IMPLEMENTATION OF METHODS OF COMPUTATIONAL INTELLIGENCE

Bodgan WILAMOWSKI

Auburn University, Alabama Microelectronic, Science and Technology Center, 200 Broun Hall,
Department of Electrical and Computer Engineering, AL 39849, USA e-mail wilam@ieee.org

Abstract: Nonlinear processes are difficult to control because there can be so many variations of the nonlinear behavior. Traditionally, a nonlinear process has to be linearized first before an automatic controller can be effectively applied. This is typically achieved by adding a reverse nonlinear function to compensate for the nonlinear behavior so the overall process input-output relationship becomes somewhat linear. The adaptive systems are best handled with methods of computational intelligence such as neural networks and fuzzy systems. This presentation will focus on several methods of developing close to optimal architectures and on finding efficient learning algorithms. The problem becomes even more complex if the methods of computational intelligence have to be implemented in hardware. Various practical solutions will be presented and compared.

Key words: Computational intelligence, neural networks, fuzzy systems.

1. INTRODUCTION

Conventional controllers such as PID, and many advanced control methods are useful to control linear processes. In practice, most processes are nonlinear. Nonlinear control is one of the biggest challenges in modern control theory. While linear control system theory has been well developed, it is the nonlinear control problems that cause most challenges. Traditionally, a nonlinear process has to be linearized first before an automatic controller can be effectively applied. This is typically achieved by adding a reverse nonlinear function to compensate for the nonlinear behavior so the overall process of the input-output relationship becomes somewhat linear. The issue becomes more complicated if a nonlinear characteristic of the system changes with time and there is a need for an adaptive change of the nonlinear behavior. These adaptive systems are best handled with methods of computational intelligence such as neural networks and fuzzy systems [1][2]. This presentation will focus on several methods of developing close to optimal architectures and on finding efficient learning algorithms.
Any dynamic nonlinear system can be described by the following set of nonlinear state equations:

$$\begin{aligned} y_1 &= \int f_1(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_n)\ dt \\ y_2 &= \int f_2(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_n)\ dt \\ &\cdots \\ y_n &= \int f_n(x_1, x_2, \cdots x_n, y_1, y_2, \cdots y_n)\ dt \end{aligned} \tag{1}$$

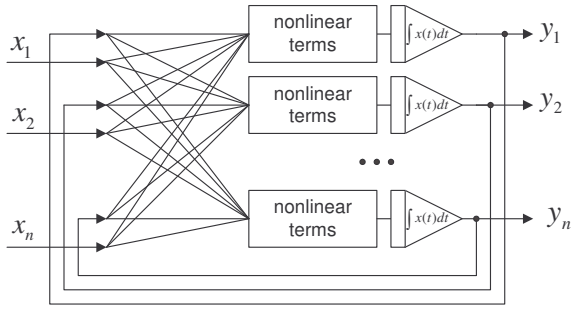

Fig. 1. Block diagram of nonlinear system derived from a set of state variable equations.

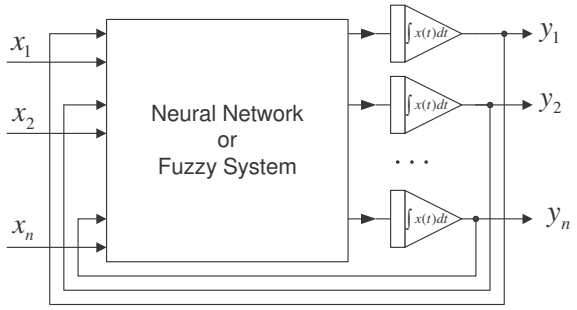

Fig. 2. Block diagram of nonlinear dynamic system using neural networks or fuzzy systems.

Such a system can be implemented as a composition of integrators and nonlinear terms as shown in Fig. 1. Implementation of analog integrators on silicon chips is relatively simple. It requires a capacitance and an operational or transcounductance amplifier. Nonlinear terms with multiple inputs are most difficult to implement. These nonlinear blocks can be developed as universal elements using neural networks or fuzzy systems (Fig. 2). In both cases these nonlinear terms can be digitally controlled. In the case of neural networks only weights

need to be digitally controlled. In the case of the fuzzy systems parameters of fuzzifiers and deffuzifiers have to be digitally adjusted. In both cases signals can always be in analog form. This analog type of signal processing is especially important in systems where a large signal latency is not acceptable.

2. FUZZY SYSTEMS

The fuzzy set system theory was developed by Zadeh [3]. The block diagram of a typical fuzzy system, as proposed by Mamdani [4] is shown in Fig. 3.

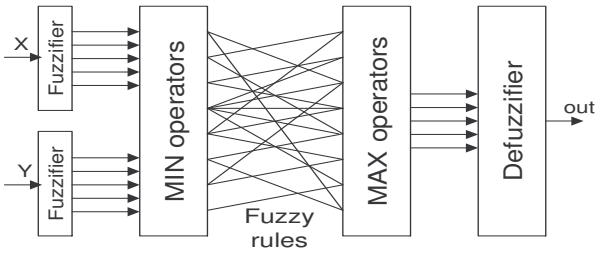

Fig. 3. Block diagram of a Mamdani type fuzzy controller.

At the left side of the diagram, analog inputs are converted by fuzzifiers into sets of fuzzy variables. For each analog input, several fuzzy variables typically are generated. Each fuzzy variable has an analog value between zero and one. Various types of fuzzification methods can be used as shown in Fig. 4.

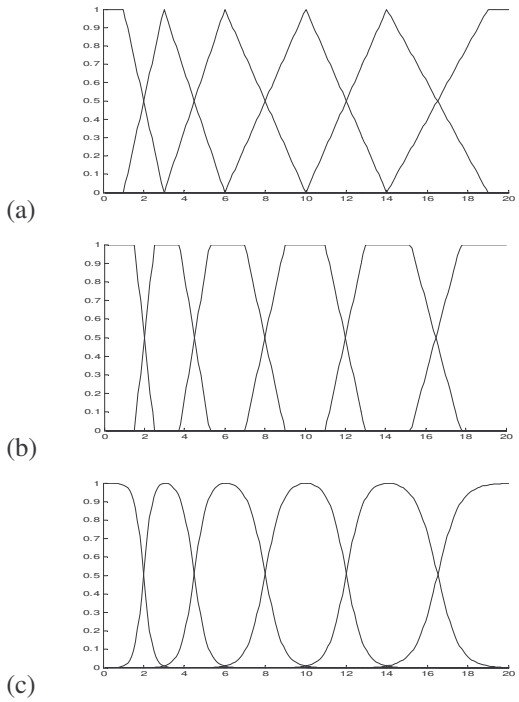

Fig. 4. Membership functions for various fuzzification methods: (a) triangular, (b) trapezoidal, and (c) Gaussian.

Each point of the input analog variable should belong to at least one, and preferably no more than two, membership functions. For overlapping functions, the sum of two membership functions must not be larger than one. This also means that overlap must not cross the points of maximum values (ones).

