
Universidad de Córdoba
Escuela Politécnica Superior de Córdoba

Computer Science Engineering Degree
Mention: Computation
Fourth year. First quadrimester


Introduction to Computational Models

# Assignment 2: Multilayer Perceptron for classification problems


*Ventura Lucena Martínez*
31008689C
i72lumav@uco.es



Academic Year 2021-2022
Cordoba, October 31, 2021

# Contents

# List of Figures

# List of Tables

# List of Algorithms

**Abstract**

This lab assignment [1] serves as familiarisation with neural network computational models, in particular, with the multilayer perceptron. To do this, an implementation of the basic back-propagation algorithm for the multilayer perceptron has been carried out with an analysis checking the effect of the different parameters: network architecture or topology, moment factor ($\mu$), use of validation set, decrease of the learning rate ($\eta$) for each layer and so on). In special, this experiments will be tested in the off-line back-propagation algorithm, specific to this practice.

# 1 Architecture

The concept of **architecture** referred to neural networks makes mention not only of the number of neuronal layers or the number of neurons in each of them, but also the connection between neurons or layers, the type of neurons present and even the way in which they are trained.

## 1.1 Layers and Neurons

On the first hand, a **layer** is a set of neurons whose inputs come from a previous layer (or from the input data in the case of the first layer) and whose outputs are the input from a later layer. We will denote each layer as $l_i$.

On the second hand, the basic unit of the neural network is the **perceptron** or **neuron**. Each neuron has **inputs** with $x_i$ values, each one weighted with its corresponding **weight** $w_i$ to be optimized.

## 1.2 Connetions

The neural network has its **neurons fully connected**; this is, each neuron of the layer $i$ is fully connected with each neuron of the layer $i + 1$, for any neuron from $i = 0$ to $i = n\_neurons$ - 1. In the experiments we will see the following cases:

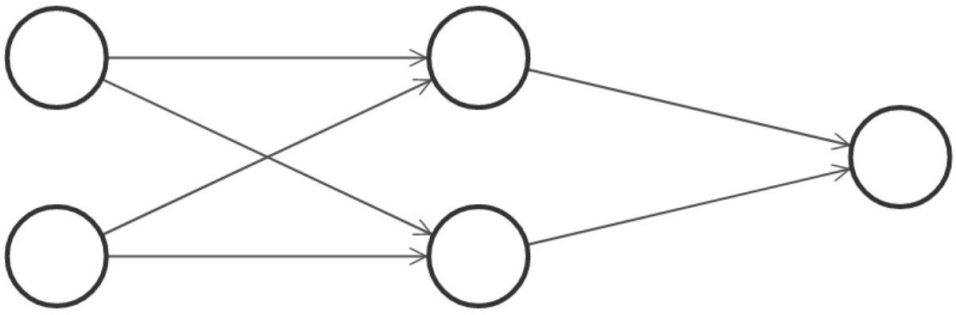

Figure 1: Topology based in 1 hidden layer and 2 neurons/each.

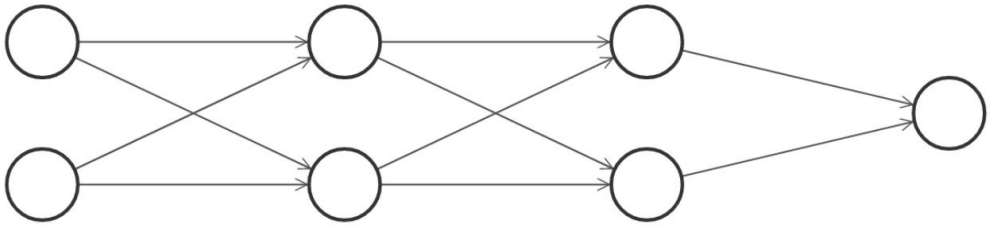

Figure 2: Topology based in 2 hidden layer and 2 neurons/each.

