Artificial neural networks - Final project

May 29, 2018

NAME: *CRISTINA LUNA JIMÉNEZ*
STUDENT NUMBER: *r0693025*
PROGRAM: *ERASMUS*

Contents

1 Problem 1:nonlinear regression and classification with MLPs

1.1 Regression

The aim of this first section is to train a network with the capacity of approximate a nonlinear function that will be the following one:

$$f(X_1, X_1) = \frac{1}{25}[9f_1(X_1, X_1) + 6f_2(X_1, X_1) + +5f_3(X_1, X_1) + 3f_4(X_1, X_1) + 2f_5(X_1, X_1)]$$

In order to avoid correlation between samples, we divided the dataset into 3 in a random way but ensuring that the training had samples representatives of all the domain in which the data is distributed and for reducing the correlation between samples. In the following picture we can see the surface of the training set selected in the way described before:

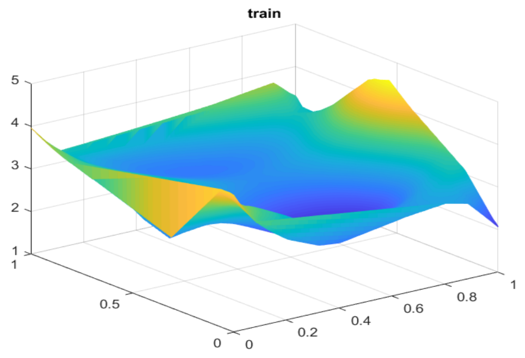


Figure 1: Training data points

In the experiments of this section, we used the following training algorithms: 'traingd', 'traingda', 'raincgf', 'traincgp', 'trainbfg', 'trainlm' and 'trainbr'. Additionally, we ran each network configuration for 10 times for obtaining meaningful results.

For the first test, we used 1 hidden layer architectures with 2 activation functions in that layer, specifically, we tested 'tansig' and 'logsig' because they are some of the most popular and suitable for regression problems, the output neuron was always purelin because the output of the regression function varies between the range [1.5 – 4] so if we would use 'tansig' or 'logsig' at the output, they won't be able to cover this range. The results obtained from these experiments were quite similar although 'tansig' seems to obtain lower MSE values in the validation set.

In the following picture we can see the MSE results comparison between training and validation test using 'tanh' for different number of neurons:

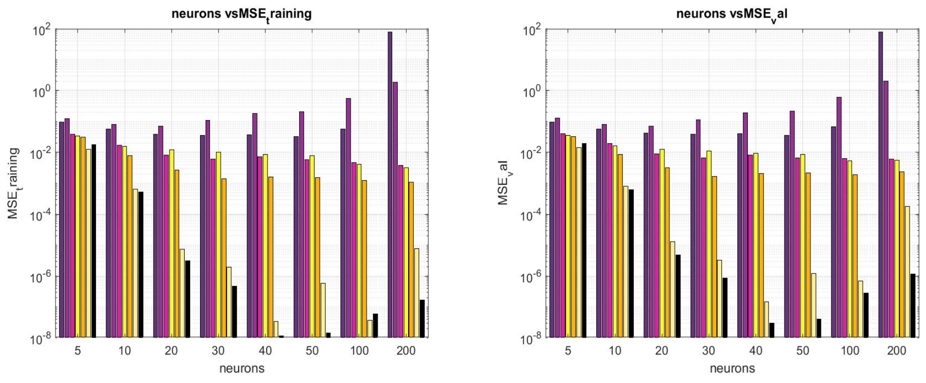


Figure 2: MSE training results(left) and MSE validation results(rigth)

According with the results, it seems that the best architecture for this problem is the obtained using 50 neurons, 'tansig' as activation function and 'trainbr' as learning algorithm.

