Universidad de Córdoba
Escuela Politécnica Superior de Córdoba

Ingeniería Informática
Especialidad: Computación
Cuarto curso. Primer cuatrimestre

Introduction to computational models.

Lab Assignment 2: Multilayer perceptron for classification problems.

Javier Herrero Porras
i72hepoj@uco.es

Academic Course 2020-2021
Córdoba, October 25, 2020

Contents

Figure index ii

Table index iii

Algorithm index iv

1 New model features 1

2 Pseudocode 2

2.1 Weight adjustment 2

2.2 Forward Propagation 3

2.3 Back Propagation Error 4

3 Experiments and results discussion 5

3.1 Datasets 5

3.2 Parameters 5

3.3 Results obtained 6

3.3.1 XOR Problem 6

3.3.2 Divorce Dataset 8

3.3.3 noMNIST Dataset 11

List of Figures

1 Divorce Dataset 9
2 CCR for noMNIST 14
3 Incorrect Classified Patterns noMNIST 17

List of Tables

1 XOR Function 6
2 Result of experiment XOR 7
3 Result of experiment XOR: Algorithm Version 7
4 Result of experiment for best arquitecture XOR 8
5 Result of experiment Divorce Dataset 8
6 Result of experiment Divorce 9
7 Result of experiment Divorce: Algorithm Version 9
8 Result of experiment for best architecture Divorce 10
9 Result of experiment noMNIST Dataset 11
10 Result of experiment noMNIST 11
11 CCR Result of experiment noMNIST 12
12 Result of experiment noMNIST: Algorithm Version 12
13 CCR Result of experiment noMNIST: Algorithm Version . . . 13
14 Result of experiment for best architecture noMNIST 14
15 CCR Result of experiment for best architecture noMNIST . . 15
16 Confusion matrix: Cross Entropy + Softmax 15
17 Confusion matrix: MSE + Softmax 16

Algorithm Index

1 Weight Adjustment 2
2 Forward Propagation 3
3 Back Propagation Error 4

1 New model features

In this practice I have worked with the model developed for the previous assignment in order to be make it able to work with classification problems. This is achieved by adding the following characteristics to the code:

- Softmax activation function in the output layer.
- Cross-entropy as an error measure.
- Off-line version of the algorithm.

The first feature is referred to a probabilistic problem. There is a pattern that belongs to a set of classes: $C_1, C_2, ..., C_J$ so the the model has to predict the class by the probability of belonging to each of the classes. To make it possible, I used the softmax function in the output layer, so that probabilities will be between 0 and 1 and their sum will be 1:

$$net_j^H = w_{j0} + \sum_{i=1}^{n_{H-1}} w_{ij} * out_i^{H-1}$$

$$out_j^H = \frac{\exp\left(net_j^H\right)}{\sum_{l=1}^{n_H} \exp\left(net_l^H\right)}$$

As the sum of all probabilities has to be 1, we can work with with $n_H - 1$ neurons in the output layer, because the last neuron probability will be $n_H = 1 - \sum_1^J o_j$.

Also, I have used cross entropy as error function, because in classification problem it takes into account if the prediction is close to a desired output and it is a better way to compute error. It happens because MSE gives more importance to the incorrect outputs and we need a measure that penalise more the errors made for the correct class.

Furthermore, in this lab assignment I have implemented off-line version of the algorithm, which consists in considering all training patterns to make a weight update. Both versions (on-line and off-line) will be analysed in the experiments to decide which is better for each dataset.

The algorithm used to make the network learning is the same as the previous lab assignment **Back-propagation Algorithm**. The main idea is to adjust the weights of the connections between neurons to reduce the error of regression or classification on a given set.

2 Pseudocode

In this lab assignment the code has been changed in some functions. I will explain these changes in this section.

2.1 Weight adjustment

This algorithm update weights from layer 1 to layer H according values of learningRate, deltaW and lastDeltaW.