## 1.3 Type of neurons

We can see two types of neurons in the code:

### 1.3.1 Sigmoid activation function

As the previous lab assignment, we can find the on-line mode where the main types of neurons have a sigmoid activation function:

$$\sigma(x) = \frac{1}{1 + e^{-(x-t)}} \tag{1}$$

### 1.3.2 Softmax activation function

Furthermore, we have implemented the off-line mode, where we can find the softmax activation function. The network is configured to output N values, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class:

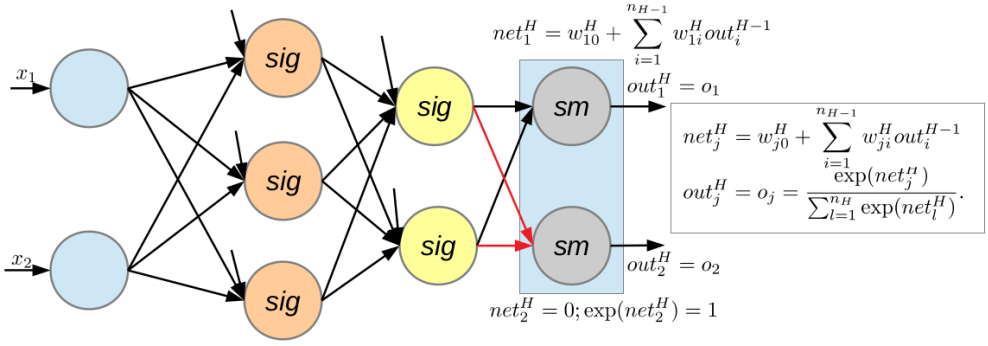


Figure 3: Softmax activation function. [2]

## 1.4 Training

Two types of error functions were used:

### 1.4.1 Minimum Square Error

It is an estimation method which minimizes the mean square error (MSE), which is a common measure of estimator quality, of the fitted values of a dependent variable. We recall that the MSE is defined as follows:

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\left(\frac{1}{k}\sum_{o=1}^{k}(d_{po} - o_{po})^2\right) \tag{2}$$

### 1.4.2 Cross-Entropy

Cross-entropy is a measure from the field of information theory, building upon entropy and generally calculating the difference between two probability distributions.

$$L = \frac{1}{N}\sum_{p=1}^{N}\left(\frac{1}{k}\sum_{o=1}^{k} d_{po} ln(o_{po})\right) \tag{3}$$

- N: number of pattern in the considered dataset.
- $k$: number of outputs.
- $d_{po}$: target value for pattern $p$ and the output variable $o$.

- $o_{po}$: predicted value.

# 2 Back-propagation algorithm

The pseudo code that adheres to the off-line backpropagation algorithm is the following:

**Algorithm 1** Back-propagation

```
i ← 1
w^h_{j,i} ← U[−1, 1]                    ▷ Random values between -1 and +1
while StopCondition == false do
    while i < nPatterns − 1 do
        Δw^h_{j,i} ← 0
        out^0_j ← x_j
        feedInputs();
        forwardPropagate();
        backpropagateError();
        accumulateChange();
        weightAdjustment();
    end while
end while
```

- *feedInput()*: feed the input neurons of the network with a vector passed as an argument.
- *forwardPropagate()*: calculate and propagate the outputs of the neurons, from the first layer until the last one.
- *backpropagateError()*: backpropagate the output error wrt a vector passed as an argument, from the last layer to the first one.
- *accumulateChange()*: accumulate the changes produced by one pattern and save them in $\Delta_w$.
- *weightAdjustment()*: update the network weights, from the first layer to the last one.

We have to consider some changes compared to the previous delivery:

1. *forwardPropagate()*: some changes were made in order to adapt the neurons of our neural network to a softmax function, so that, we have to pay attention at the last layer of the network, were we apply the softmax.

