Proceedings of the 5th WSEAS Int. Conf. on Electronics, Hardware, Wireless and Optical Communications, Madrid, Spain, February 15-17, 2006 (pp64-69)

# From a New Way to Show the Universality of a Two-Layer Perceptron to a New Approach to Digital Design

JOSÉ BARAHONA DA FONSECA
Department of Electrical Engineering and Computer Science
New University of Lisbon
Monte de Caparica, 2829-516 Caparica
PORTUGAL
http://www.dee.fct.unl.pt

*Abstract:* - We show in a constructive way the universality of a two layer perceptron. We show how to build any minterm of any logical function with one perceptron and how to build the OR of all the minterms with another perceptron. This way it is possible to implement a logical function with *N* minterms with a two layer perceptron with *N*+1 perceptrons. Next we addres the question of minimizing the number of perceptrons or threshold elements to implement a given logical function. Here we have to answer to two open questions that have not yet answered in the literature:1) Is the logical function linearly separable? 2) If the logical function is linearly non-separable how to minimize the number of perceptrons or threshold elements to implement the logical function? If we know that the logical function is linearly separable it can be implemented by one perceptron trained with the Rosenblatt's Perceptron Learning Rule [1]-[2] that guarantees convergence in the case of the existence of a solution. We propose as a test of linearly non-separable logical function the Perceptron Learning Rule did not converge after a very great number of epochs or iterations. As a method of minimization of the number of perceptrons or threshold elements to implement a linearly non-separable function we propose a trial and error procedure that generates all manners to implement the logical function with the mnimum number of perceptrons or threshold elements, and if for all of them the learning rule did not converge for all associations of minterms then we must increment the number of perceptrons and generate and test again all manners of implementation of the logical function till we got a solution. Finally we present some examples of linearly non-separable logical functions that we implemented with a minimum number of perceptrons or threshold elements.

*Key-Words:* - Logical Function, Minterms, Linearly Non-Separable Logical Function, Perceptron Learning Rule, Methodology to Implement a Linearly Non-Separable Logical Function with the Minimum Number of Perceptrons or Threshold Elements.

## 1 Introduction

This work arises as a consequence of one of our classes of *Neural Networks* where we show to our students that it is possible to implement any logical function, even linearly non-separable, with a two layer perceptron. Then at home we read what we have done and said to ourselves: 'Heureka! This is a new approach to Digital Design!'. Then we made some literature research and found that our idea is not original. In 1988 Yon B. Cho and Yoshiyasu Takefuji [3] developed complex logical circuits, including synchronous circuits, with only operational amplifiers, resistances and capacitors. More recently with the advent of VLSI and RTD circuit design the idea of implementation of logical functions based on threshold elements, which are nothing more than Rosenblatt's perceptrons, have been developed and also very few design methodologies have been proposed [5]-[10]. By the contrary there are a lot of proposals of methodologies to implement linearly non-separable logic functions with Rosenblatt perceptrons [11]-[25]. So why to propose one more methodology? Because our methodology guarantees always a minimum number of perceptrons or threshold elements. As a curiosity we found an old book published in 1971 about applications of Threshold Logic [4] that seems to ignore the works of Rosenblatt published in 1958 and 1961. Another curious fact is that Valeriu Beiu in [7] *seems* to ignore that the logical functions A>B and A≥B are linearly separable logic functions and so can be implemented by only one perceptron or threshold element and A=B may be defined by not(A>B) AND not(A<B). For example, for four bits words, we found after 15 epochs the solution

$\mathbf{W_1}$=[16 8 4 1 -16 -8 -4 -3],

$\mathbf{b_1}$=-1,

departing from zero weights. In figure 1 we show the evolution of the accumulated error over one epoch using the Perceptron Learning Rule [1]-[2]. For the function A≥B we found after 13 epochs the solution

$\mathbf{W_1'}$= [16 7 3 1 -16 -7 -3 -1],

$\mathbf{b_1'}$=0.