For higher accuracy, more membership functions should be used. However, very dense functions can lead to frequent controller action (also known as "hunting"), and sometimes this may lead to system instability.

If the required nonlinear function has the shape as shown in Fig. 5, then actual implemented functions will have slightly different shapes depending on the type of fuzzification used (Fig. 6). In most cases best results are obtained with triangular fuzzifiers, as shown in Fig. 6(a)

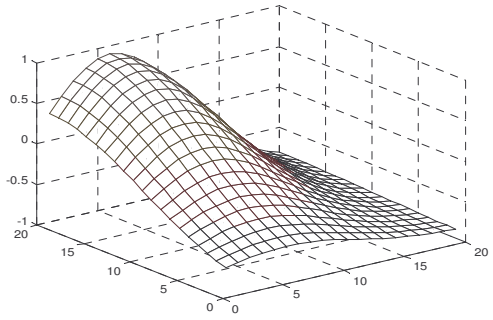

Fig. 5. Required nonlinear function

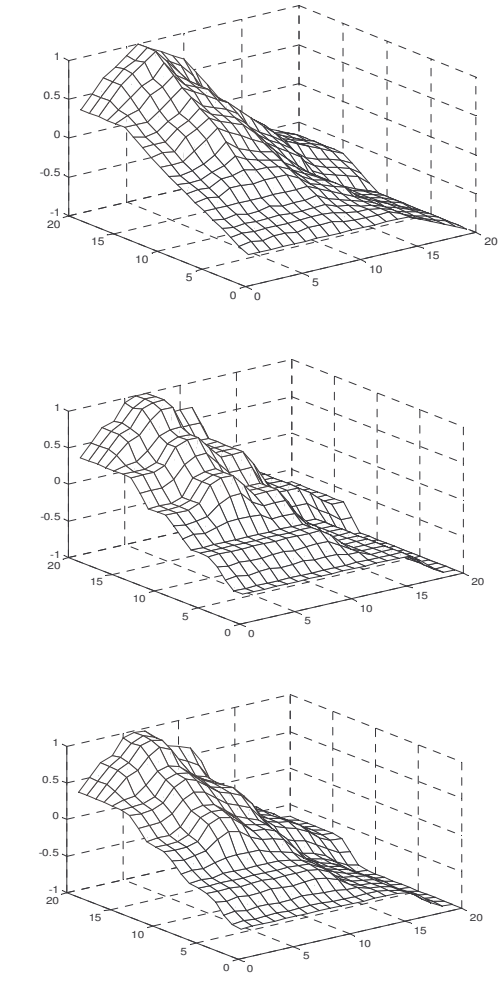

Fig. 6. Different control surfaces obtained with different fuzzification methods: (a) triangular, (b) trapezoidal, and (c) Gaussian.

In the center of the Mamdani control diagram of Fig. 3, fuzzy variables from fuzzifiers are processed by fuzzy logic blocks with MIN and MAX operators. The fuzzy

logic is similar to Boolean logic but instead of AND operators, MIN operators are used and in place of OR operators, MAX operators are implemented.

Interestingly fuzzy logic has a more general nature and it works equally well as Boolean logic. Fig. 7 shows fuzzy logic operations on zero-one Boolean variables (Fig. 7(a)) and on fuzzy variables (Fig. 7(b)).

(a)

MIN $A \cap B$			MAX $A \cup B$		
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	1

(b)

MIN		$A \cap B$	MAX		$A \cup B$
0.2	0.3	0.2	0.2	0.3	0.3
0.2	0.8	0.2	0.2	0.8	0.8
0.7	0.3	0.3	0.7	0.3	0.7
0.7	0.8	0.7	0.7	0.8	0.8
		union			***intersection***

Fig. 7. Comparison of (a) Boolean and (b) Fuzzy logic.

The Mamdani concept follows the rule of ROM and PLA digital structures where AND operators are selecting specified addresses and then OR operators are used to find the output bits from the information stored at these addresses. Also, in the case of the fuzzy system, as presented in Fig. 3, first MIN and then MAX operators are used.

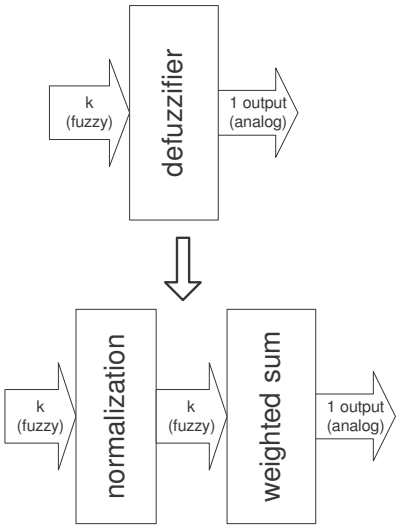

Fig. 9. Conversion from Mamdani to TSK fuzzy architecture.

In some implementations, MIN operators are replaced by product operators (signals are multiplied). Fuzzy systems with product encoding are more difficult to implement but they generate a slightly smoother control surface. Fig. 8 shows the surface obtained with product encoding, which is smoother than the surface of Fig. 6(a) which is obtained with a MIN encoding.

As a result of fuzzy logic block of Fig. 3 a new set of fuzzy variables is generated, which later has to be converted to an analog output value. The rightmost block of the diagram represents defuzzification, where the output analog variable is retrieved from a set of output fuzzy variables. Several more or less complicated defuzzification schemes are used. The most common is the centroid type of defuzzification.

More recently Mamdani architecture was replaced by TSK (Takagi, Sugeno, Kang) [4][5] architecture where the defuzzification block was replaced with normalization and weighted average (see Fig. 9)

The TSK structure, as shown in Fig. 10, also does not require MAX operators, but a weighted average is applied directly to regions selected by MIN operators. What makes the TSK system really simple is that the output weights are proportional to the average function values at the selected regions by MIN operators.

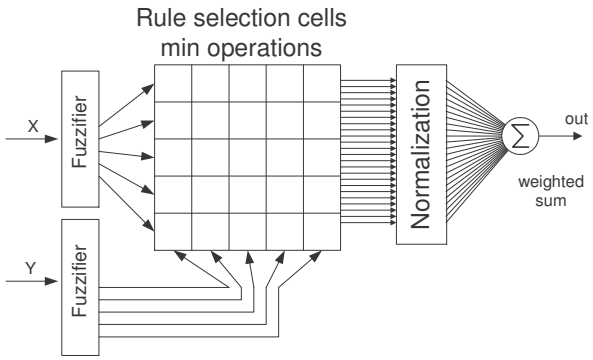

Fig. 10. TSK (Takagi-Sugeno-Kang) fuzzy architecture.

There were many attempts to further improve fuzzy controllers by replacing fuzzifiers and MIN operators by other weighted sum approaches and RBF (Radial Base Function) networks [6]. These areas of research are known as fuzzy-neuro systems and the resulting architectures are more close to neural networks than to fuzzy systems.

