

# Neural Networks – A Model of Boolean Functions

Bernd Steinbach, Roman Kohut  
Freiberg University of Mining and Technology  
Institute of Computer Science  
D-09596 Freiberg, Germany  
e-mails: steinb@informatik.tu-freiberg.de  
kohut@math.tu-freiberg.de

## Abstract

This paper deals with the representation of Boolean functions using artificial neural networks and points out three important results. First, using a polynomial as transfer function, a single neuron is able to represent a non-monotonous Boolean function. Second, the number of inputs in the neural network can be decreased if the binary values of the Boolean variables are encoded. This approach simplifies significantly the necessary number of neurons in the artificial neural network. Finally, an algorithm to compute the minimal number of neurons was developed. The lower bound, calculated by this algorithm, corresponds to a suggested structure of artificial neural networks. An example shows, how such a simple artificial neural network may represent a Boolean function.

## 1 Introduction

The rapid development of computer systems is one of the reasons why theoretical and practical researches of the principles of the nature were forced. Many new scientific and technological branches, which have connection both with biological laws of the nature and applied technical methods, appeared in the last fifty years, e.g. bioelectronics, artificial intelligence. Especially methods of the artificial intelligence are widely used in modern technologies. New waves of development such methods like data mining [8], fuzzy logic, cluster analysis, expert systems, genetic algorithms, and visual data recognition were caused by the computer science. Some of these methods base on neural networks.

The connection between neural networks and Boolean function is not new. One of the pessimistic results in the history of neural networks was the conclusion of Minsky and Papert about impossibility of representation of all functional dependence. They proved this property on the Boolean function exclusive OR (XOR) [4, 8, 10]. This statement is correct only for simplest artificial neural network, named Perceptron of Rosenblatt [6]. The result of Minsky and Papert has been later by Mkrtschjan disproved [5]. In general, all types of functional dependence including Boolean functions can be represented using neural networks. This paper gives an introduction, how artificial neural networks can model Boolean functions.

The rest of this paper is organized as follows. Section 2 introduces both Boolean functions and neurons. Section 3 shows, how non-monotonous Boolean function may represent by single neurons. Section 4 describes, how the encoding of the Boolean values can be used to minimize the size of an artificial neural network. An algorithm to calculate a strong lower bound of neurons is presented in section 5. Finally, section 6 concludes the paper.

## 2 Preliminaries

Let  $x_1, x_2, x_3, \dots, x_n$  be Boolean variables,  $x_i \in \{0, 1\} = B$ . A Boolean function  $f$  is defined by  $f: B^n \rightarrow B$ , where  $B^n = \{(0\dots 00), (0\dots 01), (0\dots 10), \dots, (1\dots 11)\}$  [1, 7]. The number of vectors in the Boolean space  $B^n$  is  $N = 2^n$ . The simplest representation of a Boolean function is a table that defines the function value for each of the  $2^n$  input vectors, see example in figure 1 a.

There are many other representations of Boolean functions. One of them is the visualization of the Boolean function as  $n$  - dimensional hypercube [1, 7]. Each input vector of the Boolean function corresponds to exactly one vertex in  $n$  - dimensional space. The function values are labeled by “●” if  $f= 1$  or “○” if  $f= 0$ . The visualization of the function  $f = x_2 \cdot (\overline{x_1} + x_3 \oplus \overline{x_2})$  shows figure 1b.

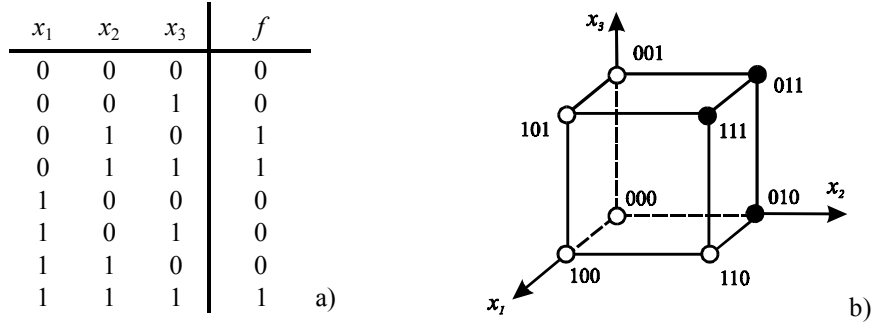


Figure 1. The Boolean function  $f = x_2 \cdot (\overline{x_1} + x_3 \oplus \overline{x_2})$ , a) Function Table, b) Visualization

Next, the artificial neural network is introduced. These networks models are simple way biological neural networks. An artificial neural network is a connection of simple processing elements, called neurons, which operates in parallel [2, 3, 4, 6, 10]. In this paper artificial neural networks are restricted to feed forward networks, which can be represented by directed acyclic graphs (DAG), see figures 4, 6. The structure of a neuron is shown in figure 2.