In the following picture we can see its error curve for 200 epochs:

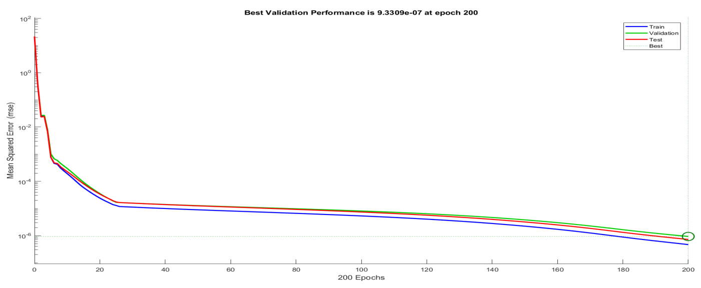


Figure 3: Error curve for training, validation and test sets

From the previous plot,it seems that the model has overfitting,, however as trainbr is an algorithm able to choose the regularization parameters, applying regularization constraints seems that is not coherent with it, so for reducing overfitting we could reduce even more the number of maximum epochs although it also means to reduce the 'accuracy' of the system because it hasn't reached a minimum yet and the MSE in training, validation and test set are still similar at 200 epochs ($MSEtraining$: $4.6864 * 10^{-7}$, $MSEvalidation : 9,3309 * 10^{-7}$, $MSEtest : 7,2317 * 10^{-7}$). For this reason, we consider this a good option of network and number of epochs.

Finally, in the following plot we can see the comparison between test values and network test predictions:

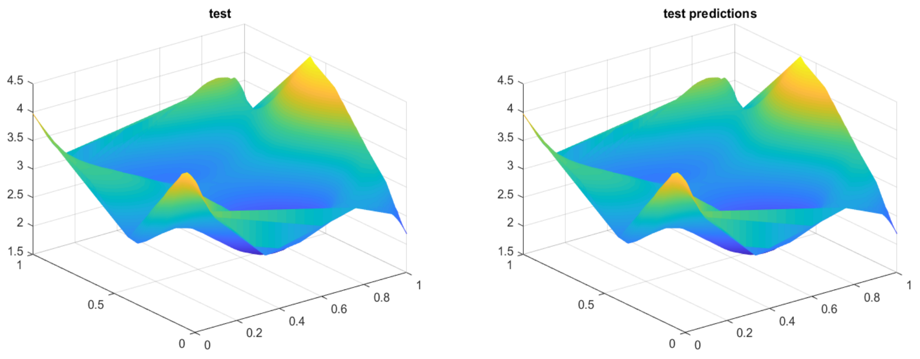
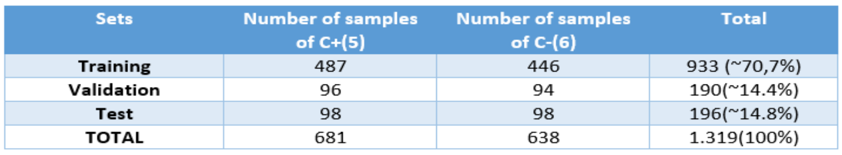


Figure 4: Comparative between original test data surface(left) and predicted test data surface(rigth)

1.2 Classification

The dataset that we have to wok with is the one which corresponds with digit 5, it means that we should train a network able to distinguish between class 5 and class 6.

In the following table, we can see the way in which we split the dataset after randomizing the samples:



Sets	Number of samples of C+(5)	Number of samples of C-(6)	Total
Training	487	446	933 (~70,7%)
Validation	96	94	190(~14.4%)
Test	98	98	196(~14.8%)
TOTAL	681	638	1.319(100%)

Figure 5: Division of the dataset into training, validation and test sets.

We have tried to maintain a balance between the number of positive and negative samples in all the sets for not training a model biased to one of the classes.

Regarding the attributes, we did an initial analysis of them by plotting their histograms with respect to both classes and we found that there is a strong overlapping between classes in all the attributes, below we can see and example of this for attribute 1 and 2. One of the things that we can notice when we study them is that they are neither normalized nor standardized, so we standardized all attributes to zero-mean and standard deviation 1.