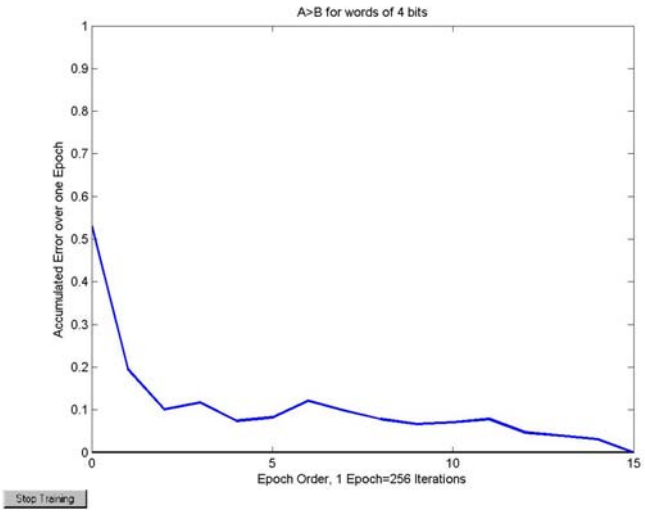


**Fig. 1**- Accumulated error over one epoch for A>B for words of 4 bits length. We got one solution in only 15 epochs, 15x256=271 iterations.

And for 5 bits words we got the solution for A>B given by

$\mathbf{W_2}$=[32 16 8 4 2 -32 -16 -8 -4 -3],

$\mathbf{b_2}$=-1

after only 21 epochs as you can see in figure 2. For the function A≥B we found

$\mathbf{W_2'}$=[22 11 6 3 1 -22 -11 -6 -3 -1],

$\mathbf{b_2'}$=0

after only 8 epochs.

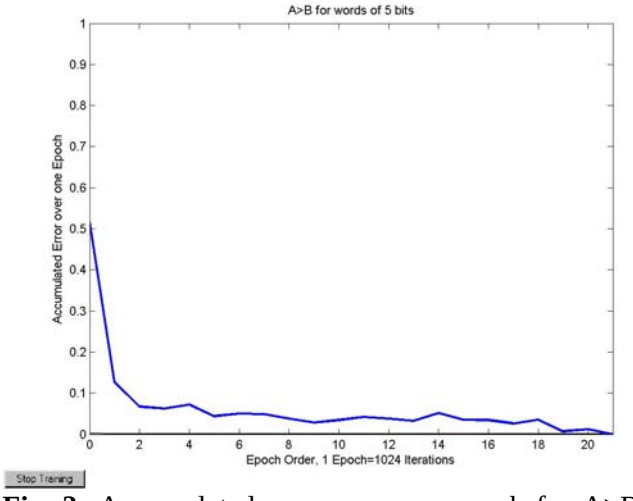


**Fig. 2**- Accumulated error over one epoch for A>B for words of 5 bits length. We got one solution after only 21 epochs, 21x1024=21504 iterations.

And for 6 bits words we got the solution for A>B given by

$\mathbf{W_3}$=[97 48 24 11 5 2 -97 -48 -23 -11 -5 -3],

$\mathbf{b_3}$=-2

after only 119 epochs as you can see in figure 3. For the function A≥B we found after 116 epochs the solution

$\mathbf{W_3'}$=[100 49 25 12 5 3 -100 -49 -25 -12 -6 -4]

$\mathbf{b_3'}$=2

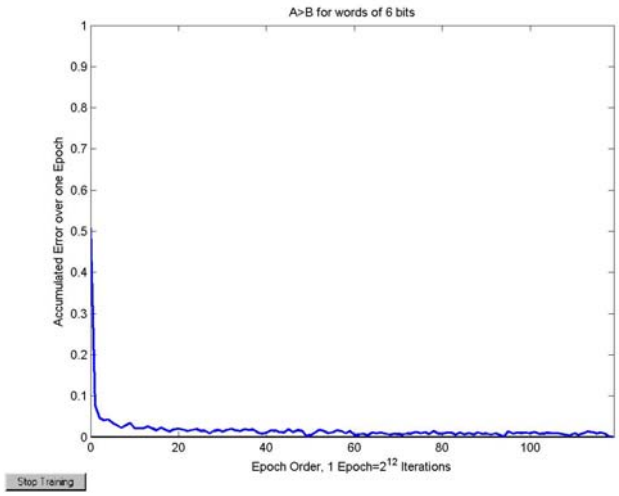


**Fig. 3**- Accumulated error over one epoch for A>B for words of 6 bits length. We got one solution after only 119 epochs, 119x$2^{12}$ iterations, in about 26 minutes with the Matlab *nntool* over a PC with a clock of 1GHz.