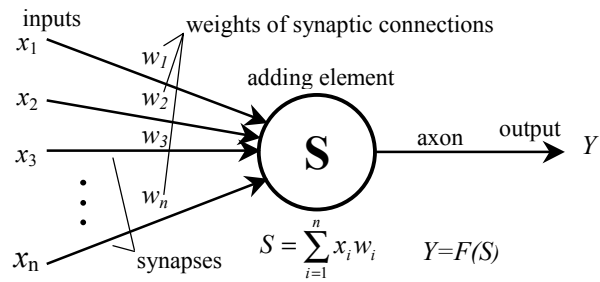


Figure 2. General structure of neuron

The mathematical description of neuron is:

$$Y = F\left(\sum_{i=1}^n x_i \cdot w_i\right) \quad (1)$$

where  $Y$  - output signal of neuron,  
 $F$  - activation function,  
 $x_i$  - signal from the input  $i$  to the synapse  $i$ ,  
 $w_i$  - weight coefficient of the input  $i$ ,  
 $n$  - number of inputs.

In general, the condition to activate a neuron is (2), where  $\theta$  is threshold of transfer function.

$$\sum_{i=1}^n w_i x_i - \theta \geq 0 \quad (2)$$

The unequation (2) includes the equation (3) of a  $n$ -dimensional hyper-plane that divides the vertices of the hypercube into two part sets.

$$\sum_{i=1}^n w_i x_i - \theta = 0 \quad (3)$$

The output of a neuron may be connected to several synapses of other neurons. The activation function is sometimes called a *transfer function*. In the case of a bounded ranges the activation functions are called *squashing functions*, such as the commonly used *tanh* (hyperbolic tangent) and the logistic function  $(1 + e^{-x})^{-1}$  [3, 10], see figure 3.

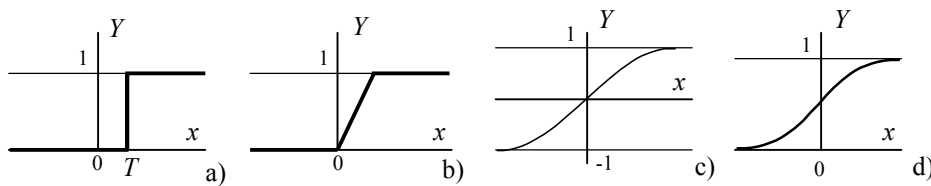


Figure 3. Activation functions: a) jump, b) threshold, c) hyperbolic tangent, d) logistic function

### 3 Neural Networks of Non-monotonous Boolean Functions

#### 3.1 The Problem

We consider elementary Boolean functions NOT, OR, AND, XOR. Table 1 shows these functions.

Table 1. Elementary Boolean functions

$x_1$	$x_2$	$\overline{x_1}$	$x_1 + x_2$	$x_1 \cdot x_2$	$x_1 \oplus x_2$
0	0	1	0	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	1	0	1	1	0

The basic question is, whether one neuron is enough to represent each of these Boolean functions. The neural networks to represent the first three functions are shown in figure 4.

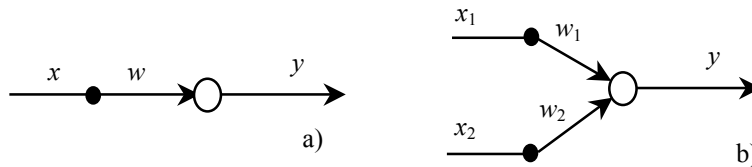


Figure 4. Neural network's structures: a) for NOT – function; b) for OR-, or AND - functions

The functions NOT, OR, or AND can be represented using one neuron.