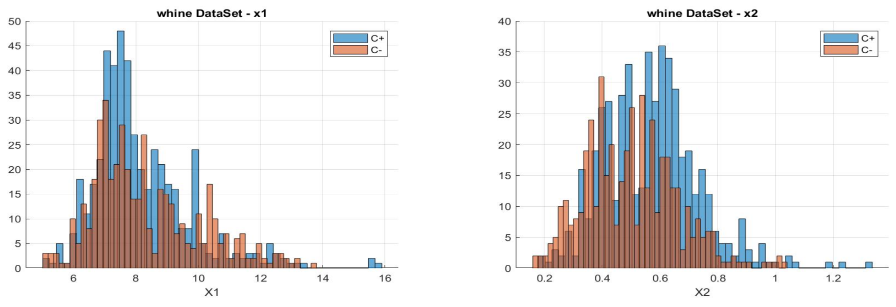


Figure 6: Comparative between original test data surface(left) and predicted test data surface(rigth)

With respect to the labels of the classes, we transformed them to 1-hot encoding in order to be able to use softmax as activation function of the last layer of 2 neurons. The reason of choosing 2 neurons at the output instead of 1 and this activation function is based on previous studies made in the classification field that reveal that this function usually gives good results and help to understand the outputs as a probability of belonging to one of the classes or the other, in addition to this, it also gives extra information about the reliability of the prediction.

All the experiments were tested using the following training algorithms: ‘traingd’, ‘traingda’, ‘raincgf’, ‘traincgp’, ‘trainbfg’, ‘trainlm’ and ‘trainbr’; and crossentroy as loss function (except for trainlm and trainbr, in which mse was used).

For the first experiment, we tested 3 activation functions for the hidden layers: ‘logsig’, ‘tansig’ and ‘poslin’(relu); using one hidden layer network with 5,10,20,30,40 and 50 neurons in each of the experiments. The results obtained shows that the worst function in terms of accuracy reached was the logsig. ‘tansig’ and ‘poslin’ results were similar although it seems that ‘poslin’ is more robust against overfitting although values are not enough meaningful as to affirm completely this fact. In terms of time they show also similar times in spite of the literature that assert that relu is faster in training so perhaps the reason is that the networks that we are using are not too complex for noticing it. For the reasons commented before, we used relu (‘poslin’) as activation function for the rest of the experiments.

In general, after inspecting the results, when we surpass 20-30 neurons overfitting appears and the results in the training set differ from the obtained in the validation set. Especially relevant is the case in which we use ‘trainlm’ as training algorithm, although it is also the one that obtains the best results.

For the second experiment, we tested if the use of more layers could improve the results, so we used 2 and 3 layers halving the number of neurons per layer with respect to the previous one. The initial number of neurons of the experiments were: 5(just for 2 layers),10,20,30, 40 and 50. The results show that the performance doesn’t increase in an important way so for this problem a one hidden layer network is enough, as the Hornik’s theorem establishes.

In Figure 7, we can see the results obtained for the validation set in experiments made with 1,2 and 3 layers’ networks and ’poslin’ as activation function.

At this point, the best networks found had 1 hidden layer, relu as activation function, ‘trainlm’ as training algorithm and a number of neurons between 20-50. As we had an overfitting problem, we tested these results using different values of regularization parameter higher than the used by default. After this experiment the network that had higher values in validation data was the one obtained using 50 neurons and regularization parameter equal to 0.01 but it seems to have still some overfitting (CCR training: 79.31%; CCR validation: 74,16%; CCR test: 77.35%).

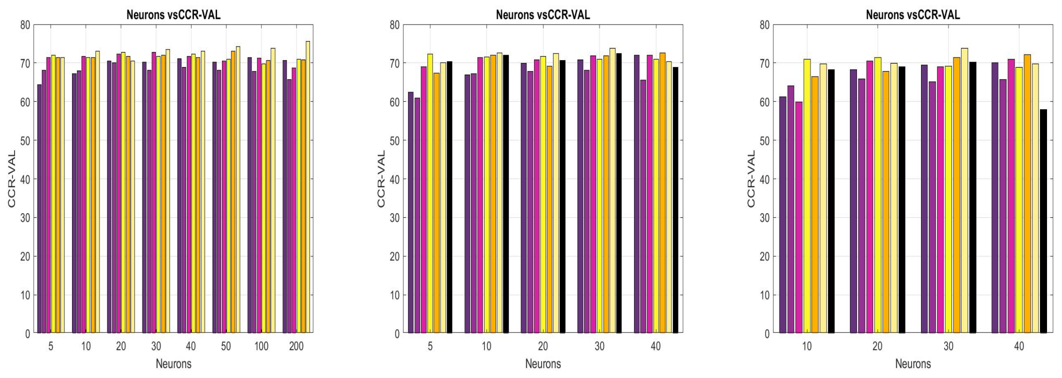


Figure 7: CCR in validation set for 1 hidden layer(left), 2 hidden layers(center)and 3 hidden layers(rigth)

Let's start with the dimensionality reduction tests. Before doing PCA, first we applied the same standardization process that we used in the previous section in the attributes because if we don't apply this standardization, results seem to be non-realistic due to the scale differences.