3. NEURAL NETWORKS

A single neuron can divide input space by line, plane, or hyperplane, depending on the problem dimensionality. In order to select just one region in *n*-dimensional input space, more than *n+1* neurons should be used. For example, to separate a rectangular pattern 4 neurons are required, as is shown in Fig. 11. If more input clusters should be selected then the number of neurons in the hidden layer should be properly multiplied. If the number of neurons in the hidden layer is not limited, then all classification problems can be solved using the three layer network.

With the concept shown on Fig. 11 fuzzifiers and MIN operators used for region selection can be replaced by a simple neural network architecture. Let us analyze Fig. 12 where a two-dimensional input space was divided by six neurons horizontally and by six neurons vertically. The corresponding neural network is shown in Fig. 13. Each neuron is connected only to one input. For each neuron input, weight is equal to +1 and the threshold is equal to the value of the crossing point on the x or y axis. Neurons

in the second layers have two connections to lower boundary neurons with weights of +1 and two connections to upper boundary neurons with weights of -1. Thresholds for all these neurons in the second layer are set to 3. Only two of them are drawn on Fig. 13. One of them is selecting region A and the second is selecting region B (Fig. 12).

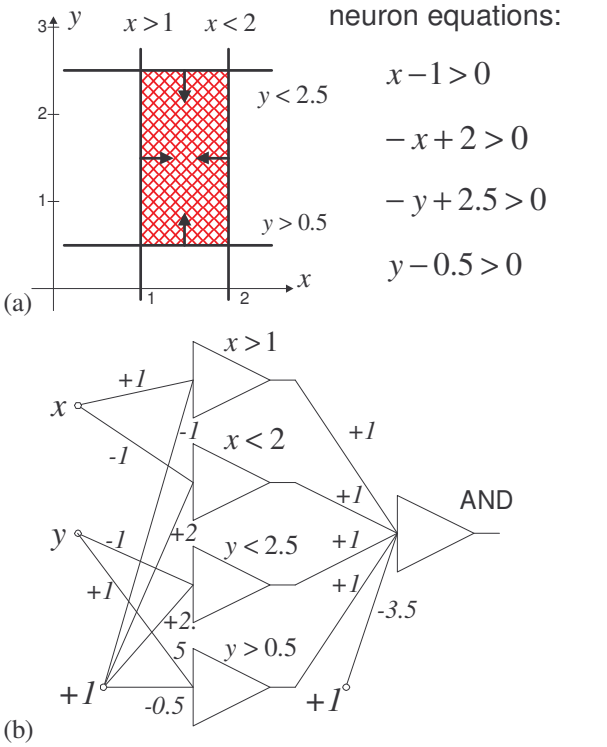

Fig. 11. Separation of the rectangular area on a two dimensional space (a) and desired neural network to fulfill this task (b).

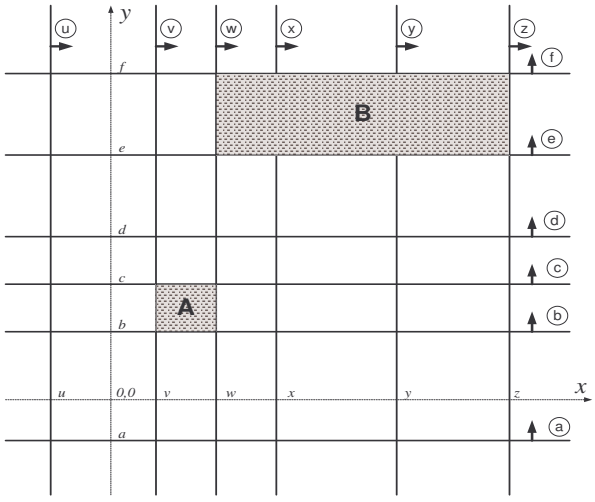

Fig. 12. Two-dimensional input plane separated vertically and horizontally by six neurons in each direction.

Weights in the last layer have values corresponding to the expected function values in selected areas. All neurons in Fig. 13 have a unipolar activation function and if the system is properly designed, then for any input vector in certain areas only the neuron of this area produces +1 while all remaining neurons have zero values. In the case of when the input vector is close to a boundary between two or more regions, then all participating neurons are producing fractional values and the system output is generated as a weighted sum. For proper operation it is important that the sum of all outputs of the second layer must be equal to +1. In order to assure the above condition, an additional normalization block can be introduced, in a similar way as it is done in TSK fuzzy systems as shown in Fig. 10.

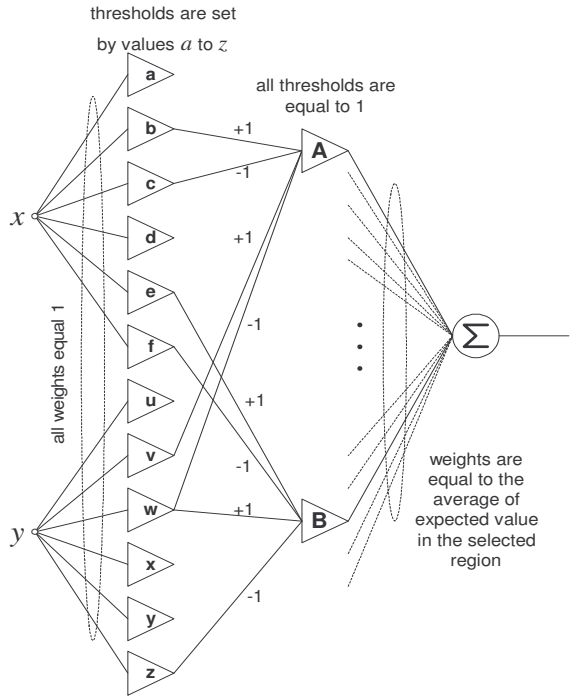

Fig. 13. Simple neural networks performing the function of TSK fuzzy system.

It was shown above that a simple neural network of Fig. 13 can replace a fuzzy system. All parameters of this network are directly derived from requirements specified for a fuzzy system and there is no need for a training process.

One may observe that if the training process is allowed then the network architecture of Fig. 13 can be significantly simplified. Let us compare in the following subsections the commonly used neural network architectures.

3.1. General feedforward networks

The most commonly used neural network architectures are shown in Fig. 14. The most used learning algorithm such as EBP – Error Back Propagation or LM - Levenberg-Marquardt, were developed for this type of feedforward network.

This feedforward network can become much more powerful if weight connections across layers are allowed. Unfortunately only very few software packages are capable of training fully connected neural networks.

3.2. Functional link and polynomial networks

One layer neural networks are relatively easy to train, but these networks can solve only linearly separated problems. One possible solution for nonlinear problems

was elaborated by Pao [8] using the functional link network (shown in Fig. 15). Note that the functional link network can be treated as a one layer network, where additional inputs are generated offline using nonlinear transformations. If nonlinear terms are generated using a polynomial function then this network is known as a polynomial network. These networks are extremely easy to train, but it is usually not known what type of nonlinear functions are best suited for specific problems. In the case of polynomial networks a more generalized approach is possible, but with an increase in the dimensionality of the problem the number of polynomial terms grow exponentially and these networks become impractical.