Algorithm 1 Weight Adjustment

```
procedure WEIGHTADJUSTMENT
    for all layer $l$ from 1 to H do
        $lR \leftarrow eta * decrementFactor^{-1*(H-l)}$
        for all neuron $n$ of layer $l$ do
            for all neuron $nPL$ of layer $l-1$ do
                $delta \leftarrow layers[l].neurons[n].delta[nPL]$
                $lastDelta \leftarrow layers[l].neurons[n].lastDelta[nPL]$
                if online then
                    $layers[l].neurons[n].w[nPL] -= (lR * delta + mu * lR * lastDelta)$
                else
                    $layers[l].neurons[n].w[nPL] -= (lR * delta/nTrainingPatterns + mu * lR * lastDelta/nTrainingPatterns)$
                end if
            end for
            $delta \leftarrow layers[l].neurons[n].delta[0]$
            $lastDelta \leftarrow layers[l].neurons[n].lastDelta[nPL]$
            if online then
                $layers[l].neurons[n].w[0] -= (lR * delta + mu * lR * lastDelta)$
            else
                $layers[l].neurons[n].w[0] -= (lR * delta/nTrainingPatterns + mu * lR * lastDelta/nTrainingPatterns$
            end if
        end for
    end for
end procedure
```

2.2 Forward Propagation

This algorithm takes the values from the input layer and propagate them through the network. In this case the output function can be sigmoidal or softmax, so the code has been modified in order to implement softmax function.

Algorithm 2 Forward Propagation

```
procedure FORWARDPROPAGATE
    $accumulateNet \leftarrow 0$
    for all layer $l$ from 1 to H-1 do
        for all neuron $n$ of layer $l$ do
            $net[n] \leftarrow 0$
            if $layers[l].neurons[n].w! = NULL$ then
                for all neuron $nPL$ of layer $l - 1$ do
                    $weight \leftarrow layers[l].neurons[n].w[nPL]$
                    $output \leftarrow layers[l - 1].neurons[nPL - 1].out$
                    $net[n] += weight * output$
                end for
                $net[n] += layers[l].neurons[n].w[0]$
                $layers[l].neurons[n].out \leftarrow sigmoidal(net[n])$
            end if
            $accumulateNet += \exp^{-1*net[n]}$
        end for
        if $outputFunction == 1l == H - 1$ then
            for all neuron $n$ of layer $l1$ do
                $layers[l].neurons[n].out \leftarrow exp^{-1*net[n]}/accumulateNet$
            end for
        end if
    end for
end procedure
```

2.3 Back Propagation Error

In this algorithm the error of the output layer is calculated with respect to a target or a desired output. The derivative is saved on delta and then, it goes through the layers to obtain each neuron's deltaW. In this part I will focus in the first loop, where each neuron's delta of the neuron layer is calculated.

Algorithm 3 Back Propagation Error

```
1: procedure BACKPROPAGATEERROR
2:     $target \leftarrow desiredOutput$
3:     for all neurons $n$ from layer H-1 do
4:         $output \leftarrow layers[H-1].neurons[n].out$
5:         if $outputFunction == 0$ then
6:             if $errorFunction == 0$ then
7:                 $delta \leftarrow -1 * (target[n] - output) * output * (1 - output)$
8:             else
9:                 $delta \leftarrow -1 * (target[n]/output) * output * (1 - output)$
10:            end if
11:        else
12:            for all neurons $nSL$ from layer H-1 do
13:                if $errorFunction == 0$ then
14:                    $sumDelta+ = -1 * (target[n] - output) * output * (I(n == NSL) - output)$
15:                else
16:                    $sumDelta+ = -1 * (target[n]/output) * output * (I(n == NSL) - output)$
17:                end if
18:                $delta \leftarrow sumDelta$
19:            end for
20:        end if
21:    end for
22:    Update neurons delta from layer H-2 to 1
23: end procedure
```

3 Experiments and results discussion

3.1 Datasets

For these experiments, 3 different datasets have been used to train and test the neural network:

- *XOR problem*, where training dataset consists of 4 patterns with 2 inputs and 2 outputs. Train and test dataset have the same patterns.
- *Divorce dataset*, where training dataset consists of 127 patterns with 54 inputs and 2 outputs. Also, test dataset has 43 patterns.
- *noMNIST dataset*, where training dataset consists of 900 patterns with 784 inputs and 6 outputs. Also, test dataset has 300 patterns.

