University of Córdoba

Computer engineering
Computer Science Engineering Degree
4th course

Introduction to computational models

Lab Assignment 1: Multilayer perceptron implementation

Valentín Gabriel Avram Aenachioei
03524931C
p92avavv@uco.es

Academic year 202-2023
Córdoba, June 9, 2023

Contents

List of Tables

List of algorithms

In this paper, I will set out how I have done the first lab assignment of the subject, the Multilayer Perceptron, explaining the Neural Network model used, its architecture and layer organization, how its algorithm works, and the pseudocode explaining the implementation of the learning process. As a conclusion, I will show, as experiments, ways to improve the error values given as a guideline, using different architectures for each dataset.

The main goal of this lab assignment is the subsequent analysis, emphasising this over the implementation of the code.

1 Description of the model

The neural network model implemented is a Multilayer Perceptron, explained in the theoretical classes. The multilayer perceptron is a simple C structure with 4 parameters, a learning rate, a momentum factor, the number of neurons, and a vector of layer structures. The multilayer perceptron structure contains the functions of the back-propagation algorithm.

Each layer strucuture has only two parameters, the number of neurons on each layer, and a vector of neuron strucutres. The main component of the architecture are the neuron structures, containing their own deltas, weights stored in a vector, and output. The neurons in the hidden and output layers will apply a sigmoidal function, and will have a bias.

The structures of neurons, layers, and the multilayer perceptron is already given.

There are multiple layers, an input layer, an output layer, and in between, there are $\boldsymbol{l}$ hidden layers given by the user. Each hidden layer has $\boldsymbol{h}$ nodes, given by the user, being the neurons of the input and output layer determined by the problem itself. The default values are 1 hidden layer with 5 neurons, using a learning rate of 0.1 and a momentum factor of 0.9.

The model works in online mode, which means that in every iteration of the online back-propagation algorithm applied to the training set, the error will be computed and the weights of each neuron will be update according to the error. In the first iteration, the weights of the neurons will be established randomly.

Two datasets will be used, a training dataset and a testing dataset. The neural network will train using the training dataset, applying $\boldsymbol{i}$ iterations for

the back-propagation algorithm, and it will test the results with the testing dataset. The algorithm can stop earlier if the network it is not improving after 50 iterations of the algorithm.

Also, it is posible to normalize the data from both datasets before using them.

2 Pseudocodes

The back-propagation algorithm does $\boldsymbol{n}$ iterations of ***onlineEpoch*** function:

The pseudocode 1 shows the onlineEpoch function.

Algorithm 1 Backpropagation

```
1: procedure ONLINE EPOCH
2:     for all hidden and output layers i do
3:         for all neurons j in layer i do
4:             for all neurons k +1 in layer i − 1 do
5:                 Δ^i_jk ← 0
6:     for all inputs i do
7:         out^0_j ← i_j
8:     forwardPropagation()
9:     backpropagateError()
10:    accumulateChange()
11:    weightAdjustment()
```

2.1 Forward Propagation

The pseudocode 2 shows the forward propagation function, where the output of the neurons are calculated and propagated, from the first layer to the last one.

Algorithm 2 Forward propagation of the ouputs

```
1: procedure FORWARDPROPAGATION
2:     for all layer i do
3:         for all neurons j in layer i − 1 do
4:             out^i_j ← 1 / (1+exp(−(w^i_j0 + Σ_{k=1}^{n_{i−1}} w^i_jk out^{i−1}_k)))
```

2.2 backpropagation of the Error

The pseudocode 3 shows the backpropagation of the output error function, where the output error of the neurons are backpropagated, from the last layer to the first one.

Algorithm 3 Backpropagation of the output error

1: **procedure** BACKPROPAGATEERROR
2: **for all** output neuron i **do**
3: $\delta_i^{layers-1} \leftarrow -(target_i - out_i^{layers-1})(out_i^{layers-1})(1 - out_i^{layers-1})$
4: **for** i from layers - 1 to 1 **do**
5: **for all** neurons j in layer i **do**
6: $\Delta_j^i \leftarrow ((\sum_{k=1}^{n_{i+1}} w_{kj}^{i+1}\delta_k^{i+1})out_j^i(1 - out_j^i))$

2.3 Accumulate Changes

The pseudocode 4 shows the accumulation of changes function, where the changes produced by one pattern are accumulated and saved in $\delta_{ji}^h(t)$.