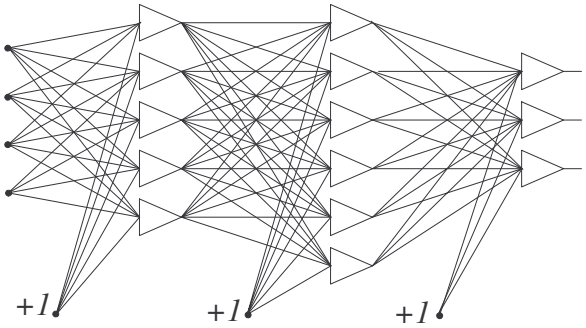

Fig. 14. Feedforward neural network with two hidden layers.

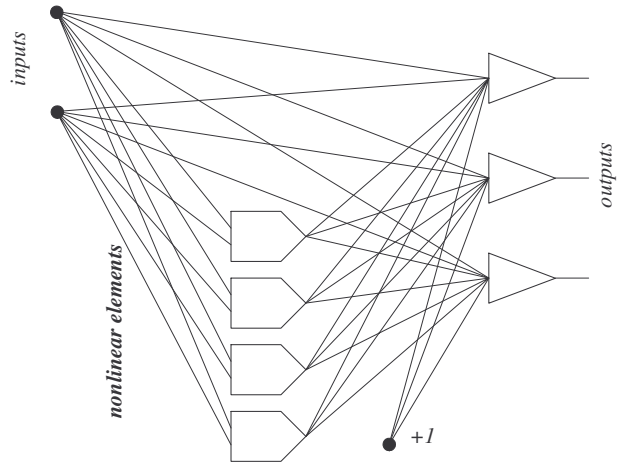

Fig. 15. Functional link and polynomial networks.

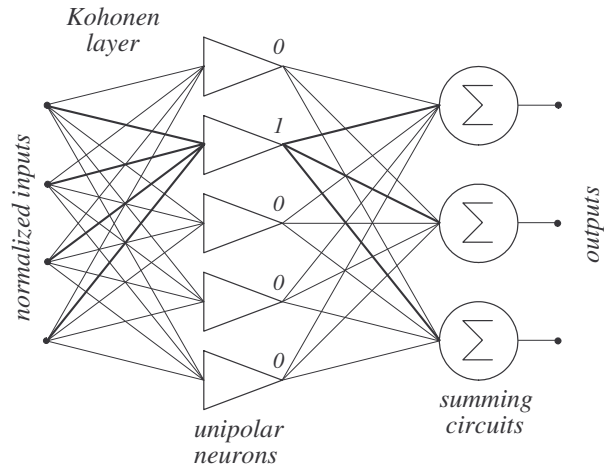

Fig. 16. Counterpropagation networks.

3.3. Counterpropagation networks

Counterpropagation networks were originally proposed by Hecht-Nilsen [9]. This architecture, which is shown in Fig. 16, requires several hidden neurons which are equal to the number of input patterns, or more exactly, to the number of input clusters.

When binary input patterns are considered, then the input weights must be exactly equal to the input patterns. Since for a given input pattern, only one neuron in the first layer may have the value of one, and the remaining neurons have zero values, the weights in the output layer are equal to the required output pattern.

The counterpropagation network is very easy to design. The number of neurons in the hidden layer should be equal to the number of patterns (clusters). The weights in the input layer should be equal to the input patterns and, the weights in the output layer should be equal to the output patterns. A disadvantage of the counterpropagation network is that number of neurons in the hidden layer must be equal to number of training patterns and this number is sometimes excessively large.

3.4. LVQ Learning Vector Quantization networks

LVQ networks are derived from counterpropagation networks by combining some patterns into clusters. By doing this the size of the network is reduced. In the LVQ network the first layer detects subclasses. The second layer combines subclasses into a single class (Fig. 17). The first layer computes Euclidean distances between input patterns and stored patterns. A winning "neuron" is the one with the smallest distance in the input pattern.

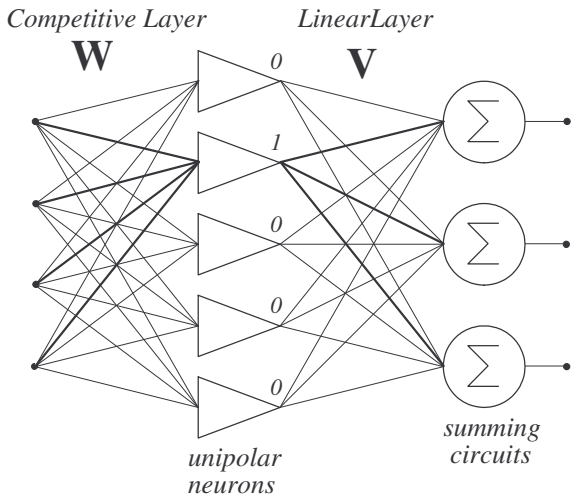

Fig. 17. LVQ Learning Vector Quantization networks.

3.5. Cascade correlation architecture

The cascade correlation architecture was proposed by Fahlman and Lebiere [10] (Fig. 18) The process of network building starts with a one layer neural network and hidden neurons are added as needed. In each training step, the new hidden neuron is added and its weights are adjusted to maximize the magnitude of the correlation between the new hidden neuron output and the residual error signal on the network output that we are trying to

eliminate. The output neurons are trained using a simple one-neuron training algorithm. Each hidden neuron is trained just once and then its weights are frozen. The network learning and building process is completed when satisfactory results are obtained.

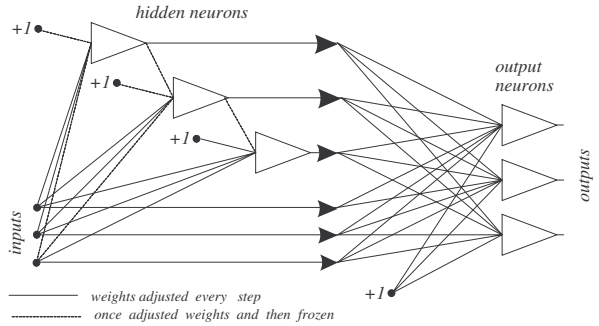

Fig. 18. Cascade correlation architecture.

3.6. RBF - Radial Basis Function networks

The structure of the radial basis network is shown in Fig. 19. This type of network usually has only one hidden layer with special "neurons". Each of these "neurons" responds only to the input signals close to the stored pattern. The output signal h_i of the *i-th* hidden "neuron" is computed using the following formula.

$$h_i = \exp\left(-\frac{\|\mathbf{x}-\mathbf{s_i}\|^2}{2\sigma^2}\right) \tag{2}$$

Note, that the behavior of this "neuron" significantly differs from the biological neuron. In this "neuron", excitation is not a function of the weighted sum of the input signals. Instead, the distance between the input and stored pattern is computed. This "neuron" is capable of recognizing certain patterns and of generating output signals that are functions of a similarity.