And for 7 bits words we got the solution for A>B given by

$\mathbf{W_4}$=[195 98 49 24 11 6 2 -195 -98 -49 -24 -11 -5 -3],

$\mathbf{b_4}$=-2

after only 235 epochs as you can see in figure 4.

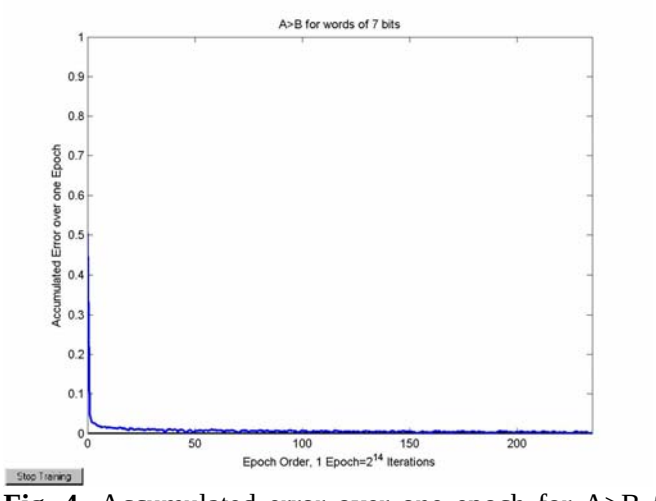


**Fig. 4**- Accumulated error over one epoch for A>B for words of 7 bits length. We got one solution after only 235 epochs, 235x$2^{12}$ iterations, in about 4 hours with the Matlab *nntool* over a PC with a clock of 1GHz.

For 8 bits we got the following solution of A>B after 594 epochs and about 3 days of runtime with matlab 7 over a Pentium IV with a clock of 3.6GHz,
**W**=[427 213 107 54 27 12 7 3 -427 -213 -107 -54 -27 -12 -5 -3],
**b**=-3.
It *seems* that for any number of *N* bits the function A>B is linearly separable. Next we will show in a constructive way the following result:

**Theorem 1.** The function A>B for words of any number of *N* bits is linearly separable.

Is simple to verify that the perceptron with

**W**=[48 24 12 6 3 2 -48 -24 -12 -6 -3 -1]
**b**=-2

implements the function A>B for words of 6 bits and also

**W**=[96 48 24 12 6 3 2 -96 -48 -24 -12 -6 -3 -1]
**b**=-2

implements the function A>B for words of 7 bits, and so on, so this way we can construct any comparator of any *N* bits with only one perceptron, as we wanted to show. The general expression of the weight vector is

**W=**[$2^{N-4}$x6 $2^{N-5}$x6 ... 6 3 2 -$2^{N-4}$x6 -$2^{N-5}$x6 ...-6 -3 -1]

Of course there are much more ways to build a comparator with only one perceptron, but it is enough to find one general expression to prove theorem 1. Next we will show the following similar result:

**Theorem 2.** The function A≥B for words of any number of *N* bits is linearly separable.

Is simple to verify that the perceptron with

**W**=[28 14 7 3 2 -28 -14 -7 -3 -1]
**b**=0

implements the function A≥B for words of 5 bits and also

**W**=[56 28 14 7 3 2 -56 -28 -14 -7 -3 -1]
**b**=0

implements the function A≥B for words of 6 bits, and so on, so this way we can construct any comparator of any *N* bits with only one perceptron, as we wanted to show. The general expression of the weight vector is

**W=**[$2^{N-3}$x7 $2^{N-4}$x7 ... 7 3 2 -$2^{N-3}$x7 -$2^{N-4}$x7 ...-7 -3 -1]

Nevertheless it remains two open questions to be answered satisfactorily:

1) How to Identify a Linearly Non-Separable Logical Function?

2) How to Minimize the Number of Perceptrons or Threshold Elements to Implement a Linearly Non-Separable Logical Function?