Algorithm 4 Accumulation of changes

1: **procedure** ACCUMULATECHANGE
2: **for all** layers i **do**
3: **for all** neurons j in layer i **do**
4: **for all** neurons k in layer $i - 1$ **do**
5: $\Delta_{jk}^i \leftarrow \Delta_{jk}^i + (\delta_j^i * out_k^{i-1})$
6: $\Delta_{j0}^i \leftarrow \Delta_{j0}^i + \delta_j^i$

2.4 Adjust Weights

The pseudocode 5 shows the weight adjustment function, wherew the weights of the network are updated, from the first to the last layer.

Algorithm 5 Adjustment of weights

1: **procedure** WEIGHTADJUSTMENT
2: **for all** layers i **do**
3: **for all** neurons j in layer i **do**
4: **for all** neurons k in layer $i-1$ **do**
5: $w^i_{jk} \leftarrow w^i_{jk} - (\eta\Delta w^i_{jk} + \mu(\eta\Delta w^i_{jk}(t-1)))$
6: $w^i_{j0} \leftarrow w^i_{j0} - (\eta\Delta w^i_{j0} + \mu(\eta\Delta w^i_{j0}(t-1)))$

With the *onlineEpoch* 1 function and the five functions it calls inside, the neuronal network does its learning process.

3 Experiments

In this section, different architectures will be tested, looking to improve the error given as a guideline for each possible dataset.

In the following tables, μ represents the momentum factor and η represents the learing rate. The architecture will be represnted as $\{n : h : h : k\}$, where n represent the input layer, k represents the output layer and h represents the number of neurons on each hidden layer.

3.1 XOR Dataset

The XOR dataset represent the problem of non-linear classification of the XOR logical operation. It is the smallest dataset, and the same dataset will be used for training and for testing. In this dataset, we are trying to improve the given guideline :

$MSE_{train} = MSE_{test} = 0.25$

The table 1 shows the results of the experiments using the given architectures and the default values for μ, η and the number of iterations. The improved results, trying to improve the given guidelines for MSE values using different values for the parameters are marked in **bold**.

As a conclusion, we can observe that with a low number of neurons, the MSE will not improve much, doesnt matter how we increase the number of iterations, momentum factor or learning rate. As we add more neurons, the results are better. For the same architecture, we can upgrade the results by using higher μ values or higher η values in a specific range.

Using too high μ or η values will worsen the result, while using values in a specific range will improve the results as proben in the architecture $\{n : 2 : 2 : k\}$. In that case, using a hig value of η the result is better, but if the value of μ or η is too increased or too decreased, the results can get worse.

We can see an improvement as we increase the number of neurons, but with a single layer and 100 neurons, there is a worsening compared to using less neurons. As a contrast, using a small number of neurons with two layers give us worse results compared to the same number of neurons with a single layer. The number of neurons must be proportional to the number of layers.

Some of the results can be improved, but as every architecture already

Architecture	**Nº iterations**	μ	η	**Training MSE**	**Testing MSE**
$\{n : 2 : k\}$	1000	0.9	0.1	0.21951	0.21951
$\{n : 4 : k\}$	1000	0.9	0.1	0.19038	0.19038
$\{n : 8 : k\}$	1000	0.9	0.1	0.01367	0.01367
$\{n : 16 : k\}$	1000	0.9	0.1	0.07317	0.07317
$\{n : 32 : k\}$	1000	0.9	0.1	0.03703	0.03703
$\{n : 64 : k\}$	1000	0.9	0.1	0.02019	0.02019
$\{n : 100 : k\}$	1000	0.9	0.1	0.06228	0.06228
$\{n : 2 : 2 : k\}$	1000	0.9	0.1	0.24788	0.24788
$\boldsymbol{\{n : 2 : 2 : k\}}$	**1000**	**0.9**	**1**	**0.16338**	**0.16338**
$\{n : 4 : 4 : k\}$	1000	0.9	0.1	0.24574	0.24574
$\{n : 8 : 8 : k\}$	1000	0.9	0.1	0.20652	0.20652
$\{n : 16 : 16 : k\}$	1000	0.9	0.1	0.05835	0.05835
$\{n : 32 : 32 : k\}$	1000	0.9	0.1	0.00819	0.00819
$\{n : 64 : 64 : k\}$	1000	0.9	0.1	0.00278	0.00278
$\{n : 100 : 100 : k\}$	1000	0.9	0.1	0.00165	0.00165

Table 1: XOR dataset results

improves the given guideline, we can test different values for μ, η or the number of iterations to get better results. We can notice that $MSE_{train} = MSE_{test}$, because we are using the same dataset for training and for testing.