3.2 Parameters

In order to make the experiments analysed in this report, these values have been considered:

- *Validation Ratio* to create a subset from test dataset that will be used to validate neural network: $\{0.0; 0.15; 0.25\}$.
- *Learning Rate*, which is a tuning parameter of the neural network, that determines the step size at each iteration: $\{1.0; 2.0\}$.
- The number of *Hidden Layers* of the neural network, so the model can have a different number of layers to find the best configuration for each dataset: $\{1, 2\}$.
- The number of *Neurons* which form each hidden layer. This number together with hidden layers establish the *topology* of the neural network: $\{4; 8; 16; 64\}$.
- *Eta*, which controls the update of the weights, allowing us to give more or less importance to the weight derivative: $\{0.7\}$.
- *Mu* or momentum, to improve the convergence of the algorithm by controlling the direction of the weights: $\{1\}$.

3.3 Results obtained

3.3.1 XOR Problem

In this section, I have to make a neural network for the XOR problem, that consists in predicting the outputs of an XOR logic gate with two binary inputs. This problem has been modified in this lab assignment, because it has 2 outputs. The first one is activated when inputs are not equal, and the second one when are equal. Thus, the function output is shown in Table 1.

X	Y	Class 1	Class 2
-1	-1	0	1
-1	1	1	0
1	-1	1	0
1	1	0	1

Table 1: XOR Function

According to these values, the neural network should be able to learn from these patterns and make approximate outputs. This is a special problem because there are only 4 patterns, so train dataset and test dataset measures (MSE and std) will be the same.

In this experiment I have only worked with the best configuration achieved in the previous lab assignment, with 2 hidden layers and 100 neurons. In both lab assignments, I have implemented 2 error functions (MSE and Cross Entropy) and 2 activation functions in the output layer (sigmoidal and softmax), so there are 4 possible schemes:

- **MSE + Sigmoidal**
- **MSE + Softmax**
- **Cross Entropy + Softmax**
- **Cross Entropy + Sigmoidal**: this scheme won't be used because in one hand, cross entropy is often used in classification problems that means it penalises the error for the correct class. In the other hand, sigmoid function does not work properly in classification problems caused by it models the variables independently, and we want the sum of the predictions to be 1.

The result obtained for every possible scheme are shown in table 2 with 2 hidden layers and 100 neurons.

Combination	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
MSE + SIGMOID	0.0002852	0.00000635076	0.0002852	0.00000635076
MSE + SOFTMAX	0.0.1177150	0.1131220	0.1177150	0.1131220
CROSS ENTROPY + SOFTMAX	**0.0002203**	**0.0000285**	**0.0002203**	**0.0000285**
~~CROSS ENTROPY + SIGMOID~~	0.0000076	0.0000044	0.0000076	0.0000044

Table 2: Result of experiment XOR

The best result obtained in the experiment is the combination of Cross Entropy and Softmax in output layer. XOR problem is now a classification problem and as we saw in theory lessons Softmax works better with this type of problems. Also, mean square error does not work properly when we have a classification problem (which has probabilistic outputs).

In the lab lessons it was said that we should not try the combination of cross-entropy and sigmoidal activation function. But in this experiment, it got the best results by far in terms on train and test error. We will see in the last dataset why this combination is bad in classification problems. After reading some reports and forums [1], I have concluded that sigmoid function is used in some classification problems (with two output classes like in this dataset), but it is not appropriate when having 2 or more neurons in the output layer in binary classification models [2].