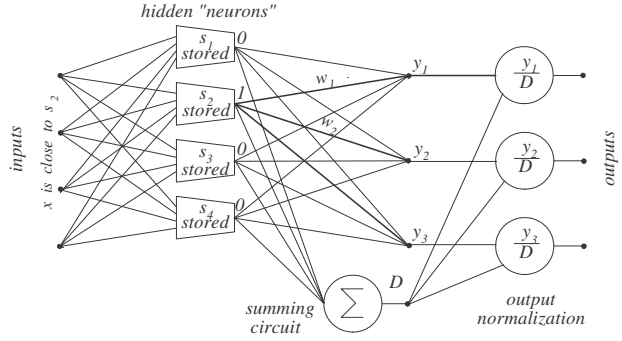

Fig. 19. RBF - Radial basis function networks.

3.7. Sarajedini and Hecht-Nielsen network

The Sarajedini and Hecht-Nielsen [11] network of Fig. 20 is capable of calculating Euclidean distances between input pattern and stored pattern using only information about the square of the input vector length and the neuron with a linear activation function. The network is based on the following analytical formulas:

$$\|\mathbf{x}-\mathbf{w}\|^2 = \mathbf{x}^T\mathbf{x} + \mathbf{w}^T\mathbf{w} - 2\mathbf{x}^T\mathbf{w} \tag{3}$$

and

$$\|\mathbf{x}-\mathbf{w}\|^2 = \|\mathbf{x}\|^2 + \|\mathbf{w}\|^2 - 2net \tag{4}$$

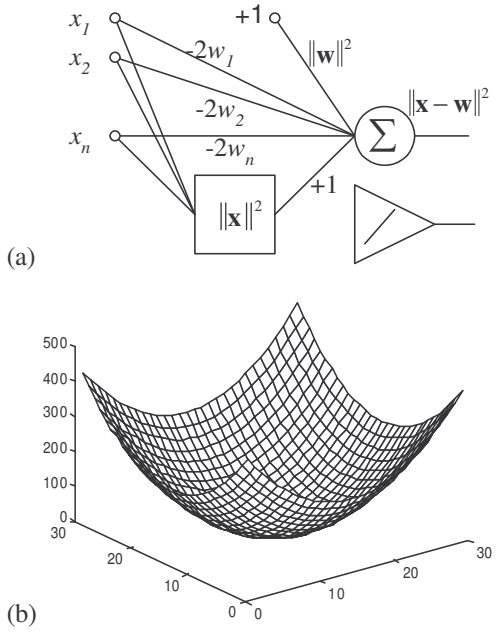

Fig. 20. Sarajedini and Hecht-Nielsen neural network: (a) network architecture, (b) surface generated (for 2-dim case).

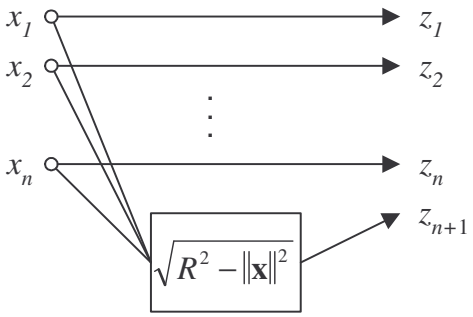

Fig. 21. Networks with increased dimensionality.

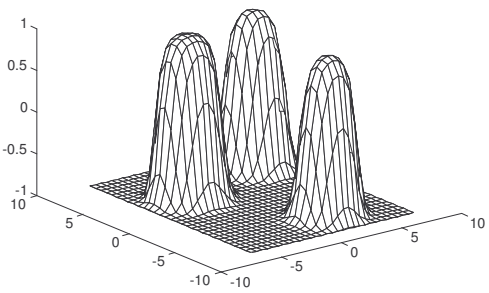

Fig. 22. Result of cluster separation of networks with increased dimensionality.

3.8. Networks with increased dimensionality

The network shown in Fig. 21 has a similar property (and power) to RBF networks, but it uses only traditional neurons with sigmoidal activation functions [12][13]. In this network an additional input is generated using the formula:

$$z_{n+1} = \sqrt{R^2 - \|\mathbf{x}\|^2} \tag{5}$$

This way all input patterns are projected on a hyper sphere with a radius *R*. Round clusters could be separated by hyper planes (traditional sigmoidal neurons).
Fig. 22 shows a separation of three clusters using three sigmoidal neurons. With this approach traditional neurons are gaining the capability of separating patterns by circle, sphere, or hyper sphere.

3.9. Comparison on neural network architectures

One of the most difficult problems to solve with neural networks is the parity problem. This problem has a very nonlinear character with multiple minimas and maximas [14]. Let us compare different neural network architectures to solve the parity-8 problem. This problem is so complex that the most common EBP algorithm is not able to solve it, unless, luckily, initial starting weights are used. In the case of the most popular neural networks with one hidden layer and without connections across layers there are at least 9 neurons required and 8*9+9 = 81 weights. See Fig. 23 for the network architecture.

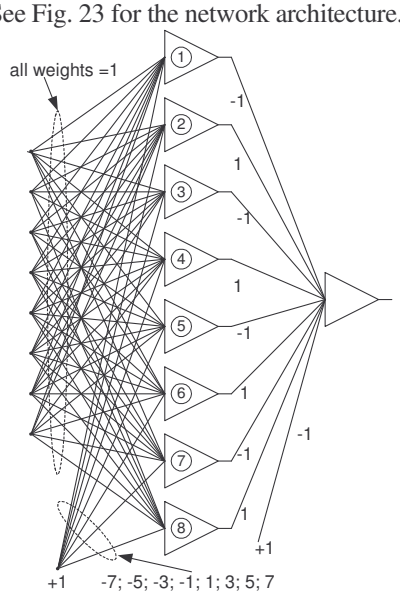

Fig. 23. Parity-8 problem with feedforward bipolar neural network with one hidden layer.

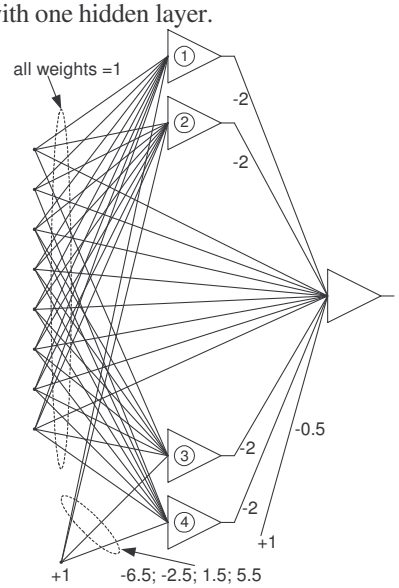

Fig. 24. Fully-connected layered bipolar neural network with one hidden layer for the parity-8 problem.