In section 2 we make the constructive demonstration of the universality of a two layer perceptron, in section 3 we enunciate our methodology to minimize the number of perceptrons or threshold elements to implement a linearly non-separable logical function and in section 4 we apply it to some simple linearly non-separable logical functions and finally in section 5 we present our conclusions and possible research vectors that may define the direction of our future work that surely will include the application of evolutionary computation techniques.

## 2 Demonstration of Universality of a Two Layer Perceptron

It is simple to show that any minterm with *N*1 positve variables, assuming that they only may be 0 or 1, and *N*2 negated variables can be implemented by a perceptron with weight +1 attributed to all positive inputs/variables and with weight –1 attributed to all negated inputs/variables and bias=-*N*1. And is also simple to show that a *N* input perceptron with all weights +1 and bias=-1 implements the logical OR of all the *N* input variables.

With these two types of perceptrons we may implement any *N* minterm logical function with *N*+1 perceptrons.

Furthermore if the logical function is linearly separable it can be implemented by only one perceptron and its weights and bias obtained with the Perceptron Learning Rule [1]-[2]. If it is linearly non-separable in most cases we can make associations of minterms that result in linearly separable logical functions and so implement it with less perceptrons or threshold elements than *N*+1, *N* being the number of minterms of the lineraly non-separable logical function.

## 3 Methodology to Minimize the Number of Perceptrons or Threshold Elements

To our knowledge nobody before us did solve this problem of minimizing the number of perceptrons or threshold elements to implement a linearly non-separable function.

First we must identify a linearly non-separable logical function. We consider that if after 100 million of epochs the algorithm of Perceptron Learning Rule [1]-[2] did

not converge then the logical function is *considered* linearly non-separable.

Next if the logical function *seems* to be linearly non-separable then we must begin to generate all the possible minterm associations that can be implemented by two perceptrons, i.e., to group the minterms in two associations, test for each manner to realize this grouping if the two logical functions are linearly separable and if for all the possible manners to group the minterms in two perceptrons they did not converge then we must go to the hypothesis of grouping the minterms in three perceptrons or threshold elements and so on, till we find a solution.

This methodology seems simple but for a linearly non-separable logical function with a lot of minterms it will take a lot of computation to find a good solution. In the near future we will try to improve this methodology in terms of its runtimes.

## 4 Three Examples of Application

The 3 variable logical function $f=m_1+m_2+m_6+m_7$ *seems* to be linearly non-separable but the association of the following minterms is a linearly separable logical function $f_2=m_2+m_6+m_7$ and it remains only $m_1$. So $f_2$ could be implemented by the following perceptron **W**=[-2 4 2] and **bias**=-4 and $m_1$ by **W**=[-1 -1 1] and **bias**=-1. Even for this simple example this took about 2h in a PC at 1GHz.

Another more complex example of a 3 variable logical function that *seems* to be linearly non-separable is $f=m_0+m_1+m_2+m_4+m_6+m_7$, with 6 minterms. After about 8h of computation we found the following *optimal* solution: $f_2=m_0+m_1+m_2+m_4$ and $f_3=m_6+m_7$ that could be implemented by $\mathbf{W_2}$=[-1 -1 -1], $\mathbf{bias_2}$=+1 and $\mathbf{W_3}$=[0 1 2], $\mathbf{bias_3}$=-3.

Finally for the following four variable logical function with eight minterms, $f=m_0+m_1+m_5+m_6+m_8+m_9+m_{14}+m_{15}$, that *seems* to be linearly non-separable, we found that $f_1=m_0+m_1+m_8+m_9$, $f_2=m_6+m_{14}+m_{15}$ and $f_3=m_5$ are linearly separable and may be implemented by three perceptrons with weights $\mathbf{W_1}$=[0 -2 -2 0], $\mathbf{b_1}$=0, $\mathbf{W_2}$=[-3 4 6 3], $\mathbf{b_2}$=-10, $\mathbf{W_3}$=[1 –1 1 –1], $\mathbf{b_3}$=-2, respectively. This way this complex function could be implemented only by 4 perceptrons or threshold elements.

