

Digits recognition via neural networks

Emilio González Montaña (egoeht69@hotmail.com)
at <http://emiliogonzalez.sytes.net>

2007/07/05

Abstract

The aim of this document is to study different methods to make digits pattern recognition via neural networks. In this article we will use two techniques: multilayer perceptron (MPL) and Kohonen map, two different ways of neural networking, one with driven learning (MPL) and the other not (Kohonen).

1 Introduction

1.1 Objective

The main goal is to make digit pattern recognition based in neural networks, to achieve this we will use two different networks.

The purpose to use two different learning methods (one driven and the other not) is checking if there is any advantage in one of them, comparing the results achieved by each of them.

1.2 Neural networks

Artificial neural network (ANN) is an interconnected group of neurons (figure 1), that with a mathematical model (normally simulated in a computer) make a connectionist computation [1]. The main advantage is achieve a masive parallel and distributed knowledge storage.

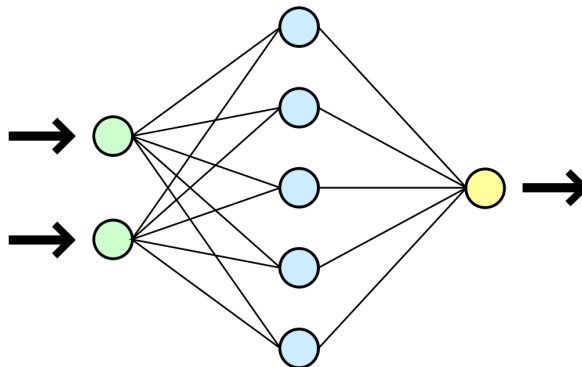


Figure 1: Example of artificial neural network

1.2.1 Multilayer Perceptron

The multilayer perceptron (MLP) is a network formed by one or more hidden neurons layers (figure 2), this allows the MLP to resolve non linear problems (like the XOR problem with simple perceptron). There are three types of layers (input, output and hidden layers) [2].

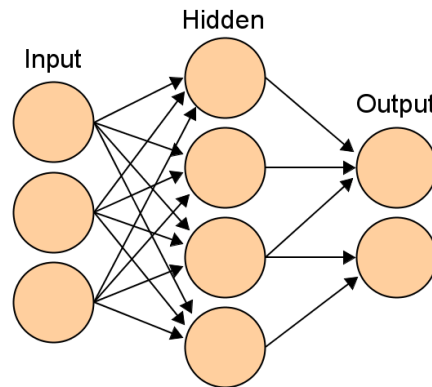


Figure 2: MLP topology

The learning is driven and is based into the error backpropagation, where the error between the output and the expected output is propagated from the output layers to the input layers (through the hidden layers), correcting the neurons weights, so the neural network learn the input patterns step by step.

1.2.2 Kohonen maps

Self organizing map (SOM) or Kohonen map is a not driven learning network, distributed in a grid, the input layer is fully connected to the output layer (figure 3) [3].

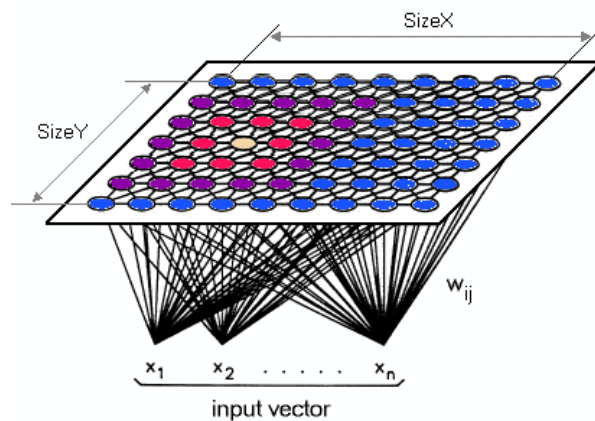


Figure 3: Kohonen topology

The idea is specialize each output neuron in a type or class of input, each time we supply an input pattern to the network, we choose the most excited neuron, and we make it the winner, and make it be more near to the input, so the next time that neuron is more probably be the winner again, and it will recognize the input pattern. The learning is not driven because we don't correct or tell the neuron what is the expected output or what has the error been.

2 Data

The data set is formed by two text files, each of one has a list of digits information, each digit information has two things: one is the array of pixels with the digit (5x8 pixels, figure 4) and the expected output (10 digits \Rightarrow 10 outputs).

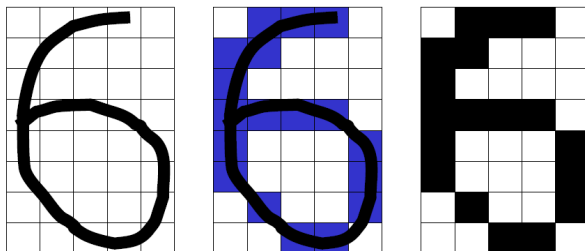


Figure 4: Digit grid

We use two different data sets:

- One for training the networks (270 digits, 27 of each type).
- Other for validating the networks (70 digits, 7 of each type).

