46/2001 A05/1

Raport Badawczy

Research Report

RB/55/2001

fXOR fuzzy logic networks

Witold Pedrycz, Giancarlo Succi

Instytut Badań Systemowych
Polska Akademia Nauk

Systems Research Institute
Polish Academy of Sciences

POLSKA AKADEMIA NAUK

Instytut Badań Systemowych

ul. Newelska 6

01-447 Warszawa

tel.: (+48) (22) 8373578

fax: (+48) (22) 8372772

Pracę zgłosił: prof. dr hab.J.Kacprzyk

Warszawa 2001

fXOR fuzzy logic networks

Witold Pedrycz[1,2] and Giancarlo Succi[1]

[1]Department of Electrical and Computer Engineering
University of Alberta
Edmonton, Canada T6G 2G7
&
[2]Systems Research Institute, Polish Academy of Sciences
01-447 Warsaw, Poland

Abstract The study introduces a new class of fuzzy neurons and fuzzy neural networks exploiting a model of a generalized multivalued exclusive-OR (XOR) operation. The proposed neural architecture is useful in an algebraic representation (description) of fuzzy functions regarded as mappings between unit hypercubes, say $[0,1]^n \rightarrow [0,1]^m$. Some underlying properties of the fXOR neurons are discussed and a detailed learning algorithm is given along with a number of illustrative numeric examples.

Keywords fuzzy functions, optimization, logic networks, learning, fuzzy hardware, Reed-Muller expansion, algebraic form of fuzzy functions

1. Introduction

The algebraic Muller-Reed expansion of Boolean functions, cf. [9] is regarded as one of the fundamental approaches in the design of digital systems, especially when dealing with the VLSI technology. This algebraic representation is also useful in error detection and error correction models [9][10]. The design of multivalued logic circuits has been an interesting endeavor pursued vigorously in the realm of multivalued logic and fuzzy sets. Various optimization techniques have been developed both for combinational (static) as well as dynamic (sequential) systems.

In this study, we are concerned with the generalization of the Reed-Muller algebraic representation applied to multivalued (fuzzy) functions. To accomplish that, one has to define fuzzy exclusive-OR functions that are a cornerstone of such representation. The objective of this study is to develop a logic-based architecture of fuzzy neural networks, called here fXOR networks, that are capable of realizing this type of the mapping. In contrast to the "standard" way of approximation of fuzzy functions (that is realized via a generalized sum of minterms and becomes a generalized Shannon representation model), this approach leads to a compact representation and features useful learning properties.
In comparison to neural networks [2][3], the class of fuzzy neural networks comes with interpretation capabilities as its structure could be easily interpreted in the language of multivalued logic. At the same time, the network can be easily trained as we have a vast array of plastic connections of the logic neurons.

The material of this study is arranged into six sections. We introduce a concept of a fuzzy XOR neuron (fXOR), study its properties (Section 2), then move to the overall architecture of the network and discuss a detailed learning algorithm (Section xx4 Numeric experiments are covered in Section 5.

In what follows, we adhere to the standard notation being used in fuzzy sets. In particular, we use t- and s-norms as treated as two general classes of logic connectives (logic operators). The complement of the variable is denoted by an overbar, that is $\overline{x} = 1 - x$. All variables assume the values in the unit interval.

2. The fXOR fuzzy neuron

The fuzzy XOR neuron (fXOR, for short) is a generalization of the standard exclusive – OR operation (gate) used in digital (Boolean) systems. More formally, an n-input single output fXOR is governed by the expression

$$y = \mathrm{fXOR}(\mathbf{x},\mathbf{w}) \tag{1}$$

where $\mathbf{x}$ and $\mathbf{w}$ are elements in the n-dimensional unit hypercube, that is $\mathbf{x}, \mathbf{w} \in [0,1]^n$. The underlying logic static transformation is realized through the use of s- and t-norms. More specifically, we have

$$y = \mathrm{fXOR}(\mathbf{x},\mathbf{w}) = \bigoplus_{i=1}^{n} (x_i s w_i) = (x_1 s w_1) \oplus (x_2 s w_2) \oplus \ldots \oplus (x_n s w_n) \tag{2}$$

where the generalized exclusive-OR operation given above ($\oplus$)is defined by the following s-t composition

$$a \oplus b = (\bar{a} t b) s (a t \bar{b}), \quad a, b \in [0,1]$$

The expression of the fXOR neuron in (2) comes with x_i and w_i being the coordinates of the fuzzy sets, i=1, 2,...,n and t- and s- describing triangular norms and conorms. The above convolution of $\mathbf{x}$ and the weight vector (vector of the connections) $\mathbf{w}$ is just a t-s composition of two fuzzy sets and as such follows the fundamentals of the calculus of fuzzy relational equations, cf [1].