Once I found the best combination, I should check what algorithm version (on-line or off-line) is the best for this dataset. The results are shown in table 3

Version	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
Online	0.0000441	0.0000125	0.0000441	0.0000125
Offline	0.0002203	0.0000285	0.0002203	0.0000285

Table 3: Result of experiment XOR: Algorithm Version

As we can see in the previous table the best version is online. This learning update the weights for every pattern, not like offline learning, that updates the weights when considering all the training patterns. It is caused because we have few patterns, and it doesn't forget the old patterns.

Once we find the best learning version for the problem, we should change some internal parameters of the algorithm in order to reach the optimal configuration. In this case there are 4 patterns, therefore, the problem does not allow a validation subset. Then, we can only work with decreasing factor which influences the learning rate of the network. If this decreasing factor is higher, then the learning rate of each layer will increase.

v	F	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
0.0	**1**	**0.0000441**	**0.0000125**	**0.0000441**	**0.0000125**
0.0	2	0.0000803	0.0000180	0.0000803	0.0000180

Table 4: Result of experiment for best arquitecture XOR

Table 4 show us that the network works better with a lower decreasing factor (F), so we can conclude that it has better results with a lower learning rate which causes a lower update of the weights during the iterations.

3.3.2 Divorce Dataset

In this section the neural network will deal with a divorce dataset. It contains a set of questions to predict the divorce of a partner. In this first experiment, I have to find the best architecture for this dataset. The results are shown in table 5.

Hidden Layers	Neurons	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
1	**4**	**0.000924133**	**0.000453453**	**0.063785**	**0.0105061**
1	8	0.000458404	0.00006425	0.066887	0.0149574
1	16	0.000355219	0.00003110	0.065754	0.00603792
1	64	0.000209397	0.000007474	0.078624	0.0126553
2	4	0.0005356	0.00006124	0.068168	0.00806007
2	8	0.000341486	0.00001459	0.078960	0.00795407
2	16	0.000242971	0.00001972	0.078045	0.0140258
2	64	0.000126054	0.000008161	0.092593	0.00633997

Table 5: Result of experiment Divorce Dataset

In this case, results (test mean) obtained with 1 hidden layer and 4 neurons are the best ones obtained for the model. It seems that this function does not work well with a high number of neurons as we can see also with 2 hidden layers. This fact can be easily seen in figure 1 where we can see how the errors obtained with train dataset decrease when the model has more neurons, and in the right picture we can check that neural network is able to predict better the test dataset with a lower number of neurons. It is caused because the train data has a high number of inputs (54) and it does not have many patterns (only 127), so the model is poorly trained.

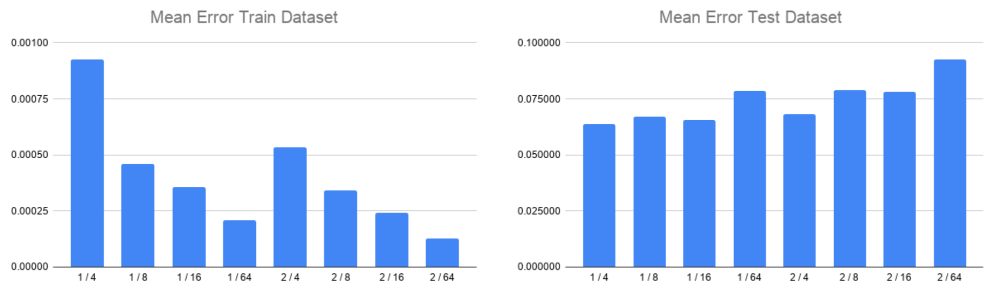

Figure 1: Divorce Dataset

Once the best architecture is found, we should check what combination of error function and activation function is working properly for this problem. To make this, I will test every possible scheme (section 3.3.1). The results obtained for every possible scheme are shown in table 6 with 1 hidden layer and 4 neurons.