So, we avoid to memorize the input patters (over-training) and don't lose the generalization capacity.

3 ANN implementation

We have implemented the ANN in C++ language, without using any kind of library except the ANSI C libraries for file input/output.

The main purpose of this election is to manage completely the solution and to gain knowledge into the ANN basic concepts. Other main issue is to achieve more proficiency that in other language libraries (JAVA, matlab, ...).

We have generated different programs to each type of ANN, based all of them in a our ANN common libraries.

4 Multilayer perceptron

The program was parametrized to use a range of number of hidden neurons, and other range of learning iterations (epoques). The internal parameters used into the MLP were: memento 0.95 and learning factor 0.05.

The results are compared in the figure 5.

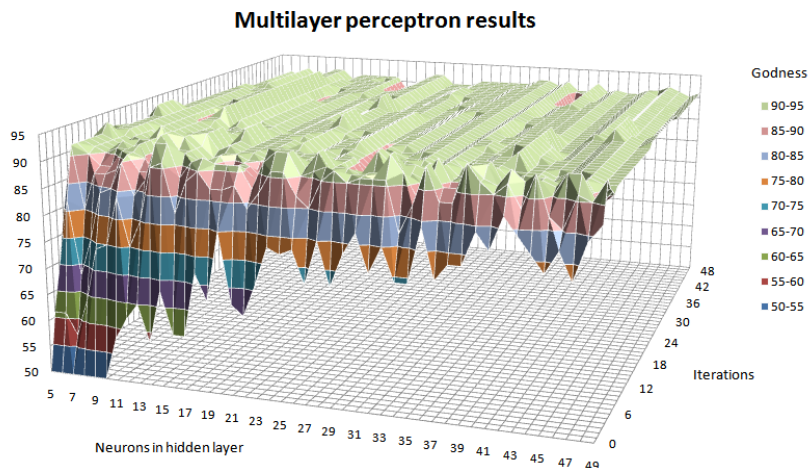


Figure 5: Perceptron results comparing hidden neurons count with epochs

Analyzing the results (in graphic and numeric table), we optime value for the hidden layer neurons count was encoutered around 33, and all the networks converged around 10 to 15 iterations.

5 Kohonen map

The parametrization of this program was made to change the dimensions of the output layer neurons grid. The internal parameters used into the map were: learning factor 0.25 and 15 iterations.

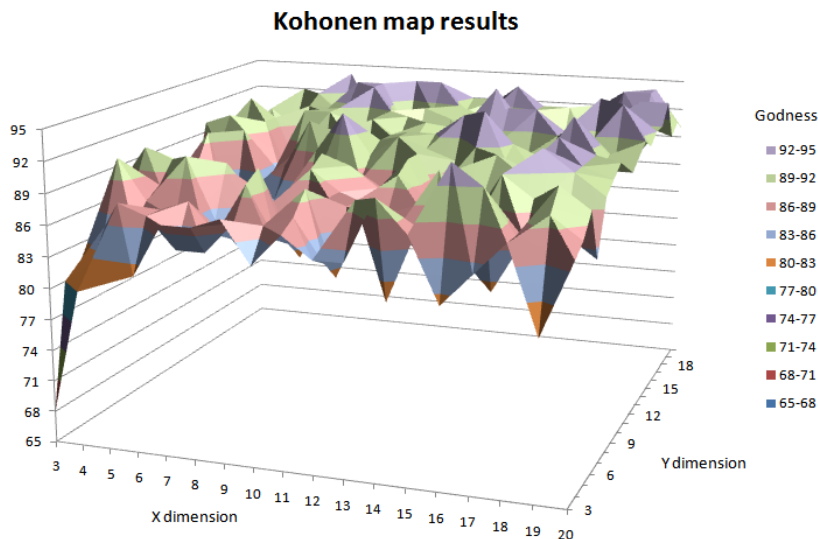


Figure 6: Perceptron results comparing width with height

The results are compared in the figure 6.

The results shows a good tendency when the dimensions grows, but better tendency when the width and height are not equal (a relation between of 1 to 2 and 1 to 1.5). The rectangular grids are common in Kohonen maps, being better than square grids.

An optime value of the dimensions will be arround: 9x6.

References

- [1] Christopher M. Bishop: *Neural Networks for Pattern Recognition*, Ed. Oxford University Press, 2000.
- [2] Simon Haykin: *Neural Networks : A Comprehensive Foundation*, Ed. Prentice Hall, 1999 (Second edition)
- [3] Ben J.A. Krose and P.Patrick van der Smagt: *An Introduction to Neural Networks*, University of Amsterdam, Faculty of Mathematics and Computer Science (The Netherlands). Eighth Edition, November 1996.