By studying the characteristics of the neuron (for n=2), Figure 1, we clearly realize that for different inputs we achieve higher values of the output. The more similar the inputs, the lower the output of the fXOR neuron. In the binary (two-valued) case we end up having the standard characteristics of the XOR function.

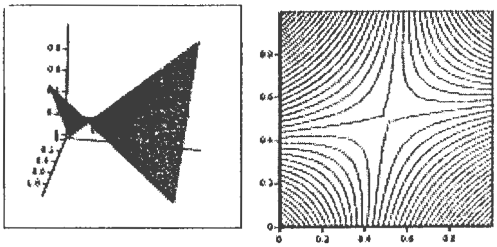
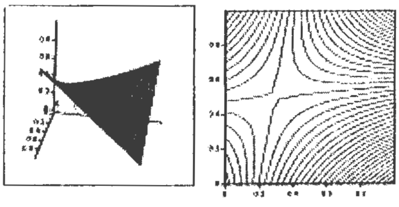
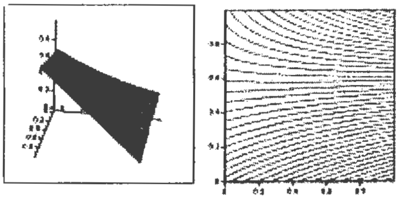


Figure 1. Characteristics of the fXOR for $w_1 = w_2 = 0$.

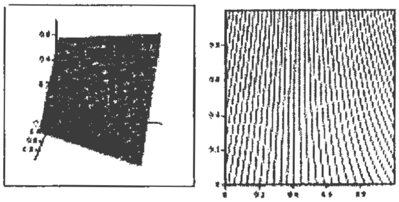
The meaning of the connections becomes obvious by studying the properties of the t-s composition. Visually, by comparing the outputs of the neuron for selected values of the connections, Figure 2, we learn that the connections help quantify the relationships between the input variables and the corresponding output. The higher the value of the connection, the less intensive (visible) the impact of the corresponding input variable on the output of the neuron. For $w_i = 1$ this impact is totally eliminated. On the other hand, for $w_i = 0$, the impact is the most evident.



(a)



(b)



(c)

Figure 2. Characteristics of the fXOR neuron for selected values of the connections; note the changes in the input-output relationships depending on the values of the connections: (a) $w_1=0.3$, $w_2=0.0$ (b) $w_1=0.7$ $w_2=0.0$ (c) $w_1=0.3$ $w_2=0.8$

3. The architecture of the network

The proposed fuzzy neural network generalizes the algebraic representation of fuzzy functions and is in analogy to what occurs in the Reed-Muller expansion of Boolean functions. Two types of fuzzy neurons are being used: the fXOR neurons already discussed and the AND neurons. As shown in Figure 3, the network exhibits a single hidden layer consisting of AND neurons that is followed by the output layer of the fXOR neurons. The role of the AND neurons is to build a logical AND aggregation of the input variables (appearing here in a direct as well as complemented format), that is x_1, x_2, ...,x_n. As discussed in [4][5][6][7][8], the expression governing this processing reads as

$$z = \mathrm{AND}(\mathbf{x};\mathbf{v})$$

(3)

where **v** is a vector of the connections (weights) of this neuron. Considering individual variables, we rewrite (3) as follows

$$z = \mathop{\mathrm{T}}_{i=1}^{n} (x_i s v_i) t \mathop{\mathrm{T}}_{i=1}^{n} (\overline{x}_i s v_{n+i})$$

(4)

or using the uniform notation in which we combine all inputs and their complements

(in the sequel we will not be using this detailed notation by simply alluding to the vector inputs x assuming that it includes also the complemented variables).

Again, in view of the t-s composition, the role of the connections become evident: the higher the value of the connection, the less impact becomes reported for the corresponding variable. If $v_i=1$ then the associated variable (x_i) does not impact the output (z) as it has been totally eliminated. Interestingly, when we confine to the {0,1} world, the AND neuron generalizes a well-known AND gate.

The result of the AND aggregation are then combined through the fXOR operation and this is realized at the output layer, see again Figure 3, y_j = fXOR(z, w_i), i=1,2,..,m.