3.2 Sine function Dataset

The sine function dataset represent the graphic representation of a sine function, adding some random random noise to it. Two different datasets will be used, one for training and one for testing. For this dataset, we are trying to improve the given guideline :

$MSE_{train} = 0.02968$ $MSE_{test} = 0.036366$

The table 2 shows the results of the experiments using the given architectures and the default values for μ, η and the number of iterations. The improved results, trying to improve the given guidelines for MSE values using different values for the parameters are marked in **bold**.

We can draw similar conclusions as in the previous case, with a low number of neurons, we cannot obtain the expected results, as in the $\{n : 2 : k\}$ case. We can improve those result using two layers instead of one. Furthermore, with a higher number of neurons tends to perform worse. Again, it

Architecture	**Nº iterations**	μ	η	**Training MSE**	**Testing MSE**
$\{n : 2 : k\}$	1000	0.9	0.1	0.02968	0.03639
$\{n : 4 : k\}$	1000	0.9	0.1	0.02936	0.03642
$\{n : 8 : k\}$	1000	0.9	0.1	0.02894	0.03598
$\{n : 16 : k\}$	1000	0.9	0.1	0.02871	0.03537
$\boldsymbol{\{n : 16 : k\}}$	**1000**	**0.1**	**2**	**0.02330**	**0.03039**
$\{n : 32 : k\}$	1000	0.9	0.1	0.02855	0.03516
$\{n : 64 : k\}$	1000	0.9	0.1	0.02753	0.03431
$\{n : 100 : k\}$	1000	0.9	0.1	0.02775	0.03531
$\{n : 2 : 2 : k\}$	1000	0.9	0.1	0.03159	0.05778
$\{n : 4 : 4 : k\}$	1000	0.9	0.1	0.03157	0.05862
$\{n : 8 : 8 : k\}$	1000	0.9	0.1	0.03147	0.05920
$\{n : 16 : 16 : k\}$	1000	0.9	0.1	0.03187	0.06219
$\{n : 32 : 32 : k\}$	1000	0.9	0.1	0.03036	0.06273
$\{n : 64 : 64 : k\}$	1000	0.9	0.1	0.02434	0.05348
$\{n : 100 : 100 : k\}$	1000	0.9	0.1	0.02170	0.04919

Table 2: Sine dataset results

is improved using more layers. As in the previous case, the results are improved using an increased μ value and a decreased η value. Also, the result are better with a larger number of iterations.

As this dataset is complex than the XOR dataset, we can observe that the improvement is smaller than the previous case.

3.3 Quake dataset

The quake dataset represent a database which look to finds out the strength of an earthquake, using the depth of focus, the latitude and the longitude as input variables. Two different datasets will be used, one for training and one for testing. For this dataset, we are trying to improve the given guideline :

$MSE_{train} = 0.030206$ $MSE_{test} = 0.027324$

The table 3 shows the results of the experiments using the given architectures and the default values for μ, η and the number of iterations.

The conclusions are similar to the previous cases. For this dataset, the training MSE is improved in every architecture, but the testing MSE is not improved. This can be solved by decreasing the value of η.