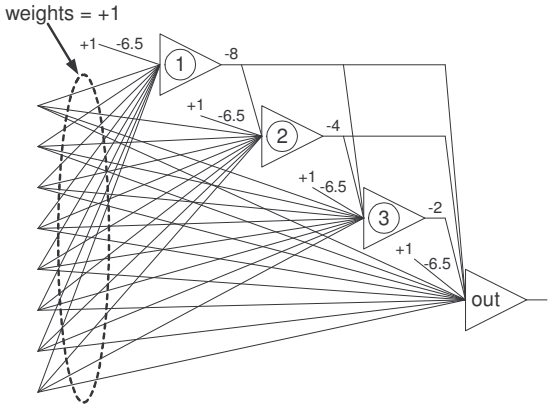

Fig. 25. Bipolar implementation of a fully connected cascade neural network for the parity-8 problem.

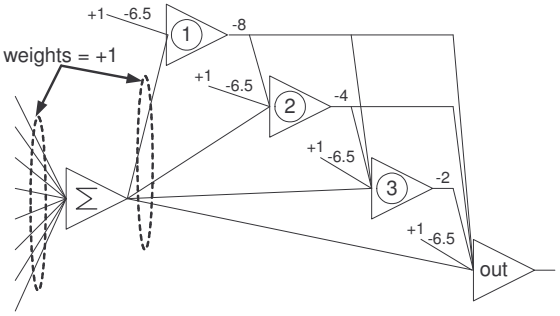

Fig. 26. Bipolar implementation of a fully connected cascade neural network for the parity-8 problem with an additional summator.

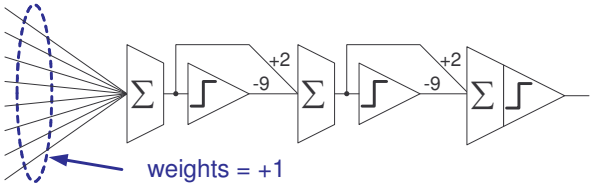

Fig. 27: Bipolar implementation of a pipeline neural network for the parity-8 problem.

In the case of when connections across layers are allowed (see Fig. 24), number of neurons in the hidden layer can be reduced from 8 to 4 and the total number of weights is 4*9+8+4+1=49. When neurons are connected in a cascade (see Fig. 25), then only four neurons are required and the total number of weights is 9+10+11+12=42.
Note, that in the case of the parity problem (due to the symmetry of inputs) each of networks shown in Fig 23 to Fig 25 can be further simplified by adding an additional neuron with linear activation function (summator) to the front. For example, the network of Fig. 25 can be simplified to the architecture shown in Fig. 26 with 5 neurons and 22 weights.
With the pipeline architecture shown in Fig. 27 the parity-8 problem can be solved with only 3 neurons and 8+2+2=12 weights. The pipeline architecture is very specific to parity problems and it cannot be generalized for other cases.
The comparison of minimum hardware requirements for the parity-8 problem is shown in Table I.

TABLE I. Comparison of various neural network architectures for implementation of the parity 8 problem

	neurons	weights
NN with one hidden layer (Fig. 23)	9	81
NN with one hidden layer with connections across (Fig. 24)	5	49
NN with fully connected cascade (Fig. 24)	4	42

One may conclude that the cascade network (Fig. 25) is the most powerful, since it would require a minimum number of elements. At the same time because of a long signal path (across many layers), the cascade architecture is more difficult to train and it is also more sensitive to the variation of weights. The fully connected network (Fig. 14) has only a slightly larger number of neuronal requirements than cascade architecture but it is easier to train and in most cases this would be the preferred choice.

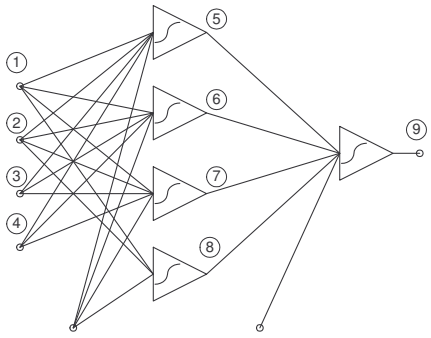

Fig. 28. Layered neural network with four neurons in the hidden layer.

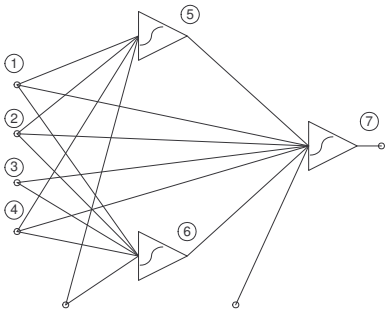

Fig. 29. Fully connected neural network with two neurons in the hidden layer.

3.10. Training algorithms

Unfortunately most of the neural network software (like MATLAB Tool Box) is not suitable for training fully connected neural networks. One exception is the SNNS (Stuttgart Neural Network System [15]), which can handle fully connected architectures, but only relatively simple neuron network training algorithms are used and the LM - Levenberg-Marquardt [16] algorithm is not implemented. The LM algorithm has currently the best reputation out of all the training algorithms. For most cases it converges within 10-20 iterations while the most popular EBP – Error Back Propagation-algorithm requires 1000 to 2000 iterations to reach a solution with a relatively large error. An additional advantage of the LM algorithm is its fast convergence to the solution, while the EBP reaches the solution only asymptotically.

The author has developed a code for a LM algorithm that is suitable for any neural network architecture (including cascade and fully connected networks) The MATLAB code can be downloaded from [17]. In the current version of the software all neural network nodes have to be numbered sequentially starting from inputs to outputs. The entire network architecture is described by a sequence of numbers. For example the network shown in Fig 28 is described by the sequence: 5, 1, 2, 3, 4, 6, 1, 2, 3, 4, 7, 1, 2, 3, 4, 8, 1, 2, 3, 4, 9, 5, 6, 7, 8 while the network shown in Fig. 29 is described by the sequence: 5, 1, 2, 3, 4, 6, 1, 2, 3, 4, 7, 1, 2, 3, 4, 5, 6. In the numerical sequence the number of neurons is listed first and then all input nodes, and then the number of the next neuron is given with all its associated inputs. The process is repeated until all neurons are listed. Note, that the software can handle fully or sparsely connected networks with arbitrary architectures, as long as the concept of a one directional signal flow is preserved. The software has the option of giving initial weights. Weights are listed in the same form as the network topology. Instead of neuron number, the biasing weight of this neuron is given and then weights associated with every input. If the initial weights are not given, then all the weights are selected randomly.

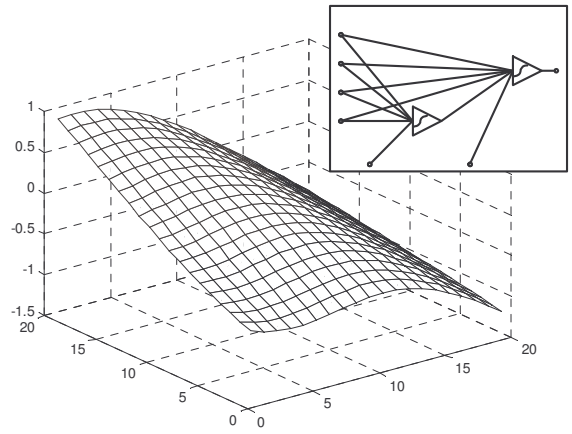

Fig. 30. Control surface obtained using fully connected neural network with one hidden neuron.