Combination	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
MSE + SIGMOID	**0.0013181**	**0.0003502**	**0.0204936**	**0.0022098**
MSE + SOFTMAX	0.0018441	0.0020785	0.0207389	0.0022993
CROSS ENTROPY + SOFTMAX	0.0009241	0.0004535	0.0637852	0.0105061
~~CROSS ENTROPY + SIGMOID~~	0.0002664	0.0000204	0.0002877	0.0000422

Table 6: Result of experiment Divorce

The best result obtained in the experiment is the combination of Cross MSE and Sigmoid in output layer. Although divorce problem is a classification problem, the differences between the output layer activation function (Sigmoid and Softmax with MSE) are minimal.

Once the best architecture and configuration for the problem is found, I will find which algorithm version (on-line or off-line) works better in this problem.

Version	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
Online	0.0000018	0.0000001	0.0000025	0.0000006
Offline	0.0013181	0.0003502	0.0204936	0.0022098

Table 7: Result of experiment Divorce: Algorithm Version

The results shown in table 7 verify that on-line version gets much better results than off-line version. Remember that on-line learning perform a weight update for each training pattern. In this case, train dataset has 127 patterns so this version is working better because there are not a high number of patterns, so the model does not forget the old patterns.

Once we find the best learning version for the problem, we should change some internal parameters of the algorithm in order to reach the optimal configuration. In this case I will work with a validation dataset (v) and a decrement factor (F) for the learning rate. Results are shown in table 8.

v	F	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
0.0	1	**0.0000018**	**0.0000001**	**0.0000025**	**0.0000006**
0.0	2	0.0000040	0.0000001	0.0000053	0.0000016
0.15	1	0.0000022	0.0000002	0.0000042	0.0000021
0.15	2	0.0000047	0.0000001	0.0000071	0.0000018
0.25	1	0.0000025	0.0000002	0.0000047	0.0000022
0.25	2	0.0000053	0.0000001	0.0000084	0.0000017

Table 8: Result of experiment for best architecture Divorce

In table 8 we check that the best combination of these parameters is $v = 0.0$ and $F = 1$. But, why are the results worse with a validation dataset?

When we have a validation dataset we remove patterns from train dataset, in consequence, the neural network has not many patterns to learn from. If training dataset has few items then the learning is poor and results are worse. Otherwise, if training dataset has many items we can remove some patterns but learning process will be still good. In our case, there are few patterns (147) and it is better to assign them to learn dataset.

3.3.3 noMNIST Dataset

In this section the neural network will work with noMNIST dataset. It contains a set of letters (which are a,b,c,d,e and f) written with different typologies or symbols. Every pattern has 784 input variables, which represent pixels of the image. The neural network has to learn these patterns and then, it has to predict some letters in the test dataset. In the first experiment we are going to compare different architectures in order to get the lowest error in predictions. Result are shown in table 9.

Hidden Layers	Neurons	Mean$_{train}$	σ_{train}	Mean$_{test}$	σ_{test}
1	4	0.06086	0.00524	0.11295	0.01058
1	**8**	**0.03917**	**0.00282**	**0.09370**	**0.00621**
1	16	0.02296	0.00207	0.10214	0.00482
1	64	0.00374	0.00038	0.11476	0.01180
2	4	0.09187	0.01480	0.13562	0.01022
2	8	0.04176	0.00109	0.10979	0.00841
2	16	0.01831	0.00119	0.10382	0.01314
2	64	0.00263	0.00023	0.11315	0.01520

Table 9: Result of experiment noMNIST Dataset

In this case, results (test mean) obtained with 1 hidden layer and 8 neurons are the best ones obtained for the model. It seems that this function does not work well with a high or low number of neurons as we can see also with 2 hidden layers, because the results obtained with 8 and 16 neurons are quite better than obtained with 4 and 64.Once the best architecture is found, we should check what combination of error function and activation function is working properly for this problem. To make this, I will test every possible scheme (section 3.3.1).