In the following picture we can see the percentage of information that each eigenvalue carries and the reconstruction error of the dataset in function of the chosen number of principal components. From the plot we can confirm the results obtained after analyzing the data at the beginning, that all the components carry a relevant amount of information, for this reason, we chose 8 components for maintaining around a 95% of the variance of the data, in this way we could remove 3 components and ensure good accuracy results in the training/testing stage.

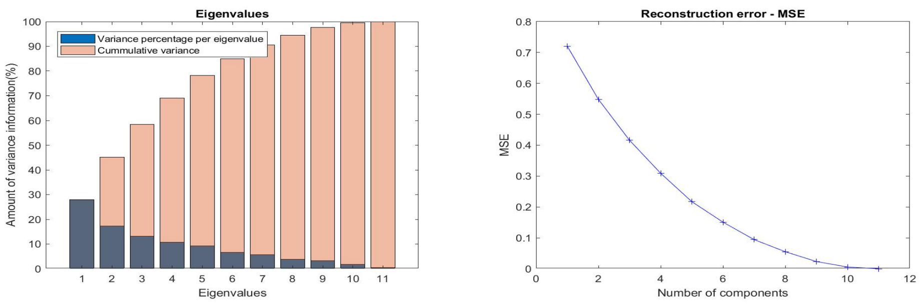
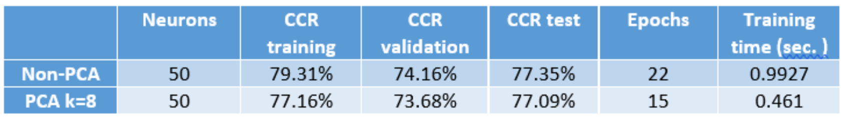


Figure 8: (Left) Eigenvalues with the percentage of information that carry and (Rigth) reconstruction error of the initial attributes after applying PCA.

The experiments performed were similar to the mentioned for the non-dimensionality reduction case so we will go directly to the conclusion of the best network. The best network obtained was one with 1 hidden layer, 50 neurons, relu as activation function and trainlm as training algorithm.

It seems that PCA helps to improve the time and the lost in accuracy is not too much, however in this case PCA application is not as important as in other problems in which there are correlation between parameters or in which most of the information is enclosed in few of them or even when the number of parameters is huge in comparison with the samples.

The summary of the best networks for PCA and non-PCA can be checked in this table, taking into account that both use relu as activation funcion and trainlm as learning algorithm:



	Neurons	CCR training	CCR validation	CCR test	Epochs	Training time (sec.)
Non-PCA	50	79.31%	74.16%	77.35%	22	0.9927
PCA k=8	50	77.16%	73.68%	77.09%	15	0.461

Figure 9: Best networks found and their performance.

2 Problem 2:character recognition with Hopfield networks

In this second part of the project, we are going to work with the capital letters of the alphabet and the first letter of my name in lowercase. In the following picture we can see some of the letter that conform the collection. Each of these characters is formed by 35(7x5) pixels and in total, the collection has 35 letters.

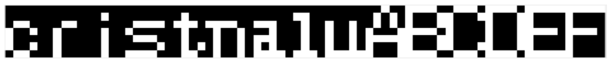


Figure 10: 15 first letter of collection

For the first experiment, we trained the Hopfield network with the whole collection and tried to recover the first 5 letters: 'crist' after adding noise to them. From the results we can see that in spite of being attractors, the net is not able to find them as equilibrium points when they have 3 pixels changed because there are other spurious states that are more similar to the noisy letters so the altered values converge to them. In the picture we can see one example and the spurious states that have appeared because we have tried to store too many patterns and we have exceeded the maximum storage capacity for retrieving them correctly.

