Artificial neural networks - Exercise session 2

Recurrent neural networks

Cristina Luna

April 30, 2018

1 Hopfield Network on handwritten digits dataset

Hopfield network is a type of RNN and it acts as an associative memory model, which means that is able to save patterns (that are stored as equilibrium points of minimum energy) thanks to the modification of its weight matrix and the state of the neurons that conform the network. Due to this memory, Hopfield networks can reconstruct new data that have been corrupted with noise and associate this data to one of the patterns saved in the network.

For this assignment, the patterns saved are vectors based on combination of -1s and 1s which represents the colors (white=1 and black=-1) of an image of 15x16 that conforms one number, these patterns are also known as attractors and they represents equilibrium states of the network.

Normally, the attractors have to be orthogonal between them because in that case, we reduce the crosstalk term and the attractors can be stored correctly in the network.

One of the drawbacks of the training is that when we want to maintain some attractors, it usually happens that their complementarities are also saved as attractors and other unwanted mixture states, these attractors are known as spurious states.

In spite of the good reconstruction ability of the network, in this exercise we have detected 3 cases in which the net is not capable of removing completely the noise that was corrupting the data, these cases are:

Few iterations: The number of iterations is not enough for reaching some of the attractors state so the reconstruction stay close but it is not exactly any of the attractors. An example of this case can be seen in the figure below, in which some of the pixels of the reconstructed image are a little darker than the real ones but the number has been reconstructed from its noisy version almost perfectly. $(Testfor : hopdigit\_v2(2,5))$. For higher number of iterations this effect disappears $(hopdigit\_v2(2,50))$.

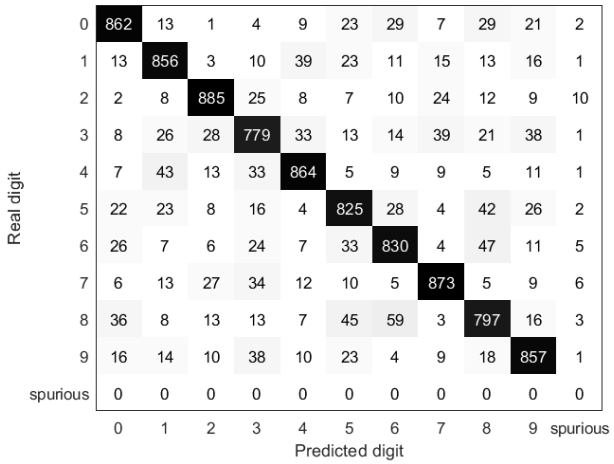
Noise high but many iterations: In this case it converges to an attractor but sometimes this is not the correct one, it is a different number. The reason is that when we add random noise, it alters some 'key' pixels of the image and the network is not able to recognize or associate the correct attractor and it finds another that is more similar for the corrupted number. $(Testfor : hopdigit\_v2(15,100))$

Spurious states: In some cases when the noise is high, the detected pattern is not one of the attractors but it is an equilibrium point that was generated during the training of the network and because of the attractors.In this situation, the original vector wit added noise gets stacked in one of this spurious states. In the figure below there is one example of a spurious state.

(a) Few iterations

(b) Noise high, many iterations

(c) Spurious states



Finally, we have the confussion matrix in which we can see the most confussed numbers afer 1.000 runnings of $hopdigit\_v2(5,500)$.

From this table we can see that the most similar number as 6 and 8 are mistaken more times than the ones that are more different as 1 and 6.

Additionally, it is interesting to see that most of the spurious states appears when we try to recognize the number 2 of the 7, it could means that there are most spurious states similar to these numbers.

2 Elman network on Hammerstein system data

In this second part, we performed some experiments with the goal of using an Elman network for mimic the behavior of the Hammerstein system. The parameters that were tested in them were: the architecture of the net (number of neurons and activation functions), number of epochs and number of training samples. All the configurations can be seen in the following table:

	Epochs	Training+ validation samples /Total samples	Test samples /Total samples	Num. of neurons (1st layer)	Act. Functions {1st layer, 2nd layer}
Exp.1	[100,250, 500,750, 1.000]	300/1.000	200/1.000	[1,2,3,4,5,6, 8,10,20,30]	{tansig,purelin}
Exp.2	500	[400,600, 800]/1.000	200/1.000	[1,2,3,4,5,6, 8,10,20,30]	{tansig,purelin}
Exp.3	[1.000, 10.000]	[850,2.125, 4.250,6.375, 8.500]/10.000	1.500/10.000	[1,2,3,4,5,6, 8,10,20,30]	{tansig,purelin}
Exp.4	750	600/1.000	200/1.000	[1,2,3,4,5,6, 8,10,20,30]	{tansig,tansig},{tansig,logsig}, {tansig,purelin},{logsig, tansig} {logsig, logsig},{logsig,purelin}

The common parameters that were not modified was the percentage of data used for training (70%) and validation (30%); and the number of repetitions that we run each configuration that was 10.

From the experiment 1,the most interesting result obtained is that when we increase the number of neurons, results get worse, it is especially notable for the case of 20 and 30 neurons, while for 1 to 8 neurons, results are in general good. The best results obtained in this case was for 3 neurons and 750 epochs (MSE test: 0.1768; MSE train: 0.1938; R test: 0.9257; R train: 0.9098).

If we analyze experiments 2 and 3, the results shows that if we increase the number of samples, we should increase also the number of epochs for obtaining better MSE results although it implies higher training times, as we expected.

Some of the results obtained that supports these affirmations are showed in the following table, for the case of 6 neurons.

Measures	10.000 epochs & 2.125 samples	1.000 epochs & 2.125 samples	500 epochs & 800 samples
R training	0.979	0.932	0.919
R test	0.977	0.927	0.898
MSE training	0.048	0.158	0.184
MSE test	0.059	0.163	0.245
Time (sec.)	40.6	4.49	1.19

Finally, the fourth experiment shows that some combination of functions perform better than others, the conclusions are that when we use tansig or purelin in the output layer results are much better than when we have the logsig at the output because the logsig function can return just positive values, although the best results are obtained when the logsig function is in the first layer, in most cases although differences are not too significative.