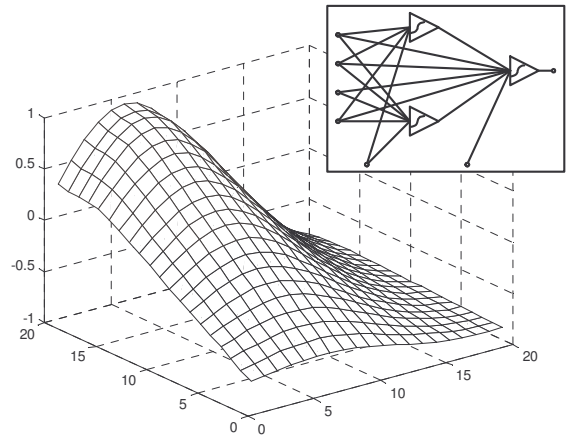

Fig. 31. Control surface obtained using fully connected neural network with two hidden neurons.

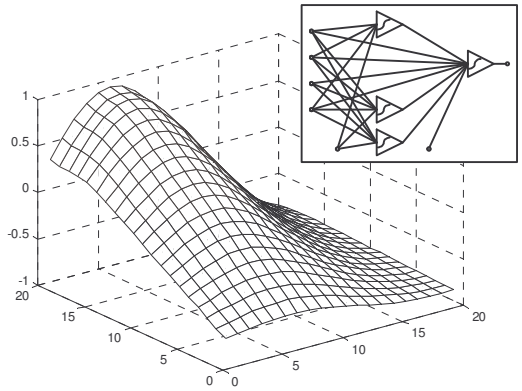

Fig. 32. Control surface obtained using fully connected neural network with three hidden neurons.

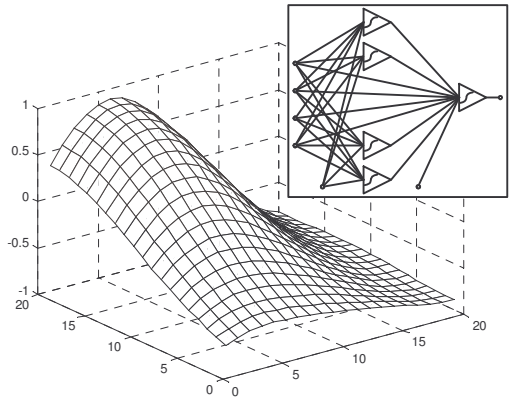

Fig. 33. Control surface obtained using fully connected neural network with four hidden neurons.

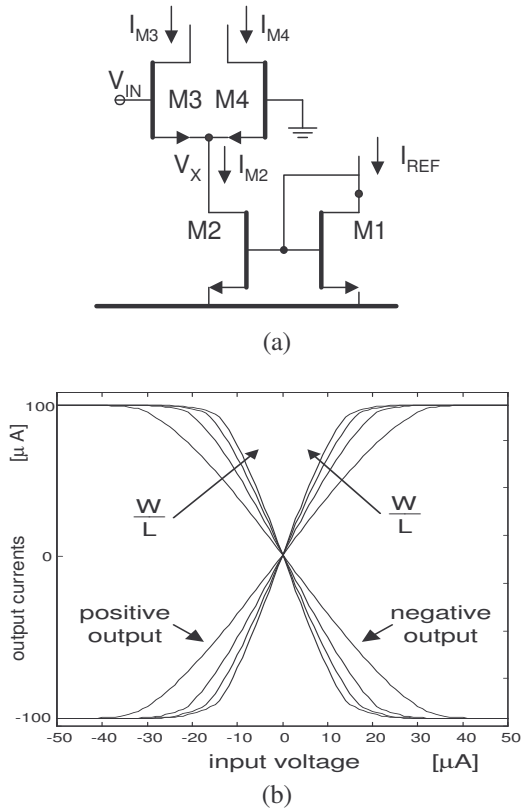

Fig. 34. Simple VLSI implementation of neuron with a differential pair: (a) circuit diagram (b) result of SPICE simulation.

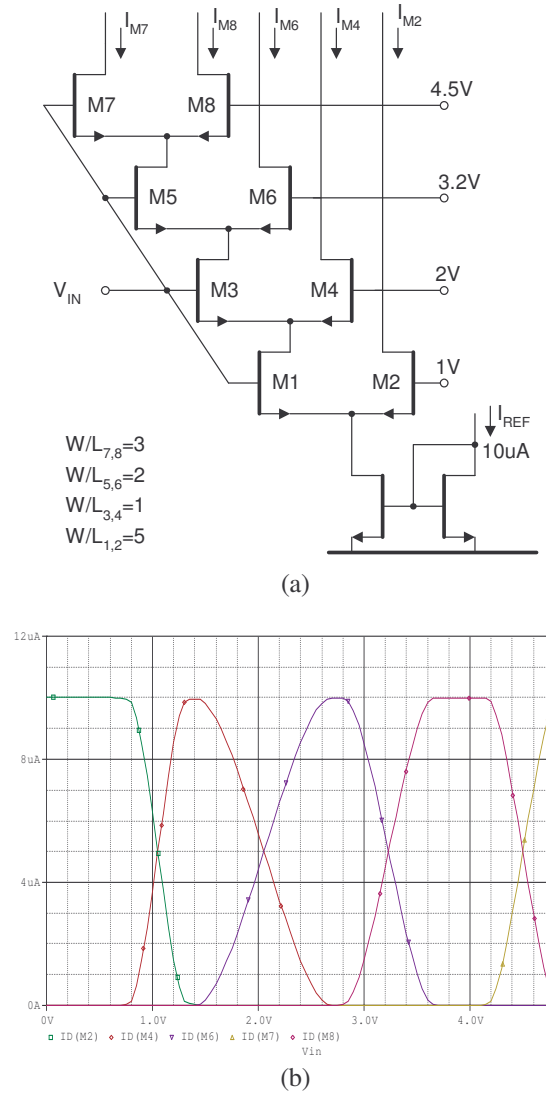

Fig. 35. VLSI implementation of fuzzifier block: (a) circuit diagram, (b) result of SPICE simulations with five membership functions plotted.