2. *backpropagateError()*: some changes were made related to the error function and the output function used:

   - Derivatives for sigmoid neurons:
     - Output layer:
       * MSE:

$$\delta_j^H \leftarrow -(d_j - out_j^H) \cdot out_j^H \cdot (1 - out_j^H) \tag{4}$$

       * Cross-entropy:

$$\delta_j^H \leftarrow -\left(\frac{d_j}{out_j^H}\right) \cdot out_j^H \cdot (1 - out_j^H) \tag{5}$$

     - Hidden layers:

$$\delta_j^h \leftarrow -\left(\sum_{i=1}^{n_{h+1}} w_{i,j}^{h+1} \delta_i^{h+1}\right) \cdot out_j^h \cdot (1 - out_j^h) \tag{6}$$

   - Derivatives for softmax functions (only output layer):
     - MSE:

$$\delta_j^H \leftarrow -\sum_{i=1}^{n_H} \left((d_i - out_i^H) \cdot out_j^H (I(i == j) - out_i^H)\right) \tag{7}$$

     - Cross-entropy:

$$\delta_j^H \leftarrow -\sum_{i=1}^{n_H} \left(\left(\frac{d_i}{out_i^H}\right) \cdot out_j^H (I(i == j) - out_i^H)\right) \tag{8}$$

3. *weightAdjustment()*: some modifications were made:

   - For each neuron from the first layer to layer $h$-1:

$$w_{j,i}^h \leftarrow w_{j,i}^h - \frac{\eta \Delta w_{j,i}^h}{N} - \frac{\mu\Big(\eta \Delta w_{j,i}^h (t-1)\Big)}{N} \tag{9}$$

   - For the bias:

$$w_{j,0}^h \leftarrow w_{j,0}^h - \frac{\eta \Delta w_{j,0}^h}{N} - \frac{\mu\Big(\eta \Delta w_{j,0}^h (t-1)\Big)}{N} \tag{10}$$

# 3 Experiments

We will test different neural network configurations and run each configuration with five seeds (1, 2, 3, 4 and 5). Based on the results obtained, the standard and mean deviation of the error will be obtained. Depending of the error function selected, the MSE error or Cross-entropy error will be calculated for both the training set and the test set.

To assess how the implemented algorithm works, we will use three datasets, in which we will use the following nomenclature:

- $n$: number of inputs.
- $h$: number of neurons in each hidden layer.
- $k$: number of outputs.
- $l$: number of hidden layers.

**NOTE: Due to performance problems with the neural network, the experiments have not been carried out yet. This is due to a coding error that, as of delivery date, could not be found. That is why it is preferable to wait for the resolution of the error to obtain good results.**

## 3.1 XOR problem

This dataset represents the problem of non-linear classification of the XOR. The same file will be used for train and test:

| | **Train** | **Test** |
|---|---|---|
| **Network Architecture {n:h:k}** | **Mean +- Std** | **Mean +- Std** |
| $\{2:2:1\}$ | | |
| $\{2:4:1\}$ | | |
| $\{2:8:1\}$ | | |
| $\{2:32:1\}$ | | |
| $\{2:64:1\}$ | | |
| $\{2:100:1\}$ | | |

Table 1: XOR test with $l$=1.

### 3.2

### 3.3

# 4 Extra experiments

Known the best network architecture for each problem, we will try some combinations for parameters $v$ and $F$, that help our neural network to not over-learn. Although we could get worse results, over-learning would be a problem if we want a general approach (instead of a concrete one):

# 5 Conclusion

# References

[1] Pedro Antonio Gutiérrez Peña. *Lab assignment 2: Multilayer perceptron for classification problems*. URL: `https://moodle.uco.es/m2122/mod/resource/view.php?id=49028`.

[2] Pedro Antonio Gutiérrez Peña. *Lab assignment 2: additional contents*. URL: `https://moodle.uco.es/m2122/mod/resource/view.php?id=49029`.