with this type of function, we can see that the most efficient is the one that uses the MSE as an error function, since it obtains similar results in much less time, so it can be the most convenient.

3.5. Experiment 3

In this section, we will compare the best models achieved until now using the offline version, with the same models using the online version.

The best models achieved until now are:

XOR Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 100
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 1 (Cross-Entropy)
- Softmax in Output Layer: True
- Data Normalization: False
- Online Version: False

COMPAS Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 4
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 0 (MSE)
- Softmax in Output Layer: False
- Data Normalization: False
- Online Version: False

noMNIST Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 16
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 0 (MSE)
- Softmax in Output Layer: True
- Data Normalization: False
- Online Version: False

Now, we are going to prove these models changing to online mode, and we will compare the results obtained.

3.5.1. Comparison of modes

1. Offline
2. Online

Run	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
1	0.000599249 +- 5.03008e-05	0.000599249 +- 5.03008e-05	100 +- 0	100 +- 0	4.230663
2	9.60261e-05 +- 1.89384e-05	9.60261e-05 +- 1.89384e-05	100 +- 0	100 +- 0	4.986208

Tabla 9: Comparison Metrics for XOR Dataset

Run	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
1	0.0647387 +- 0.0335123	0.0672062 +- 0.0348046	100 +- 0	100 +- 0	28.941387
2	0.0620363 +- 0.0323499	0.0643818 +- 0.0335814	100 +- 0	100 +- 0	38.936956

Tabla 10: Comparison Metrics for ProPublica COMPAS Dataset

Run	Train MSE	Test MSE	Train CCR (%)	Test CCR (%)	Execution Time (s)
1	0.0216922 +- 0.00167471	0.107539 +- 0.00294805	85.4667 +- 2.08629	71 +- 1.36626	685.695847
2	0.817862 +- 0.0633567	0.850576 +- 0.063249	22.4 +- 1.74072	21.9333 +- 2.61109	95.479731

Tabla 11: Comparison Metrics for noMNIST Dataset

As we can see, for the first two datasets, the difference is very small, representing an improvement, but not significant, in addition to a small increase in execution time.

However, for the last dataset, there are significant differences. It can be seen that both the MSE and CCR metrics are worse for the model that uses the online version. It is true that the difference in time is abysmal, with the online version assuming a much better time, although perhaps not compensable with the success rate obtained.

We can conclude that the online version can adapt well to small data sets when classifying, but with regard to large sets, the offline version shows to be much more effective, even requiring greater execution time and cost computational.

3.6. Conclusions

During the execution of this practice, we have been able to study and delve into the development of an algorithm capable of correctly classifying data sets into the system.

To do this, several experiments have been carried out, using the results obtained from them, making more changes and trying to improve the results obtained.

We have been able to learn how the appropriate choice of certain parameters for the dataset is essential for the algorithm to work accordingly.

It should also be said that we are aware that we can continue to delve deeper into this entire algorithm by carrying out many more tests with other parameters and datasets, in order to improve it.

3.7. Best Models

The best model we have achieved for each dataset is:

XOR Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 100
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 1 (Cross-Entropy)
- Softmax in Output Layer: True
- Data Normalization: False
- Online Version: False

COMPAS Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 4
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 0 (MSE)
- Softmax in Output Layer: False
- Data Normalization: False
- Online Version: False

noMNIST Dataset:

- Hidden Layers: 2
- Neurons in Hidden Layer: 16
- Eta (η): 0.7
- Mu (μ): 1.0
- Error Function: 0 (MSE)
- Softmax in Output Layer: True
- Data Normalization: False
- Online Version: False

And their best confusion matrix:

XOR Best Confusion Matrix

$$\begin{bmatrix} \text{Expected Class} & \text{Class 1} & \text{Class 2} \\ \text{Class 1} & 2 & 0 \\ \text{Class 2} & 0 & 6 \end{bmatrix}$$

Seed = (1,2,3,4,5)

COMPAS Best Confusion Matrix

$$\begin{bmatrix} \text{Expected Class} & \text{Class 1} & \text{Class 2} \\ \text{Class 1} & 0 & 0 \\ \text{Class 2} & 0 & 5861 \end{bmatrix}$$

Seed = (1,2,3,4,5)

NOMNIST Best Confusion Matrix

$$\begin{bmatrix} \text{Expected Class} & \text{Class 1} & \text{Class 2} & \text{Class 3} & \text{Class 4} & \text{Class 5} & \text{Class 6} \\ \text{Class 1} & 83 & 4 & 0 & 73 & 34 & 6 \\ \text{Class 2} & 0 & 188 & 1 & 5 & 6 & 0 \\ \text{Class 3} & 0 & 2 & 188 & 3 & 5 & 2 \\ \text{Class 4} & 0 & 2 & 3 & 188 & 3 & 4 \\ \text{Class 5} & 0 & 2 & 6 & 0 & 187 & 5 \\ \text{Class 6} & 0 & 3 & 2 & 1 & 4 & 190 \end{bmatrix}$$

Seed = (4)

4. Bibliografía

- Curso Introducción a los Modelos Computacionales
- scratch-multilayer-perceptron
- davidalbertonogueira/MLP: Simple multilayer perceptron
- mLEARn: An Implementation of Multi-layer Perceptron in C++