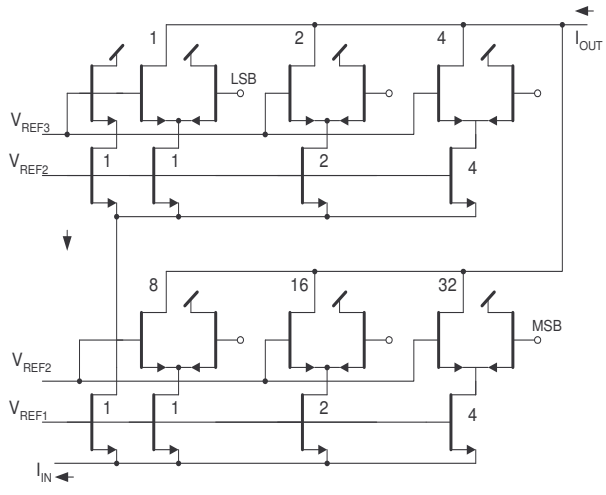

Fig. 36. Digitally programmable weights.

4. VLSI IMPLEMENTATIONS

In the case of neural networks a sigmoidal type of activation function can be implemented using a simple differential pair (Fig 34). Positive or negative weights can be implemented by taking a signal to the next layer from inverted and non inverted outputs.
One possible solution is to use current controlled weights in neural networks and current controlled parameters in fuzzy systems. Therefore, in order to fully control nonlinear systems, only digitally controlled currents are required. The same differential pairs with a unique configuration, as shown in Fig. 35(a), may act as fuzzifiers. Fig. 35(b) shows membership functions implemented by the circuit of Fig. 35(a). Fig. 36 shows a sample solution of digitally programmed 6-bit weights.

5. CONCLUSION

Fuzzy systems and neural networks as two major methods of computational intelligence were described and compared. Fuzzy systems are easier to design, while neural networks require training (optimization). In practical implementations fuzzy systems require more hardware and resulted control surface is not as smooth as in the case of neural networks. For example, in the study case the TSF fuzzy system with triangular membership function (see Fig. 6(b)) required 6+6+36= 48 values to be stored. In the case of fully connected neural network with three hidden neurons (Fig. 32), only 3*4+8=20 values had to be stored. One may notice that neural networks require not only less hardware, but also it generates superior control surface. In the case when only design rules has to be used and optimization is not desired, neural networks can also replace fuzzy systems as it was shown in Fig. 13.

IMPLEMENTACJE METOD KOMPUTEROWEJ INTELIGENCJI

Streszczenie: Tradycyjne kontrolery takie jak PID i wiele zaawansowanych metod sterowania są użyteczne dla sterowania liniowych procesów. W praktyce większość procesów jest nieliniowa. Metody nieliniowego sterowania są głównym problemem w nowoczesnej teorii sterowania. Nawet, jeśli liniowa teoria sterowania jest już dobrze ugruntowana, to problemy nieliniowego sterowania przysparzają nam wiele kłopotów. Sterowanie nieliniowych systemów nie jest łatwe, ponieważ występować w nich może wiele trybów nieliniowych zachowań. Zwykle nieliniowe procesy muszą być najpierw zlinearyzowane zanim sterownik będzie skuteczny. Można to zwykle osiągnąć poprzez dodanie odwrotnej funkcji nieliniowej dla skompensowania nieliniowych zachowań. W ten sposób, wypadkowy proces pomiędzy wejściami i wyjściami systemu będzie w przybliżeniu liniowy. Zagadnienie staje się bardziej skomplikowane, jeśli charakterystyki systemu zmieniają się w czasie i potrzebna jest adaptacyjna zmiana nieliniowych właściwości. Takie systemy adaptacyjne są skutecznie tworzone przy wykorzystaniu metod komputerowej inteligencji takich jak sieci neuronowe lub systemy rozmyte. Kolejnym problemem jest opracowywanie systemów rozmytych i sieci neuronowych, które same w sobie są zagadnieniami bardzo złożonymi. Prezentacja skupi się na kilku metodach znajdywania zbliżonych do optymalnych architektur i efektywnych metod uczenia systemów. Problemy stają się jeszcze bardziej złożone, jeśli metody komputerowej inteligencji mają być zaimplementowane w krzemie. Szereg praktycznych rozwiązań zostało zaprezentowanych i porównanych.

References

[1] Duch W., Korbicz J., Rutkowski L., Tadeusiewicz R. (2000) *Sieci Neuronowe.* Akademicka Oficyna Wydawnicza EXIT, Warszawa.
[2] Wilamowski B.M. (2002) Neural Networks and Fuzzy Systems", chapter 32 in *Mechatronics Handbook* edited by Robert R. Bishop, CRC Press, pp. 33-1 to 32-26.
[3] Zadeh L.A. (1965) Fuzzy sets. *Information and Control*, **8**, 338-353.
[4] Mamdani E.H. (1974) Application of Fuzzy Algorithms for Control of Simple Dynamic Plant. *IEEE Proceedings*, **121**, 12, 1585-1588.
[5] Sugeno and G.T. Kang (1988) Structure Identification of Fuzzy Model. *Fuzzy Sets and Systems*, **28**, 1, 15-33.
[6] Takagi T., Sugeno M. (1985) Fuzzy Identification of Systems and Its Application to Modeling and Control. *IEEE Transactions on System, Man, Cybernetics*, **15**, 1, 116-132.
[7] Pao Y.H. (1989) *Adaptive Pattern Recognition and Neural Networks*, Reading, Mass. Addison-Wesley Publishing Co.
[8] Hecht-Nielsen R. (1987) Counterpropagation networks. *Appl. Opt.*, 26(23), 4979-4984.
[9] Fahlman S.E, Lebiere C. (1990) The cascade-correlation learning architecture. nn D.S.Touretzky, Ed. *Advances in Neural Information Processing Systems* 2, Morgan Kaufmann, San Mateo, Calif., 524-532.
[10] Sarajedini A., Hecht-Nielson R. (1992) The best of both worlds: Casasent networks integrate multilayer perceptrons and radial basis functions. *IJCNN'92. International Joint Conference on Neural Networks* 7-11 Jun 1992, vol.3, 905-910.
[11] Wilamowski B., Hunter D. (2003) Solving Parity-n Problems with Feedforward Neural Network. *Proc. of the IJCNN'03 International Joint Conference on Neural Networks*, pp. 2546-2551, Portland, Oregon, July 20-23, 2003.
[12] University of Tübingen. [Online]. Available: http://www-ra.informatik.uni-tuebingen.de/SNNS/
[13] Hagan, M. T. and Menhaj, M., "Training feedforward networks with the Marquardt algorithm", IEEE Transactions on Neural Networks, vol. 5, no. 6, pp. 989-993, 1994.
[14] Auburn University [Online]. Available: http://www.eng.auburn.edu/users/wilambm/BMW.zip.
[15] Wilamowski B.M., Binfet J. (1999) Do Fuzzy Controllers Have Advantages over Neural Controllers in Microprocessor Implementation. *ICRAM'99 2-nd International Conference on Recent Advances in Mechatronics* -, Istanbul, Turkey, pp. 342-347, May 24-26 1999.
[16] Wilamowski B.M., Hung J.Y., Gottiparthy R. (2005) Digitally Tuned Analog VLSI Controllers. *ISIE'05 IEEE International Symposium on Industrial Electronics,* Dubrovnik Croatia, June 19-22 2005.