Architecture	**Nº iterations**	μ	η	**Training MSE**	**Testing MSE**
$\{n : 2 : k\}$	1000	0.9	0.1	0.03011	0.02721
$\{n : 4 : k\}$	1000	0.9	0.1	0.02997	0.027108
$\{n : 8 : k\}$	1000	0.9	0.1	0.02986	0.02704
$\{n : 16 : k\}$	1000	0.9	0.1	0.029805	0.02697
$\{n : 32 : k\}$	1000	0.9	0.1	0.02979	0.2696
$\{n : 64 : k\}$	1000	0.9	0.1	0.02978	0.027005
$\{n : 100 : k\}$	1000	0.9	0.1	0.02974	0.027001
$\{n : 2 : 2 : k\}$	1000	0.9	0.1	0.03011	0.04199
$\{n : 4 : 4 : k\}$	1000	0.9	0.1	0.03006	0.04189
$\{n : 8 : 8 : k\}$	1000	0.9	0.1	0.02998	0.04175
$\{n : 16 : 16 : k\}$	1000	0.9	0.1	0.02989	0.04158
$\{n : 32 : 32 : k\}$	1000	0.9	0.1	0.02978	0.04139
$\{n : 64 : 64 : k\}$	1000	0.9	0.1	0.02953	0.04133
$\{n : 100 : 100 : k\}$	1000	0.9	0.1	0.02938	0.02713

Table 3: Quake dataset results

We can observe how the training MSE gets worse from the architecture $\{n : 64 : 64 : k\}$.

3.4 Crop yield Dataset

The Crop Yield dataset considers the yield prediction of harvest of the 10 main crop of harvest of the 10 main crops that are mainstay of food. Two different datasets will be used, one for training and one for testing. Both, training and testing datasets needs to be normalized before applying the back-propagation algorithm. For this dataset, we are trying to improve the given guideline :

$MSE_{train} = 0.0505$ $MSE_{test} = 0.0645$

The table 4 shows the results of the experiments using the given architectures and the default values for μ, η and the number of iterations.

The conclusions are, again, similar to the other cases. For this dataset, every architecture tested with the default parameters has improved the given guideline. There is no need to change any of the parameters.

With a single layer, there is an improvement until the architecture $\{n : 32 : k\}$. From this, there is not any improvement, even using more neurons.

Architecture	**Nº iterations**	μ	η	**Training MSE**	**Testing MSE**
$\{n : 2 : k\}$	1000	0.9	0.1	0.00394	0.00555
$\{n : 4 : k\}$	1000	0.9	0.1	0.00378	0.00471
$\{n : 8 : k\}$	1000	0.9	0.1	0.00365	0.00429
$\{n : 16 : k\}$	1000	0.9	0.1	0.00375	0.00391
$\{n : 32 : k\}$	1000	0.9	0.1	0.00371	0.00376
$\{n : 64 : k\}$	1000	0.9	0.1	0.00369	0.00351
$\{n : 100 : k\}$	1000	0.9	0.1	0.00366	0.00374
$\{n : 2 : 2 : k\}$	1000	0.9	0.1	0.00421	0.00515
$\{n : 4 : 4 : k\}$	1000	0.9	0.1	0.00386	0.004605
$\{n : 8 : 8 : k\}$	1000	0.9	0.1	0.00372	0.00434
$\{n : 16 : 16 : k\}$	1000	0.9	0.1	0.00376	0.00395
$\{n : 32 : 32 : k\}$	1000	0.9	0.1	0.003704	0.00382
$\{n : 64 : 64 : k\}$	1000	0.9	0.1	0.00369	0.00363
$\{n : 100 : 100 : k\}$	1000	0.9	0.1	0.00367	0.00374

Table 4: Crop yield dataset results

With two layers, there is an improvement as we increase the number of neurons.

3.5 Experimental conclusions

After experimenting with all the datasets, we can draw some conclusions.

First of all, with a small number of neurons, the neural network does not obtain its best results. As we increase the number of neurons, the results are improved. This increase in the number of neurons has to go along with the increase of the layers. With a single layer, a high number of neurons can worse the results. As we increase the number of layers, this results improve. The increase in the number of neurons must go hand in hand with the increase in the number of layers.

The number of iterations can improve the results as we increase it. This improvement is not so remarkable, since after a certain point the improvement is not noticeable. The improvement of the result, without changing the architecture of the network, has to be done by modifying the values of μ and η.

The η means that the weights in the network are updated by η times the

estimated weight error. This value should be in the range [$10^{-}6$ - 1.0] . With smaller values of η we can obtain better results

The μ parameter controlls the effect of the momentum, the influence of the previous changes in the direction of the movement. With higher values of μ we can improve the results.

The change of η and μ depends on the problem. We have to try values until we get the results wanted. Too large μ values or too small η values can worse the results instead of improving them.