The results obtained for every possible scheme are shown in table 10 with 1 hidden layer and 8 neurons.

Combination	Mean$_{train}$	σ_{train}	Mean$_{test}$	σ_{test}
MSE + SIGMOID	0.0441110	0.0025566	0.0523179	0.0043890
MSE + SOFTMAX	0.0377994	0.0025064	0.0478989	0.0035634
CROSS ENTROPY + SOFTMAX	**0.0391722**	**0.0028243**	**0.0936997**	**0.0062060**
~~CROSS ENTROPY + SIGMOID~~	0.0003620	0.0000227	0.0004128	0.0000499

Table 10: Result of experiment noMNIST

The results show us that in terms of mean error of the test dataset, the combination of cross entropy + sigmoidal is the best one. But for this problem, we need a measure to get the percentage of well classified instances. It is known as CCR *(Correctly Classified Ratio)*. This consists of:

$$CCR = 100 * 1/N * \sum_{p=1}^{N}(I(y_p = y_p^*))$$

For example, if we have a pattern which belongs to the class 4 and the neural network classify it on class 5, this pattern is not correctly classified so $(I(y_p = y_p^*)$ will be 0 and it won't be included on CCR. If for example, the neural network classify it on class 5, $(I(y_p = y_p^*)$ will be 1 and it will be computed on CCR. To resume, this is a good measure to check if our classifier is working properly for the dataset. In this case, I have been taken the best results according to CCR. Results obtained for this measure are shown in table 11.

Combination	CCR Mean$_{train}$	CCR σ_{train}	CCR Mean$_{test}$	CCR σ_{test}
MSE + SIGMOID	85.06670	1.70200	80.4000000	3.3559200
MSE + SOFTMAX	86.11110	1.07036	80.5333000	1.6944400
CROSS ENTROPY + SOFTMAX	**93.51110**	**0.55154**	**85.1333000**	**1.1274400**
~~CROSS ENTROPY + SIGMOID~~	35.60000	0.0000227	37.0000000	5.3249900

Table 11: CCR Result of experiment noMNIST

Now we can check that the combination of cross entropy + sigmoidal has the worst result by far, with a CCR = 37%. Here we prove that this combination is not valid in classification problems because it has a really good mean error, but its CCR is very low, so the classifier does not predict properly the classes. Due to this reason, I took the best architecture and best combination of error and activation function according to CCR measure.

According to the table 11, the best result obtained in the experiment is the combination of Cross Entropy and Softmax in output layer, which is in relation to the ideas learnt from theory lessons. Once the best architecture and configuration for the problem is found, I will find which algorithm version (on-line or off-line) works better in this problem, according to the errors and CCR.

Version	Mean$_{train}$	σ_{train}	Mean$_{test}$	σ_{test}
Online	0.8567640	0.0299010	0.8398550	0.0364409
Offline	**0.0391722**	**0.0028243**	**0.0936997**	**0.0062060**

Table 12: Result of experiment noMNIST: Algorithm Version

After checking the best algorithm version (table 12), we can easily confirm that according to errors, the best version by far is off-line. This idea is also verified with CCR obtained in these experiments (table 13) where on-line version obtains a very low CCR. This problem is classifying 6 classes so when it is learning the last patterns of the training dataset, it has forgotten the initial ones. Due to this reason the CCR obtained is very low, because the model is able to classify correctly the last patterns learnt but it does not classify correctly the first ones learnt. This problem could be solved by setting a update every a number of patterns.