A simple explanation for that is shown in figure 5. The activation functions from figure 3 distinguish the areas having the output zero or one. Based on the sum of the weighted inputs only monotonous Boolean functions may calculate directly by such simple activation functions. If the Boolean function is depended on one variable there exist a cut point, see figure 5 a. In case of two variables in the Boolean function, a line separates all vertexes labeled by 0 from the all other vertexes labeled by 1, see figures 5 b and c. Generally, if there are more then two Boolean variables, a plane or hyper-plane separates both areas.

For the XOR-function such a line or hyper-plane does not exist, see figure 5 d. Exactly on this example Minsky and Papert proved impossibility of representation some functional dependences using one neuron [4].

In the case of neural element with two inputs and threshold  $\theta$ , we have from (3) the equation of the line in the plane  $x_1 - x_2$ :

$$x_1 w_1 + x_2 w_2 = \theta \tag{4}$$

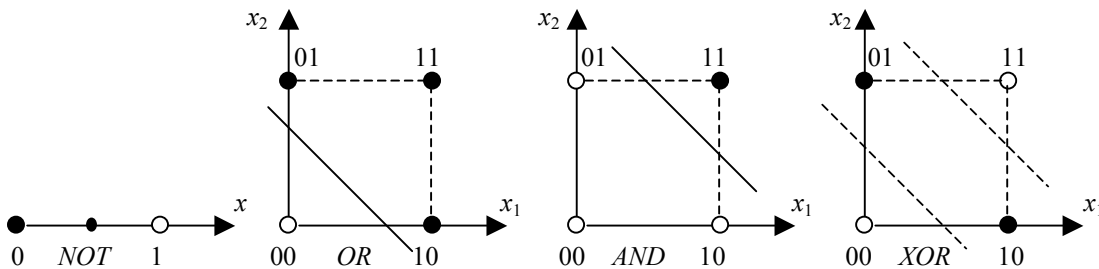


Figure 5. Visualization the separator of neural networks in the Boolean space  
a) NOT-function; b) OR-function; c) AND-function; d) XOR-function

#### 3.2 Known Approaches for Non-monotonous Boolean Functions

The simplest approach is to express the non-monotonous Boolean function as disjunctive normal form, which includes only monotonous operations. Each of these monotonous operations is mapped to

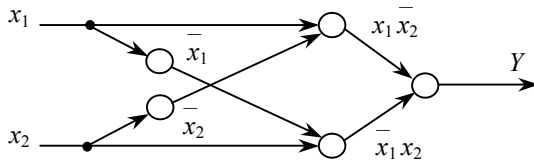


Figure 6. Neural network of the XOR-function using monotonic neurons

one neuron. This simple method is useable for each Boolean function, but already the simplest non-monotonous Boolean function (5) needs more then one neuron. The figure 6 shows the neural network, associated to the XOR-function (5).

$$Y = XOR(x_1, x_2) = x_1 \oplus x_2 = x_1 \bar{x}_2 + \bar{x}_1 x_2 \quad (5)$$

The second known method uses one additional input into one single neuron in order to realize the XOR-function. This method needs an additional OR-gate and was proposed by Mkrtschjan [5]. The main idea is the transformation of the two valued Boolean function into the three dimensional Boolean space, which changes (4) into the equation (6), where  $w_1 = -1, w_2 = -1, w_3 = 2, \theta = 0,5$  and  $a_1 = x_1, a_2 = x_2, a_3 = x_1 \text{ OR } x_2$  in order to model the XOR-function (5).

$$w_1 a_1 + w_2 a_2 + w_3 a_3 = \theta \quad (6)$$

The equation (6) substitutes the line (4) into a plane; see figure 7 a. Correspondingly, The Boolean function  $f(x) = x_1 \bar{x}_2 + \bar{x}_1 x_2$  is transformed into  $f(\alpha) = \alpha_1 \bar{\alpha}_2 \alpha_3 + \bar{\alpha}_1 \alpha_2 \alpha_3 + \alpha_1 \alpha_2 \alpha_3$ .

