Recurrent neural network for positioning purposes

Noêmio Xavier da Silveira Filho [1]
Silas Bianchi [1]
[1]Laboratory for Computing and Applied Mathematics - LAC
Brazilian National Institute for Space Research - INPE
C. Postal 515 – 12245-970 – São José dos Campos - SP BRAZIL
noemio@noemio.com | silasbianchi@gmail.com

Abstract

Several different neural networks architectures has been implemented last years, but in spite of its innovative approach, the major part of them are variations about the classical Rosenblatt's multi layer perceptron (MLP) described in Rosenblatt (1962) and historically registered as a discovery of an effective general method of training a multilayer neural network that works and can be used to solve problems in many areas. The recurrent neural network (RNN) can be recognized as a standard backpropagation MLP with at least one hidden unit, but it is different in some way because at the same time that the RNN runs its learning loop it needs some kind of interaction with the physical world, so it needs a break to stop working in a software way (in a logical mode) and operate some type of hardware interaction (in a physical mode). Thus, the RNN runs its learning time step by step with an intermediate break to pick up new informations from some kind of physical device. The goal of this project is to demonstrate that a RNN connected with a positioning device can improve the resolution of its movements until achieve the optimization of the physical limits of its capacity, even passing from a macropositioning to a micropositioning and, finally, to a nanopositioning levels (if the physical device supports).

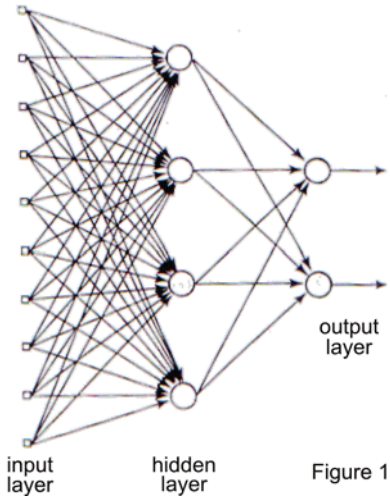

Figure 1

First of all, we need to understand the fundamentals of a neural network that works with MLP backpropagation algorithm, and is able to learn some kind of process to achieve some kind of target results. The MLP (represented by figure 1), in short, works as consequence of four functional fundamentals (Fausett, 1998, p. 190), as follows:

1) The multilayer architecture, with one layer of input values, at least one layer of hidden units also may have bias and one output layer. The bias applied on hidden and output layers acts like weights to adjust the flow of information.

2) Feed forward, the combinatory process of activation of units.

3) Backpropagation of the associated error, by comparing process of activation value and desired value to achieve.

4) Adjustment of weights, by the application of correction values for the next cycle.

The algorithm, here, is a little different in time of training and time of working process in the cycle of life of a neural network.

Inside each cycle of training process loop, the recurrent neural network has the same algorithm as an standard MLP, but its graphic representation show the intermediate time to stop and pick a new information, a new input value, from the physical world, as shown in figure 2. This new information seams that the physical world have been changed since the previous time. Logically this happened because we are working with physical movements in a positioning system, and the physical mechanism has been act in its tasks, in a sequence of its times. Thus, the RNN in its several times, t1, t2, t3... have an intermediate stop time, that is different as the final stop condition in the end of training process. This routine is able to learn how to optimize the resolution in a model of learn process that seams: learning as a quantitative increase in knowledge, learning as acquiring information, learning as memorizing, learning as storing information that can be reproduced, and learning as acquiring facts, skills, and methods that can be retained and used as necessary.

Now, the backpropagation of the associated error must be calculated by comparing the coordinates of the target point with the coordinates of the really achieved point. The difference is the associated error that can be corrected in small steps during the next times of systemic loop.

In this manner, the positioning problem works with a standard geometrical coordinate system which can be defined by its values, like this:

$O(x_o, y_o, z_o)$

$D(x_d, y_d, z_d)$

Where x_o, y_o and z_o are the coordinates of origin point and x_d, y_d and z_d are the coordinates of destiny point. Computing the differences between achieved values after each step of real motion and desired values for the same step we can have the value of error, and we can calculate the value of corrected weights of the next step. One of the most typical activation function to do this is the binary sigmoid function, wich has the range of (0, 1) and can be defined as:

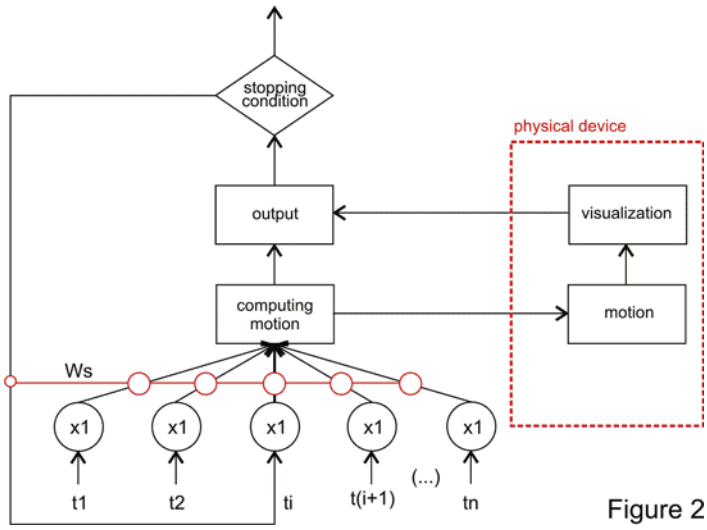

Figure 2

$$f(x) = \frac{1}{1 + \exp(-x)}$$

The algorithm given here can take the advantages of a sigmoid function when apply the correction of weights in a soft way, rather than a rough way, the second, maybe the worst way to do adjustments in a precision control system.

Of course, this algorithm depends on an accurate system of visualization of geometrical results in the physical device. In case that this shouldn't happens we have a physical indetermination of learn possibilities of our system. This is the main difference between standard MLP and RNN, because the first can finish its combinatory calculation without any interaction with physical world.

The RNN architecture can be useful to control positioning devices which depends on high level precision. In addition the RNN can be also useful to determine the level of resolution of a considered device. This can be implemented by a configurable attribution of values to acceptable errors between desired and achieved. So, we can discover the physical limits of resolution offered in this specific hardware.

References

Berg, M. (Et.al.). (1978). *Computational Geometry, Algorithms and Applications*. Springer-Verlag Berlin Heildelberg New York.

Bishop, C. M. (1996). *Neural networks for Pattern Recognition*. Oxford University Press.

Chen, J. (1996). *Computational Geometry, Methods and Applications*. Texas A&M University.

Du & Hwang (1992). *Computing in Euclidean Geometry*. World-Scientific.

Fausett, L. (1994). *Fundamentals of Neural Networks*. New Jersey, Prentice-Hall.

Ginsberg, M. (1993). *Essentials of Artificial Intelligence*. San Francisco, M. Kaufmann Pub.

Goodman & O'Rourke (1997). *Handbook of Discrete and Computational Geometry*. Boca Raton New York, CRC Press LCC.

Hayken, S. (1998). *Neural networks, a Comprehensive Foundation*. Prentice-Hall.

O'Rourke, J. (1999). *Computational Geometry in C*. Cambridge University Press, Cambridge.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. New York. Spartan.

Tsoukalas, L. H. & Uhrig, R. E. (1997). *Fuzzy and Neural Approaches in Engeneering*. New York, John Wiley & Son Inc.