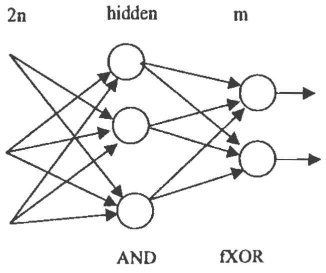


Figure 3. A general topology of the fXOR network

Altogether, the topology of the network can be concisely described as follows (we assume that the number of the nodes in the hidden layer is equal to hidden),

$$z_i = \mathrm{AND}(\mathbf{x}, v_i)\ i=1,2,\ldots,\text{hidden}$$
$$y_j = \mathrm{fXOR}(\mathbf{z}, w_j),\ j=1,2,\ldots,m$$

(5)

The connections of the neurons exhibit a clearly defined semantics so that the resulting network (once the learning is done) can be easily transformed into a certain quantified logic expression.

Interestingly, the size of the network could be made quite compact when comparing it to the standard logic processing, cf. [6][8] that is a logic network with a single hidden layer. More specifically, the hidden layer consists of AND neurons while the output layer consists of OR neurons, cf' [7]. An example of a simple XOR function being implemented by the fXOR network and the logic processor is of interest here. For "n" variables, the LP architecture requires $2^n/2= 2^{n-1}$ AND nodes in the hidden layer (that becomes obvious as we have to implement all required miniterms). This number grows up very quickly; e.g., for n=10 input variables, we require 2^9 =512 AND neurons in the hidden layer. In contrast, when dealing with the fXOR architecture, this relationship is linear (so the number of the neurons is equal to "n").

4. A detailed learning scheme

In this section, we elaborate on the details of the learning scheme. The optimization (learning) is carried out using a standard gradient-based method. The parameters to be optimized are the connections of the AND and fXOR neurons. The fundamental expression governing the learning process is

$$\mathbf{connections}(\text{iter}+1) = \mathbf{connections}(\text{iter}) - \alpha \nabla_{\mathbf{connections}} Q \tag{6}$$

where the learning rate (α >0) is used to control the speed of learning. For notational reasons, the connections are combined together and denoted as **connections**. One should view them as a vector of the connections of the AND and fXOR neurons. Alluding to the connections of the neurons, we write down

$$w_{st}(\text{iter}+1) = w_{st}(\text{iter}) - \alpha \frac{\partial Q}{\partial w_{st}(\text{iter})}, \text{ s=1, 2, ..., n, t =1, 2, ..., hidden} \tag{7}$$

$$v_{st}(\text{iter}+1) = v_{st}(\text{iter}) - \alpha \frac{\partial Q}{\partial v_{st}(\text{iter})}, \text{ s=1, 2,..., hidden, t=1, 2, ..., 2n} \tag{8}$$

(we consider that the input layer consists of the original inputs as well as their complements)

The performance index Q assumes the form (note that we are concerned with the on-line type of learning)

$$Q = (\mathbf{target\text{-}y})^T(\mathbf{target\text{-}y}) = \sum_{k=1}^{m} (\text{target}_k - y_k)^2 \tag{9}$$

In the following detailed derivations, we adhere to the detailed notation in the form shown in Figure 4.

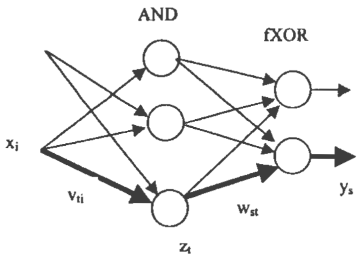


Figure 4. The development of the learning scheme in the fXOR network – notation and detailed layout of the network

Subsequently, we consider an on-line mode of learning (meaning that the updates of the connections occur after each pair of input-output data, say (**x, target**)). The detailed scheme can be derived once we decide upon the form of the triangular norms. In the following investigations, we treat a t-norm as a product operation while the s-norm is a probabilistic sum. This allows us to rewrite (7) – (8) in the following manner,