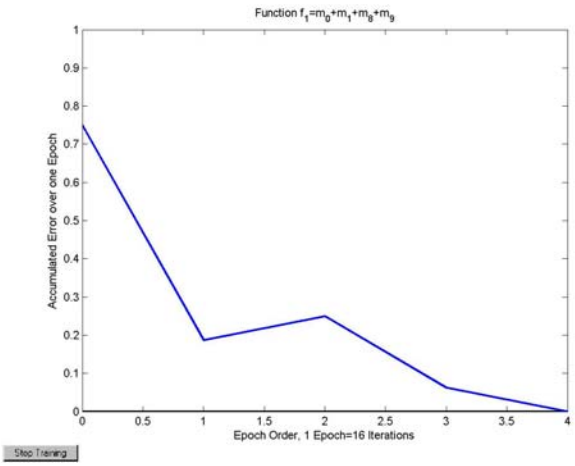


**Fig. 5**- Evolution of accumulated error over one epoch for function $f_1$. In only 4 epochs we got one solution.

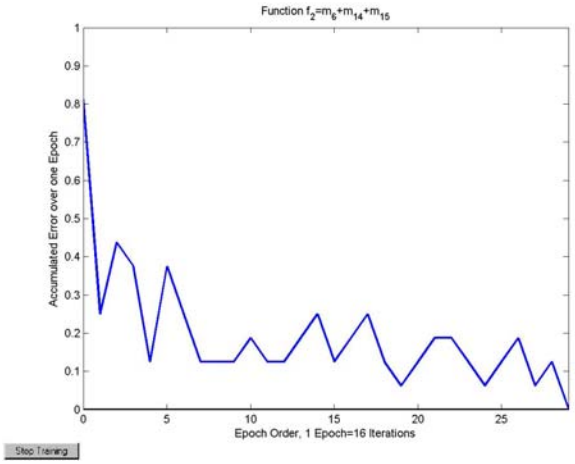


**Fig. 6**- Evolution of accumulated error over one epoch for function $f_2$. Although with less minterms we got one solution only after 29 epochs or 29x16=464 iterations.

It *seems* that the main drawback of our methodology is its bad runtimes and the evolutionary computation techniques may have a word to say in its improvement.

## 5 You May Say…

Of course we may solve these problems with sigmoides in the first layer and a linear neuron in the second layer and then use the backpropagation algorithm.

For the third example of the previous example with only three sigmoides we got a very good approximation after 1000 epochs (see figure 6) and the weights and bias of the first layer were

$\mathbf{W_1}$= [0.016061 10.9542 -10.9542 -0.01518
-13.5412 14.0067 0.00015162 13.6972
-5.8865 -25.3245 25.3248 5.8302]

$\mathbf{bias_1}^T$=[ 0.28408 14.4743 5.5862]

and the weights and bias for the linear neuron were

$\mathbf{W_2}$=[1.7643 -3.5428 1.7643], $\mathbf{bias_2}$=1.7785, for a random initialization of the weights and bias. After only 1000 epochs we got a *mse* as low as $3.5\ 10^{-13}$.

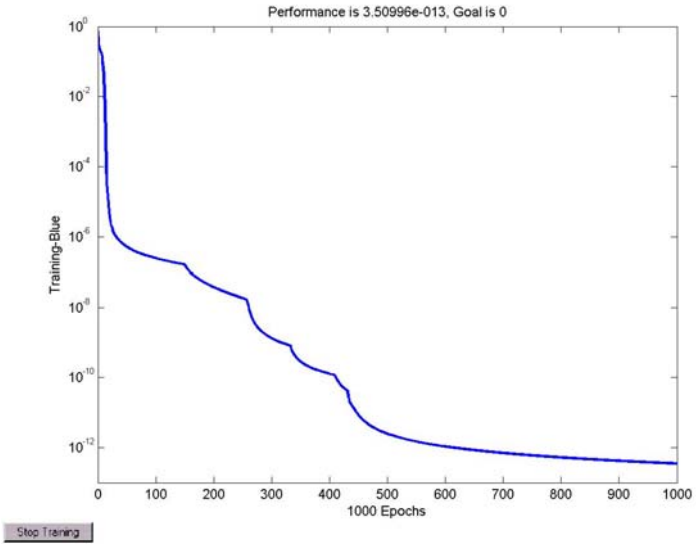


**Fig. 7** Evolution of *mse* versus the number of epochs using standardd backpropagation.

So with only 4 neurons, 3 sigmoides and one linear neuron we could make the same function that was implemented with 4 perceptrons in the previous section. Nevertheless it is much more difficult to implement a good sigmoidal neuron than a perceptron with hardware and to implement fractional or real weights than integer weights.