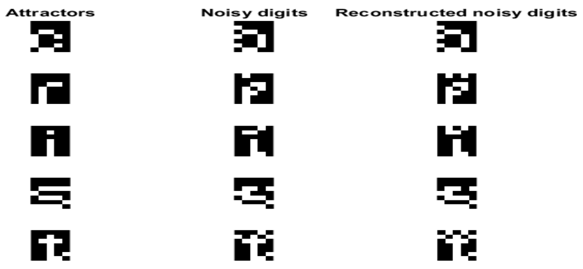


Figure 11: Reconstruction of noisy attractor images

The second task consists on finding the critical loading capacity of the network. In this task we trained the network increasing the number of patters that it had to save one by one to see in which moment errors started to appear. We tested different number of iterations, but we have plotted just the experiments based on higher number of them (500,750 and 1000) because they are the closest to convergence and the most trustable, however it is important to mention that if we decrease the number of iterations, the critical loading capacity could seem smaller.

In the following plots we have the mean percentage error in terms of pixels wrong and in terms of convergence to a difference attractor for 10 runs:

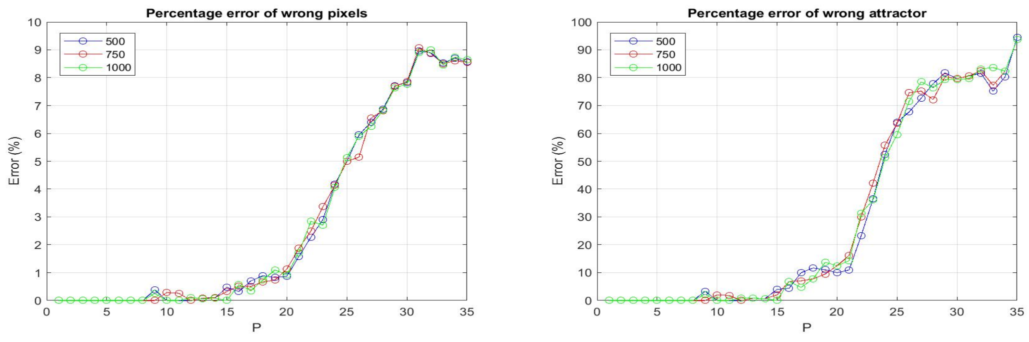


Figure 12: Percentage error for different number of patterns stored as attractors in the network.

It seems that the critical loading capacity is 8-9 patterns for a 35 neurons network. When the number of patterns that we try to storage is higher, we start to obtain errors in the reconstruction of the letter. Probably, if we define the initial collection of letters in a different way, the CLC would decrease because when we use the alphabet without the letters of my name, the CLC decreases until 4 for 1.000 iterations as we can see in the picture below:

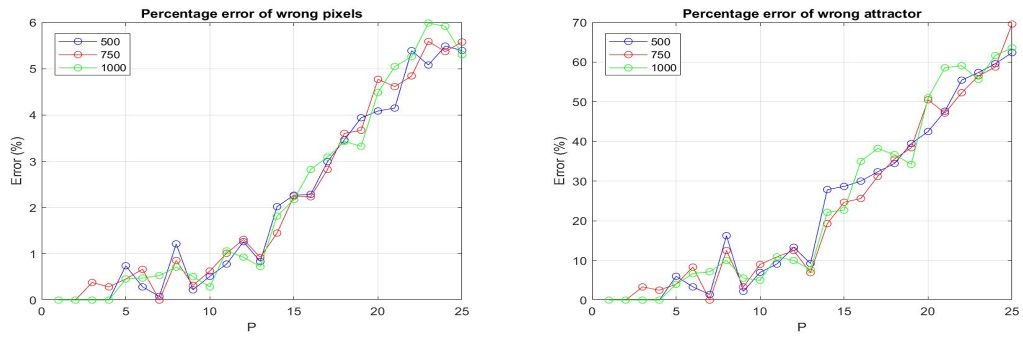


Figure 13: Percentage error for different number of patterns stored as attractors in the network without the lowercase letters.