- for the connections of the fXOR neurons forming the output layer
-

$$w_{st}(\text{iter}+1) = w_{st}(\text{iter}) - \alpha\frac{\partial Q}{\partial w_{st}(\text{iter})} = w_{st}(\text{iter}) + 2\alpha(\text{target}_s - y_s)\frac{\partial y_s}{\partial w_{st}(\text{iter})} \tag{10}$$

where

$$y_s = (z_t s w_{st}) \oplus G$$

and

$$G = \bigoplus_{\substack{j=1\\ j\neq t}}^{\text{hidden}} (z_j s w_{sj})$$

Introducing the notation $a=z_t s w_{st}$ and proceeding with the calculations of the derivative of y_s, we obtain

$$\frac{\partial y_s}{\partial w_{st}} = \frac{\partial}{\partial w_{st}}(a \oplus G) = \frac{\partial}{\partial w_{st}}(a\overline{G} \text{ s } \overline{a}G) =$$

$$= \frac{\partial}{\partial w_{st}}(a\overline{G} + \overline{a}G - \overline{a}a\overline{G}G) = (1-z_t)(\overline{G}-G) - \overline{G}G(1-z_t)(1-2a)$$

- for the connections of the AND neurons located in the hidden layer the calculations of the gradient follow the general scheme outlined in (8) with the detailed expression for the connection v_{st} assuming the form

$$\frac{\partial y_k}{\partial v_{st}} = \frac{\partial y_k}{\partial z_s}\frac{\partial z_s}{\partial v_{st}}$$

(11)

and

$$\frac{\partial y_k}{\partial z_s} = (1 - w_{ks})(\overline{G} - G) - \overline{G}G(1 - w_{ks})(1 - 2a)$$

(12)

where $a = z_s \text{ s } w_{ks}$

Additionally

$$\frac{\partial z_s}{\partial v_{st}} = A(1 - x_t)$$

with

$$A = \underset{\substack{l=1 \\ l \neq t}}{\overset{n}{\mathrm{T}}} (x_l s v_{sl})$$

Thus the learning of the network is straightforward: we proceed with an initial configuration of the connections (that are usually set up to some small random numbers) and then cycle through the data set (**x**(k), **target**(k)),k=1,2, …,N using (7) – (8) until a certain performance criterion is met and here it is a sum of squared errors between the target vectors and the corresponding outputs of the fXOR network

$$Q = \sum_{k=1}^{N} (\mathbf{target}(k)\text{-fXOR\_network}(\mathbf{x}(k))^T(\mathbf{target}(k)\text{-fXOR\_network}(\mathbf{x}(k))$$

(13)

or we have exceeded a predefined number of learning cycles (where by the cycle we mean the number of processing phases of the entire training dataset. One should mention the role of the learning rate as it affects the efficiency of the learning process; it is prudent to start with low values of the learning rate thus preferring a stable learning to its speed.

5. Numerical experiments

We start with a number of two-valued logic problems whose intent is to visualize the performance of the architecture and the associated learning algorithm. Next we discuss continuous (multivalued) data.

The XOR data set. The learning rate (α) is set to 01. We consider the problem of n = 2,3, and 4 input variables. Starting from n=2, the performance index Q is shown in Figure 5. Apparently, the learning is fast and stable resulting in a rapid reduction of the values of the performance index Q

The resulting connections assume the 0-1 values (because we are dealing with the Boolean data), this finding becomes intuitively appealing.

$$\mathbf{W} = [\,0\ \ 0\,] \qquad \mathbf{V} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$$

In light of the notation being used, this immediately translates into the Boolean expression

$$z_1 = x_1\bar{x}_2 \qquad z_2 = x_2\bar{x}_1$$

After some simplification we arrive at the expression

$$y = z_1 \oplus z_2 = x_1 \oplus x_2$$

which is nothing but a two-input XOR function.

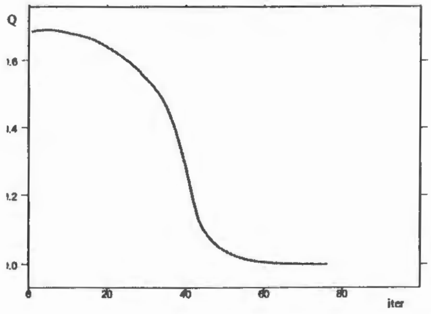


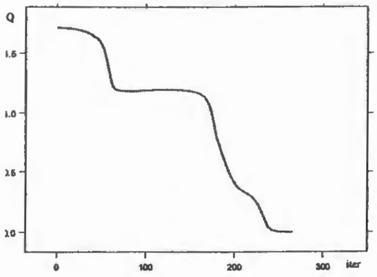
Figure 5. Performance index Q in successive learning epochs for n=2