It would be nice if we can *force* big absolute values in the weights and bias to facilitate the *translation* from a *sigmoidal* network to a perceptron network. For example, from the two variable XOR solution with a two layer perceptron,

$\mathbf{W_1}$=[1 -1
-1 1]
$\mathbf{bias_1}^T$=[-1 -1]
$\mathbf{W_2}$=[1 1], $\mathbf{bias_2}$=-1, we got the sigmoidal network (sigmoides in the first layer and a linear neuron in the second layer) defined by
$\mathbf{W_1}$=[200 -200
-200 200]
$\mathbf{bias_1}^T$=[-100 -100]
$\mathbf{W_2}$=[1 1], $\mathbf{bias_2}$=0.

## 6 Conclusions and Future Work

Our methodology of identifying and implementing linearly non-separable logical functions with a minimum number of perceptrons or threshold elements is very simple but has two big drawbacks: 1) We are never *sure* if our logical function is *really* linearly non-separable; 2) Its exhaustive search from the possible ways to associate the minterms turns it very slow and for more than 6 minterms the runtimes tend to be very big (more than 10h for a PC at 1GHz).

In the near future we are planning to compare our algorithm to the few published in the literature of VLSI design [5]-[10], and to the numerous published in the literature of Neural Networks journals and conference proceedings [11]-[25] based on a test set of complex linearly non-separable logic functions.

A possible way of evolution of our work is to try to implement our methodology with the aid of evolutionary techniques to improve its runtimes. Nevertheless there are a lot of difficulties to implement this approach namely the definition of a good *fitness* function that will rank the population of candidate solutions by their *separability*, i.e. how far the associations of minterms are from linearly separable logical functions and the number of perceptrons or threshold elements necessary to implement them.

## Acknowledgements

This work would not be possible without the aid of the powerful tool 'b-on', www.b-on.pt, subscribed by our university, that gave us the access to the full version of papers published in a lot of resources and journals, like the IEEE Xplore.

*References:*

[1] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain", *Psychological Review*, Vol. 65, pages 386-408, 1958.

[2] F. Rosenblatt, *Principles of Neurodynamics*,Washington DC, Spartan Press, 1961.

[3] Y. B. Cho and Y. Takefuji, "Analog Circuit Design Based on Neural Networks", in *Proceedings of the IEEE Twentieth Southeastern Symposium on System Theory*, pages 100-105, IEEE, 1988.

[4] S. Muroga, *Threshold Logic and its Applications*, Wiley-Interscience, 1971..

[5] A. L. Oliveira and A. Sangiovanni-Vincentelli, "LSAT: An Algorithm for the Synthesis of Two-Level Threshold Gate Networks", in *Proceedings of IEEE Int. Conf. On CAD, ICCAD'*91, pages 130-133, IEEE, 1991.

[6] M. J. Avedilo and J. M. Quintana, "A Threshold Logic Synthesis Tool for RTD Circuits", in *Proceedings of the EUROMICRO Systems on*

*Digital System Design, DSD'04*, pages 624-627, IEEE, 2004.

[7] V. Beiu, "VLSI Complexity of Threshold Gate Comparison", in *Proc. IEEE Int. Symposium on Neuro-Fuzzy Systems*, pages 161-170, IEEE, 1996.

[8]R. Zhang, P. Gupta, L. Zhong and N. K. Jha, "Threshold Network Synthesis and Optimization and its Application to Nanotechnologies", in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, No. 1, pages 107-118, IEEE, January 2005.