Version	CCR Mean$_{train}$	CCR σ_{train}	CCR Mean$_{test}$	CCR σ_{test}
Online	19.4444	2.319	19.3333	2.41293
Offline	**93.5111**	**0.55154**	**85.1333**	**1.12744**

Table 13: CCR Result of experiment noMNIST: Algorithm Version

In figure 2 we can see how the CCR from the train, test and validation is changing over all the iterations of a random seed. In this case we are not considering validation dataset *(v=0.0)* so the error will be 0. We note that CCR train always increases because the model is being trained with this dataset. On the other hand, CCR test in some iterations decreases because maybe the model has not been trained with enough patterns of some classes.

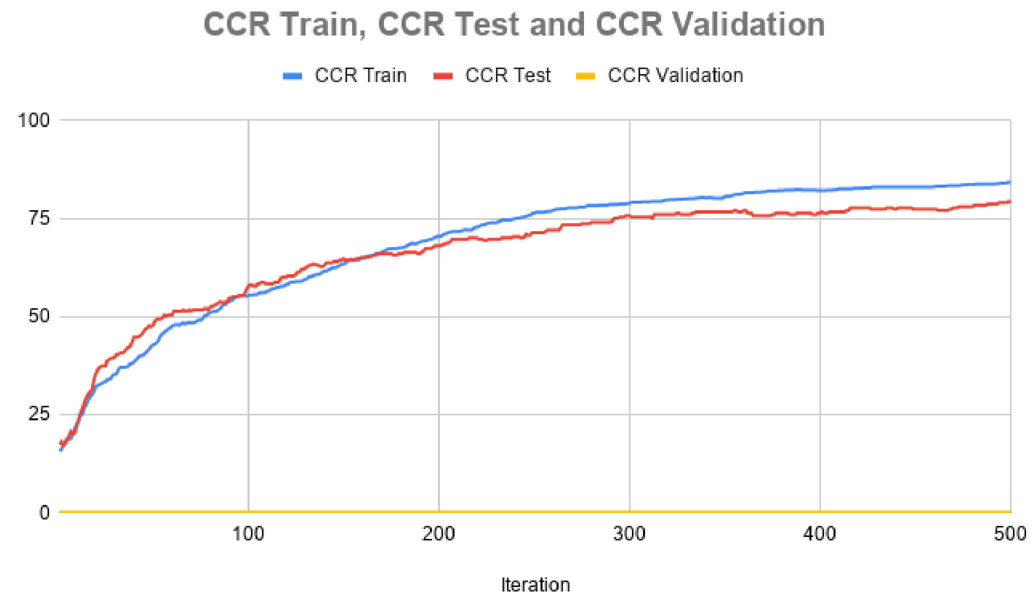

Figure 2: CCR for noMNIST

Once we find the best learning version for the problem, we should change some internal parameters of the algorithm in order to reach the optimal configuration. In this case I will work with a validation dataset (v) and a decrement factor (F) for the learning rate. Results are shown in table 14.

v	F	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
0.0	**1**	**0.0391722**	**0.0028243**	**0.0936997**	**0.0062060**
0.0	2	0.0751939	0.0054833	0.1056140	0.0063558
0.15	1	0.0523861	0.0104971	0.10101	0.00899253
0.15	2	0.0772358	0.0054122	0.1094790	0.0080764
0.25	1	0.0474902	0.0053946	0.0999645	0.0094082
0.25	2	0.0812191	0.0052836	0.115658	0.00891599

Table 14: Result of experiment for best architecture noMNIST

The results show us that the best combination of these parameters is $v = 0.0$ and $F = 1$, so the model works better without a validation dataset. As I discuss in the previous lab assignment, the validation dataset is better when training dataset has a high number of patterns because it is better in terms of computational cost, but it takes some patterns from the training dataset, so the train is worse and the mean errors increase and CCRs decrease. CCR results are shown in table 15.