The process of learning for n=3 and 4 input variables is shown in Figure 6. Interestingly, a certain pattern of learning builds up: we note that some jumps (decrease) in the values of Q occur. This effect gets more profound once the dimensionality of the problem increases. The connections are still Boolean and they lead to the explicit formula for the Boolean function. For instance for n=4, we derive

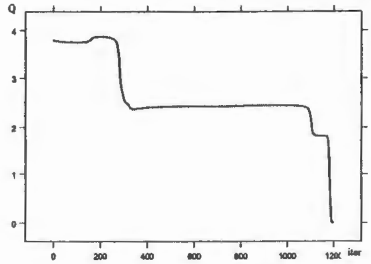
$$\mathbf{W}=[0\ 0\ 0\ 0]$$

and

$$\mathbf{V}=\begin{bmatrix} 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 0.000000 & 1.000000 \\ 0.000000 & 1.000000 & 1.000000 & 0.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 \\ 1.000000 & 1.000000 & 1.000000 & 1.000000 & 0.000000 & 1.000000 & 1.000000 & 0.000000 \\ 1.000000 & 0.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 \end{bmatrix}$$



(a)



(b)

Figure 6. Values of the performance index in successive learning epochs for n=3 (a) and n=4 (b)

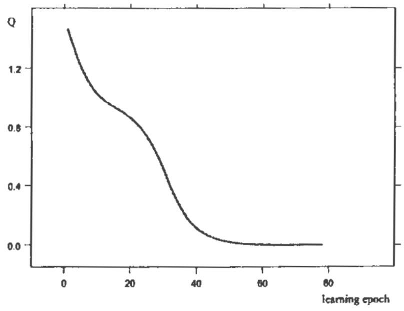
It is interesting to discuss what happens if we get too many nodes in the hidden layer. For instance, still considering the same number of inputs, we go for a higher number of the nodes in the hidden layer. When considering 7 nodes in the hidden layer and completing the learning (Q=0.0), the resulting connections are as follows,

$$\mathbf{W} = [1\ 0\ 0\ 0\ 1\ 0\ 1]$$

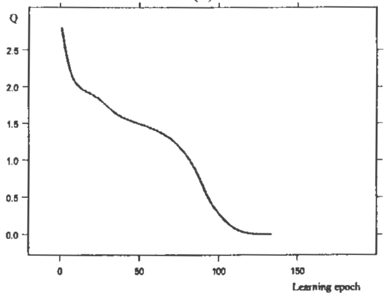
$$\mathbf{V} = \begin{bmatrix} 0.236758 & 0.521474 & 0.141856 & 0.095374 & 0.420670 & 0.282198 & 0.401083 & 0.997504 \\ 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 0.000000 & 0.000000 & 1.000000 \\ 1.000000 & 0.000000 & 0.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 \\ 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 1.000000 & 0.000000 \\ 0.904109 & 0.549009 & 0.504582 & 1.000000 & 0.176317 & 0.256148 & 1.000000 & 0.765040 \\ 1.000000 & 1.000000 & 1.000000 & 1.000000 & 0.000000 & 1.000000 & 1.000000 & 1.000000 \\ 0.646682 & 0.115568 & 0.971349 & 0.347514 & 0.097477 & 0.510434 & 0.940370 & 0.620606 \end{bmatrix}$$

Now, it becomes obvious that some of the AND neurons are not used whatsoever as the connections of the fXOR neuron (indicated in boldface) make the corresponding inputs completely inactive. Finally, we end up with the same inputs as the ones encountered before for the smaller network.

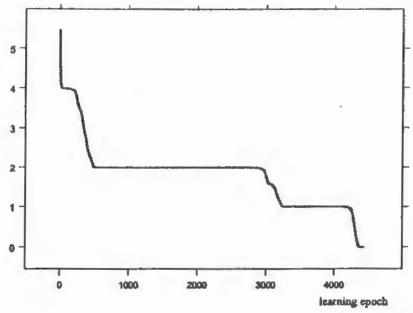
It is interesting to see how this network compares to the standard logic processor (LP) introduced and studied in [4]. We keep the same value of the learning rate as before to make the comparative analysis coherent. The differences are more profound, Figure 7, when the number of the input variables grows: the learning of the LP becomes less efficient as opposed to the optimization procedures realized in the fXOR network. For example, for n = 4 it took LP over 4,000 learning epochs to converge as opposed to 30% of the number of iterations used for the training of the fXOR network.



(a)



(b)



(c)

Figure 7. The values of the performance index Q for the logic processor (LP) in successive learning epochs for n=2 (a) , 3 (b) , and 4 (c) input variables

Next, we consider a small single-input single-output synthetic data set, see Table 1.