[9]Journal of VLSI Signal Processing 38, 157–171, 2004,Operating Margin-Oriented Design Methods for Threshold Element-Based Reconfigurable Logic Circuits Realizing any Symmetric Function, KAZUO AOYAMA

[10]High-speed hybrid threshold-Boolean logic counters and compressors, Padure, M.; Cotofana, S.; Vassiliadis, S.; Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium on Volume 3, 4-7 Aug. 2002 Page(s):III-457 - III-460 vol.3

[11]An optimal dimension expansion procedure for obtaining linearly separable subsets Yuen-Hsien Tseng; Ja-Ling Wu; Neural Networks, 1991. 1991 IEEE International Joint Conference on 18-21 Nov. 1991 Page(s):2461 - 2465 vol.3

[12] IMS algorithm for learning representations in Boolean neural networks, Biswas, N.H.; Murthy, T.V.M.K.; Chandrasekhar, M.; Neural Networks, 1991. 1991 IEEE International Joint Conference on 18-21 Nov. 1991 Page(s):1123 - 1129 vol.2

[13] Perceptrons revisited: the addition of a non-monotone recursion greatly enhances their representation and classification properties Dogaru, R.; Alangiu, M.; Rychetsky, M.; Glesner, M.; Neural Networks, 1999. IJCNN '99. International Joint Conference on Volume 2, 10-16 July 1999 Page(s):862 - 867 vol.2

[14]An improved expand-and-truncate learning Yamamoto, A.; Saito, T.; Neural Networks,1997., International Conference on Volume 2, 9-12 June 1997 Page(s):1111 - 1116 vol.2

[15]Applications of binary neural networks learning to pattern classification, Chu, C.H.; Kim, J.H.; Kim, I.; Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on, Volume 2, 27 June-2 July 1994 Page(s):907 - 911 vol.2

[16].IEEE TRANSACTIONS ON NEURAL NETWORKS. VOL. **6,** NO. ***2,*** MARCH 1995, pp. 318-331,Classification of Linearly Nonseparable Patterns by Linear Threshold Elements
Vwani P. Roychowdhury, *Member, IEEE,* Kai-Yeung Siu, *Member-, IEEE,* and Thomas Kailath, *Fellow, IEEE*

[17]IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. *6*, NO. 1, JANUARY 1995, pp. *237-238* The Geometrical Learning of Binary Neural Networks, Jung H. Kim and Sung-Kwon Park

[18]Neurocomputing 57 (2004) 455 – 461
An approachfor construction of Boolean neural networks based on geometrical expansion, Di Wang. , Narendra S. Chaudhari, N.R. Pal and M. Sugeno (Eds.): AFSS 2002, LNAI 2275, pp. 236–244, 2002.

[19]Optimal Synthesis Method for Binary Neural Network Using NETLA**,** Sang-Kyu Sung1, Jong-Won Jung1, Joon-Tark Lee1, Woo-Jin Choi2, and Seok-Jun Ji3

****REFERIR NA SEC 'You may say…':**

[20]IEEE TRANSACTIONS ON NEURAL NETWORKS. VOL. *5* . **YO.** *3.* MAY 1994 *507-508*
An Iterative Method for Training Multilayer Networks with Threshold Functions, Edward M. Corwin, Antonette M. Logar. and William J. B. Oldham

[21]Recursive branching network Al-Mashouq,K.A.;Systems, Man, and Cybernetics, 1997. 'Computational Cybernetics and Simulation'., 1997 IEEE International Conference on Volume 2, 12-15 Oct. 1997 Page(s):1341 - 1344 vol.2

[22]Backpropagation algorithm for logic oriented neural networks, Kamio, T.; Tanaka, S.; Morisue, M.; Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Volume 2, 24-27 July 2000 Page(s):123 - 128 vol.2

[23]A multi-core learning algorithm for Boolean neural networks, Wang, D.; Chaudhari, N.S.; Neural Networks, 2003. Proceedings of the International Joint Conference on Volume 1, 20-24 July 2003 Page(s):450 - 455 vol.1

[24]A simple learning of binary neural networks with virtual teacher signals, Shimada, M.; Saito, T.; Neural Networks, 2001. Proceedings. IJCNN '01. International Joint Conference on Volume 3, 15-19 July 2001 Page(s):2042 - 2047 vol.3

****Referir na introdução:*****

[25] Neurocomputing 47 (2002) 161–188 New methods for testing linear separability, M. Tajinea;., D. Elizondob