According with the Hebb rule, the maximum storage capacity should be: $p_{max} = \frac{N}{4logN} = \frac{35}{4*log35} \approx 5$ patterns which is a little different to the one obtained in our first experiment, which means that the first letters in the alphabet are not too correlated. However, the Hebb rule bound is higher than the CLC obtained for the second case so it seems that capital letters keep more correlation than the lowercase letters and the results also reaveal that this bound is not a general rule, it is more an approximation. G.Wei and Z.Yu also show in their study that in some porblems, it is possible to not reach this limit and not obtaining a perfect retrieval of the corrupted data.

Regarding the spurious states, they are created always when we add a new pattern to the network although in some cases they are the same that other attractors orthogonal to the new introduced, but it doesn't happen in this collection thus, when we increase the number of patterns, the spurious are also kept and they reduce the storage capacity of the network.

Until this point we have demonstrated that for 35 neurons, the network is able to retrieve 8-9 patterns without error(using the whole collection), however if we want to store a higher number of letters/patterns, we should employ other techniques to accomplish this. In the previous mentioned publication of G.Wei and Z.Yu they speak about pseudo-inverse rule as alternative to Hebb rule that help to increase the maximum number of patterns stored in the network, however Matlab doesn't have the option of changing to this other learning rule so, in this project a simpler approach was implemented, one based on changing the initial images size of the collection(and thus the number of neurons of the network).

According with the Hebb rule, the minimum number of neurons that we need for storing 25 patterns and be able to recover them correctly are 250, although we will use 256 for doing the mapping easier. It is true that with our collection this bound could be lower, however we have declined for a more conservative sollution using as many neurons as Hebb rule establish.

The chosen option for the mapping was to double the size of the original image (from 7x5 to 14x10) and insert this image in the center of a black background of 16x16 pixels. With this option, we ensure that the letter images maintain their shape and we increase the difference between letters and thus the distance between attractors and their influence(or attraction regions).

In the following picture, we can see an example of the new letters:

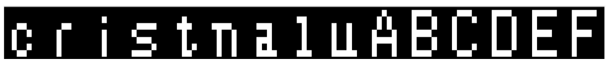


Figure 14: 15 first letter of the collection mapped into a 16x16 pixel images

As we have used a higher number of neurons that the required, the number of characters retrieved correctly is higher than 25 and thanks to the low correlation of the initial characters, this value is also higher than the Hebb rule. These results show that the 25-letter storage could be got using a lower number of neurons, perhaps with an image of 15x15 (225 px) could be enough. In the next plot, we have the mean percentage error for the new network of 256 neurons with CLC equal to 33-34:

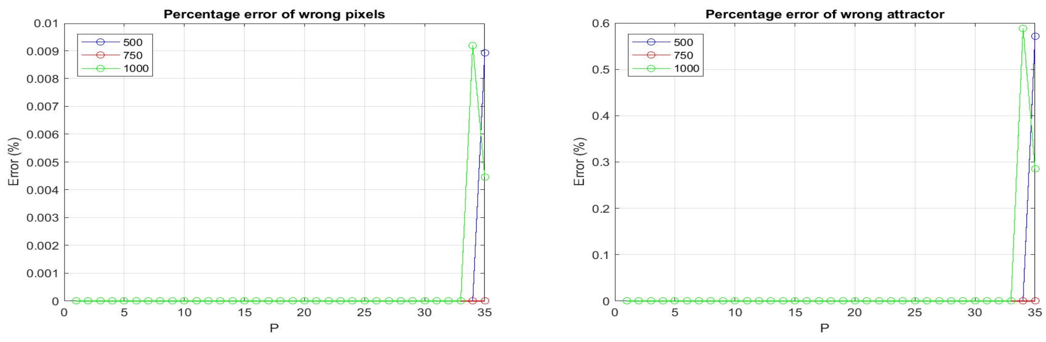


Figure 15: Percentage error for different number of patterns stored as attractors in the 256-neurons network.

References

[1] G.Wei & Z.Yu. Storage Capacity of Letter Recognition in Hopfield Networks

[2] Amos Storkey. Increasing the capacity of a Hopfield network without sacrificing functionality (1997).

[3] Yue Wu,J.Hu,Wei Wu,Y.Zhou & K.L.Du. Storage Capacity of the Hopfield Network Associative Memory (2012).

*** The draft code of the assignments and final project can be found in the following repository of Github:**

https://github.com/cristinalunaj/KUL-ANN.git