Inputs	0.7 0.1	0.2 1.0	0.5 0.8	0.3 0.7	0.1 0.8	0.6 1.0	0.3 0.2	0.9 0.4	0.1 0.7	0.8 0.7
output	0.66	0.3	0.7	0.35	0.45	0.5	0.4	0.62	0.34	0.7

The optimal size of the hidden layer is determined experimentally, Figure 8, and is equal to 4 with the connections assuming the following values

$$\mathbf{W} = [0.995430\ \ \mathbf{0.021711}\ \ \mathbf{0.053604}\ \ 0.881476]$$

$$\mathbf{V} = \begin{bmatrix} 1.000000 & 0.476227 & 1.000000 & 0.842960 \\ 0.860312 & 0.003482 & 0.768395 & 0.194043 \\ 0.006257 & 0.993714 & 0.871346 & 0.947147 \\ 0.694389 & 0.999144 & 0.998759 & 0.998386 \end{bmatrix}$$

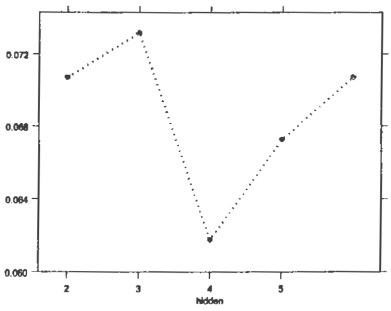
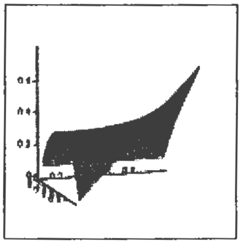
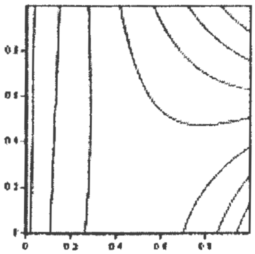


Figure 8. Performance index Q versus the size of the hidden layer (α=0.2, 8,000 learning epochs)

Considering two of the most dominant AND neurons in the architecture, the characteristics of the network computed in this way are shown in Figure 9.



(a)



(b)

Figure 9. Plots of the fuzzy neural network with the two most essential AND nodes in the hidden layer (a) 3D plot and (b) contour plot

6. Concluding comments

We have introduced and discussed a novel architecture of the logic–based model of neurocomputing generalizing an algebraic mode of computing with fuzzy functions. We showed the architecture of the network (consisting of AND and fXOR neurons) and developed a detailed learning algorithm showing how the connections of the neurons are optimized. The topology of the network is helpful in building compact representations of the logic expressions for experimental data.

Acknowledgments
The support from the Natural Sciences and Engineering Research Council of Canada (NSERC) and ASERC (Alberta Software Engineering Research Consortium) is gratefully acknowledged.

7. References

1. A. Di Nola, S. Sessa, W. Pedrycz, E. Sanchez, *Fuzzy Relational Equations and Their Applications in Knowledge Engineering*, Kluwer Academic Press, Dordrecht, 1989.
2. M.H. Hassoun, *Fundamentals of Artificial Neural Networks*, MIT Press, Cambridge, MA, 1995.
3. S.Haykin, *Neural Networks: a Comprehensive Foundation*, Macmillian College Publ, 1994.
4. W. Pedrycz, Neurocomputations in relational systems, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13, 1991, 289-296.
5. W. Pedrycz, A. Rocha, Knowledge-based neural networks, *IEEE Trans. on Fuzzy Systems*,1, 1993, 254-266.
6. W. Pedrycz, P. Lam, A.F. Rocha, Distributed fuzzy modelling, *IEEE Trans. on Systems, Man and Cybernetics*, 5, 1995, 769 - 780.
7. W. Pedrycz, F. Gomide, An Introduction to Fuzzy Sets; Analysis and Design. MIT Press, 1998.
8. W.Pedrycz, A.V. Vasilakos, Linguistic models and linguistic modeling, *IEEE Trans. on Systems Man and Cybernetics*, 1999, vol. 29, no. 6, 745-757.
9. I.S. Reed, A class of multiple-error-correcting codes and their decoding scheme, *IRE Trans. Inf. Theory,* PGIT-4, 1954, 38-49.
10. W. G. Schneeweiss, *Boolean Functions with Engineering Applications and Computer Programs*, Springer-Verlag, Berlin, 1989.