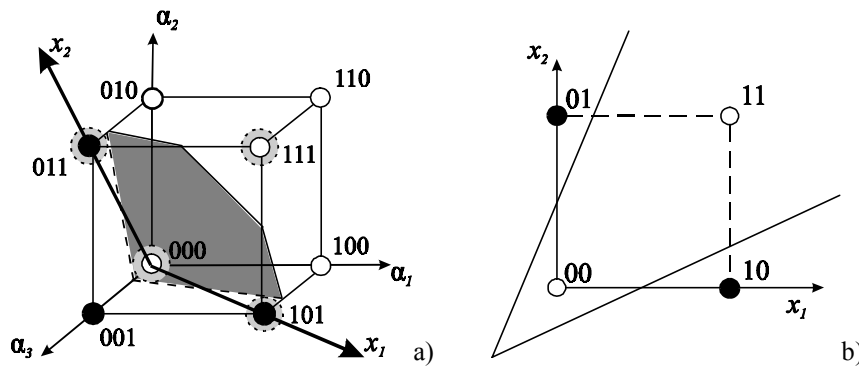


Figure 7. The representation of the XOR-function, a) plane defined by (6), b) projection of the plane in the two-dimensional space

The chosen parameter of (6) map the four vertices of  $B^2$  into the vertices labeled by a dotted circle in figure 7 a. Thus, the axis  $x_1$  can be modeled in the plane  $a_1 a_3$  and the axis  $x_2$  in the plane  $a_2 a_3$ . The return transformation of the separation plan, defined by (6) and visualized in figure 7 a, leads to the broken line, visualized in figure 7 b. This broken line separates the vertices labeled by 0 form that vertices labeled 1. Note, using this approach, the XOR-function needs two neurons in the network, the first two synapses neuron calculates the OR-function and the second three synapses neuron computes the final XOR-function.

### 3.3 Polynomial Transfer Functions

The two methods, discussed in section 3.2, need more than one neuron to realize a non-monotonous Boolean function. These methods increase number of signals, modeled in a lager dimensional space.

Alternatively, a single neuron can realize the non-monotonous Boolean function, if this neuron takes advantages of a more complex activation function, like a polynomial (7) or trigonometrical (8) function.

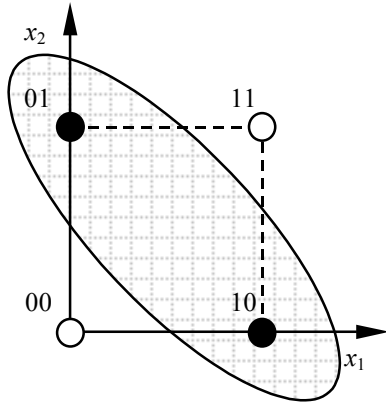


Figure 8. Representation of the XOR-function using single neurons that evaluates a polynomial transfer function

$$d * \left( w_0 + \sum_{i=1}^n w_i x_i \right)^k - \theta \geq 0 \quad (7)$$

$$f \left( w_0 + \sum_{i=1}^n w_i x_i \right) - \theta \geq 0 \quad (8)$$

The advantage of this method is a one to one mapping of the Boolean function to the neural network. In case of the XOR-function we chose  $d = w_0 = -1$ ,  $w_1 = w_2 = 1$ ,  $k = 2$ , and  $\theta = -0.5$ , and get the formula (9), which is true only if, the XOR-function is equal to one.

$$-1 * \left( -1 + \sum_{i=1}^n x_i \right)^2 + 0.5 \geq 0 \quad (9)$$

Figure 8 illustrates the valid area of the equation (9).

## 4 Encoding the Inputs of Boolean Functions

Now the case of more complex Boolean functions depending on a large number of variables is considered. In general there are three methods to create the neuronal network that describes a complex Boolean function.

First, the Boolean function is mapped to a special neuronal network using such neurons that realizes the NOT-, AND-, and OR-function. In a second step, merging of selected neurons can minimize the number of neurons of this neuronal network. This is a time consuming process.

Second, a general artificial neuronal network is used. This neuronal network has one input for each Boolean variable and a sufficient number of layers. The advantage of such simple structure of the neuronal network is connected with the disadvantage of an extremely time consuming training process. For example, a Boolean function with 30 arguments has  $2^{30} = 1\,073\,741\,824$  different binary vectors. Each of them must apply to train the artificial neuronal network. Using the training method Back Propagation this task may need approximately one year. A quick algorithm, like "Functional on the sets of tabled functions" [8, 9] to train the neural network, speed up this procedure 50 times, but this algorithms required 32 212 254 720 byte = 32 Gigabyte random access memory (RAM).