v	F	Mean_{train}	σ_{train}	Mean_{test}	σ_{test}
0.0	1	**93.5111**	**0.55154**	**85.1333**	**1.1274**
0.0	2	86.7778	1.33518	79.9333	2.64491
0.15	1	91.555	1.75689	83.3333	1.33333
0.15	2	86.4052	1.20375	80	2.23706
0.25	1	92.237	1.37897	82.3333	2.39444
0.25	2	86.6963	1.12593	80	2.0548

Table 15: CCR Result of experiment for best architecture noMNIST

In order to finish the analysis of noMNIST dataset, in tables 16 and 17 we can see the confusion matrixes obtained with the best architecture, algorithm version and internal parameters for the test dataset. Previously, the results according to errors obtained for combinations: Cross Entropy + Softmax and MSE + Softmax were very similar, but in those tables we can check the differences of the classification of each confusion matrix.

	A	B	C	D	E	F
A	143	2	1	1	1	2
B	2	134	2	4	2	6
C	2	0	141	1	5	1
D	0	3	1	140	1	5
E	0	3	1	1	139	6
F	2	1	1	3	4	139

Table 16: Confusion matrix: Cross Entropy + Softmax

A confusion matrix resume if patterns are correctly classified or not. Every row represents the real classes and every column represents the predicted classes by the model. A pattern will be correctly classified if it is set on the diagonal (real class = predicted class). We can also take out some interesting ideas like false positives and false negatives. A false positive is a value which its real class is for example A and it is classified in class B, so predicted class B will have one pattern that is not correct. A false negative is a pattern which its real class is for example B and it is classified in class A, so real class B loses a pattern that will be a false positive for class A.

For example, in table 17 for class A, it will have 143 true positives, 7 false positives, 6 false negatives and the rest of patterns will be real negatives, because are classified as other class as A. This terms are studied with other metrics. In both table we can check that CCR are very high because patterns usually are true positives and the rate of false positives and false negatives with respect to the true positives is low, so we can say that are good classifiers.

	A	B	C	D	E	F
A	133	5	2	4	4	2
B	6	119	5	8	5	7
C	1	0	133	2	8	6
D	3	1	7	130	1	8
E	0	4	8	0	126	12
F	3	2	2	4	4	135

Table 17: Confusion matrix: MSE + Softmax

Finally, I have taken some incorrect classified patterns and I will show them to see if they are confusing. This patterns have been taken from a random seed with the best architecture and parameters according to CCR:

- Pattern 30 was classified in 4 when was 0 class.
- Pattern 48 was classified in 2 when was 0 class.
- Pattern 57 was classified in 3 when was 1 class.
- Pattern 64 was classified in 5 when was 1 class.
- Pattern 90 was classified in 0 when was 1 class.
- Pattern 181 was classified in 2 when was 3 class.
- Pattern 210 was classified in 2 when was 4 class.
- Pattern 219 was classified in 1 when was 4 class.
- Pattern 234 was classified in 1 when was 4 class.
- Pattern 260 was classified in 4 when was 5 class.

In the figure 3 we can see the image of this patterns to visually check if they are confusing. In general letters wrote in italic, or very thick or thin contour are incorrectly classified.

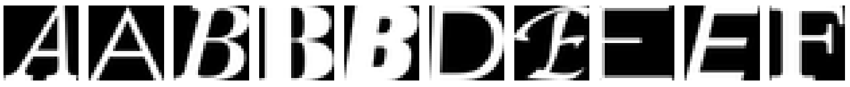

Figure 3: Incorrect Classified Patterns noMNIST

References

[1] Softmax vs Sigmoid function in Logistic classifier? URL: `https://stats.stackexchange.com/questions/233658/softmax-vs-sigmoid-function-in-logistic-classifier`. [Online. Last consultation: 23-10-2020].

[2] For Binary Classification use 1 or 2 output neurons? URL: `https://stats.stackexchange.com/questions/207049/neural-network-for-binary-classification-use-1-or-2-output-neurons`. [Online. Last consultation: 23-10-2020].