The both methods, described above, are restricted in the time, needed to specify all details of the artificial neuronal network. Alternatively, we suppose the encoding of a certain number of Boolean variables into real number. A neuron is able to process real numbers up to a given precision. A possible encoding for two or  $n$  Boolean variable into the interval  $[0, 1]$  is shown in table 3. The number of inputs and the number of neurons as well is decreased, using this encoding.

Table 3. Encoding input binary vectors for neural network

$x^*$	$x_1$	$x_2$				
$x_1^* = 0$	0	0				
$x_2^* = 1/3$	0	1				
$x_3^* = 2/3$	1	0				
$x_4^* = 1$	1	1				

For general case ➔

$x^*$	$x_1$	$x_2$	...	$x_n$
$x_1^*$	0	0	...	0
$x_2^*$	0	0	...	1
...	...	...	...	...
$x_{2^n}^*$	1	1	1	1

For instance, a 8-bits number representation of a signal allow to distinguish between  $2^8$  values. Under this assumption, a Boolean function of 16 arguments can be realized by a neural network with two inputs ( $2^8 * 2^8 = 2^{16}$ ). This decreases the number of inputs by one magnitude (from 16 to 2), and consequently the number of layers and the number of necessary neurons is decreases as well. The smaller number of neurons speeds-up of training process significantly. This approach allows to express very complex Boolean functions in much simpler structures of neural networks. The restriction of this method is sensitivity of neurons.

## 5 Lower Bound of the Number Neurons in the Neural Network

In this section we propose an algorithm, which calculates the lower bound of neurons on each layer of neural network. Thus, this algorithm finds the structure of the smallest possible neural networks of Boolean functions in terms of the number of layers and the number of neurons in each layer as well.

The algorithm uses the following assumption. The interval of the signals level is  $[0, 1]$ . The sensitivity of neurons is labeled by  $\Delta$ , which means that the neuron can distinguish between  $N_s = \Delta^{-1}$  different values of a signal. A Boolean function of  $m$  arguments needs a sensitivity  $\Delta^*$ , defined by (10).

$$\Delta^* \leq \frac{1}{2^m} \quad (10)$$

### Algorithm (Number of Neurons and Layers)

1. Assume, the Boolean function may be represented by one hidden layer,  $r = 1$ .
2. Compute the minimal number of neurons  $K_r$  on the layer  $r$  using the formula (11). The special brackets  $\left[ \right]$  denote the smallest integer value that is larger than the enclosed real value.

$$\text{if } \Delta > \Delta^*, \text{ then } K_r = \left\lceil \frac{\log_2 \left( \frac{1}{\Delta^*} \right)}{\log_2 \left( \frac{1}{\Delta} \right)} \right\rceil \quad \text{else } K_r = 0 \quad (11)$$

3. If  $2^{K_r} \leq N_s$ , then the assumption of step 1 is true and minimal number of neurons to representation Boolean function can be calculated by (12).

$$M_z = 1 + \sum_{r=1}^{r_{\max}} K_r \quad (12)$$

Otherwise, the assumption of step 1 is false, the number of hidden layers  $r$  is incremented by one, the necessary sensitivity for next layer  $\Delta^*$  is enlarged by (13).

$$\Delta^* := 2^{-K_r} \quad (13)$$

and the steps 2 and 3 must be repeated for the next hidden layer.

### Example

What is the minimal neuronal network of the non-monotonous Boolean function (14)?

$$f = \bigoplus_{i=1}^8 x_i \quad (14)$$

We assume, that the sensitivity of the available neurons is  $\Delta^* = 2^{-4}$ , so that the neuron can distinguish between  $N_s = \Delta^{-1} = 2^4$  values of a signal. Because the Boolean function has  $n = 8$  arguments, the necessary sensitivity is  $\Delta^* = 2^{-8}$ . A single hidden layer is assumed and the number of neurons in this hidden layer  $K_1$  is calculated (15).

$$K_1 = \left\lceil \frac{\log_2 \left( \frac{1}{\Delta^*} \right)}{\log_2 \left( \frac{1}{\Delta} \right)} \right\rceil = \left\lceil \frac{\log_2 2^8}{\log_2 2^4} \right\rceil = \left\lceil \frac{8}{4} \right\rceil = 2. \quad (15)$$

Since  $2^{K_1} = 4 \leq N_s = 16$ , the algorithm finds in step 3 the minimal number of neurons (16).

$$M_z = 1 + K_1 = 1 + 2 = 3. \quad (16)$$

The simplest feed forward neural network to represent the Boolean function (14) needs one input layer, only one hiding layer, and a one-neuron output layer, see figure 9. Three neurons having a sensitivity  $\Delta=2^{-4}$  determine the Boolean function which depends on 8 variables.

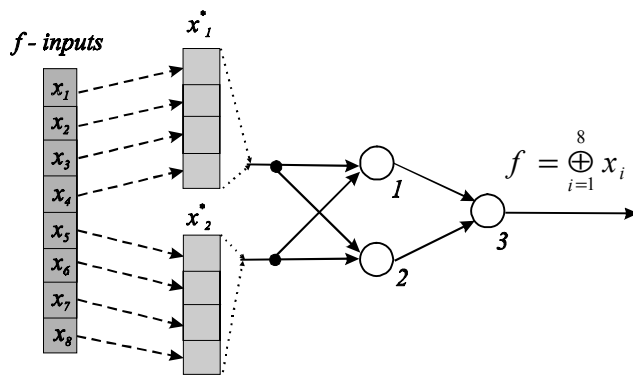


Figure 9.  
Simple architecture of feed forward neural network to represent the Boolean function (14)

## 6 Conclusion and Future Work

Each Boolean function can represent by several artificial neural networks. Using Boolean signals on the synapses and one of the traditional activation functions, a single neuron is only able to realize a monotonous Boolean function. There exist a one-to-one mapping of NOT-, AND- or OR-gates to such simple neurons. Artificial neural networks created by this simple one-to-one mapping describe an upper bound of neuronal networks and have no benefit compared to corresponding gate circuit. The reason for that is the more complex structure of the neurons in comparison to the logic gates.

Two approaches to simplify such neuronal networks were suggested. First, using polynomial or trigonometrical activation functions, a single neuron can represent non-monotonous Boolean functions, which covers a certain set of logic gates. This approach reduces the number of neurons significantly and a real benefit of such artificial neural networks in terms of space and time is possible.

The second way to take advantages of the capability of neurons consists in the encoding of a larger number of Boolean variables into each input signal of the neuron. This approach is only limited by the sensitivity of the used neurons and reduces the number of neurons and layers of the artificial neural network drastically. Thus, both the memory space to store neural network and the time of the learning and using phase are decreased.

In the future we will extend the theory artificial neural networks in order find an optimal representation of Boolean functions using such networks. Based on these results we will design and implement a package, that uses artificial neural networks for compact representation and fast computations and evaluations of Boolean functions.

## 7 References

- [1] Bochman, D.: Steinbach, B.: Logikentwurf mit XBOOLE. Verlag Technik, Berlin, 1991.
- [2] Gschwendtner A. B. DARPA Neural Network Study. AFCEA International Press, p. 60, 1988.
- [3] Kröse, B.; v. d. Smagt P.: An introduction to Neural Networks. University of Amsterdam, 1996.
- [4] Minsky M. and Papert S. Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, MA, 1969.
- [5] Mkrtschjan S.O. Neurons and neural networks - Introduction in the theory of formal neurons. (In Russian), Energy, Moscow, 1971.
- [6] Rosenblatt F. Principles of Neurodynamics. Spartan, New York, 1962.
- [7] Steinbach, B.: XBOOLE - A Toolbox for Modeling, Simulation, and Analysis of Large Digital Systems. System Analysis and Modeling Simulation, Gordon & Breach Science Publishers, 9(1992), Number 4, pp. 297-312, 1992.
- [8] Tkachenko, R.: Kohut, R.: Feed forward neural networks: the problems of synthesis and using. (In Ukrainian), Bulletin of Lviv Polytechnic National University: Computer Engineering and Information Technologies, № 433, Lviv, pp. 166-171, 2001.
- [9] Tkachenko, R.: Kohut, R.: Functional extension of inputs in feed forward neural network with non-iteration learning. (In Ukrainian), Technical news 1(12), 2(13), Lviv, pp. 91-94, 2001.
- [10] Wasserman P.D.: Neural Computing Theory and Practice. Van Nostrand Reinhold, New York, 1989.