Methods of Computational Intelligence for Nonlinear Control Systems

Bogdan M. Wilamowski

Auburn University, Alabama Microelectronic, Science and Technology Center, 200 Broun Hall,
(Tel : +1-344-844-1629; E-mail: wilam@ieee.org)

Abstract: Conventional controllers like 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 headaches. 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 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. The problem is that development of neural or fuzzy systems are not trivial. 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.

Keywords: Computational intelligence, neural networks, fuzzy systems.

1. INTRODUCTION

It is relatively easy to control linear systems. Unfortunately, in practice most systems are nonlinear. Some of them can be linearized and use well developed linear control theory, but in many cases a special nonlinear control system has to be developed. Nonlinear control is one of the biggest challenges in modern control theory. 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. In many cases nonlinear characteristics of the system change with time and there is a need for an adaptive change of the nonlinear behavior. These adaptive systems are best handled with neural networks and fuzzy systems [1][2]. In this presentation various fuzzy and neural network architectures will be studied and compared.

Any dynamic nonlinear system can be described by the set of nonlinear state equations:

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

Implementation of analog integrators on silicon chips is relatively simple. Nonlinear terms with multiple inputs are more difficult to implement. These nonlinear blocks can be developed as universal elements using neural networks or fuzzy systems (Fig. 1). In both cases, fuzzy and neural systems, these nonlinear terms can work in analog mode while they can be digitally tuned. In the case of neural networks only weights need to be digitally controlled. In the case of the fuzzy systems parameters of fuzzifiers and defuzifiers, they 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 large signal latency is not acceptable.

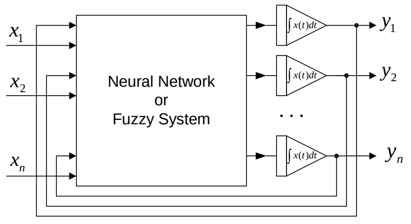


Fig. 1. Block diagram of nonlinear dynamic system using neural networks or fuzzy systems.

2. FUZZY SYSTEMS

There are two most commonly used approaches for development of Fuzzy systems. Fig. 2 shows architecture proposed by Mamdani [3] and Fig. 3 shows architecture proposed by Takagi, Sugeno, and Kang [4][5]. There were also attempts to present a fuzzy system in a form of neural network. One of such networks is shown in Fig. 4 [6]. Note that only architecture resembles neural networks because cells there perform different functions than neurons, such as signal multiplication or division.

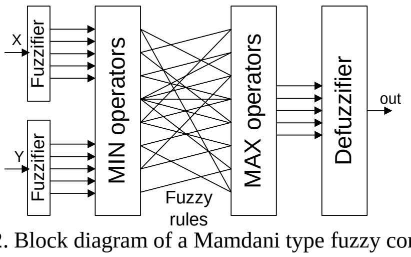


Fig. 2. Block diagram of a Mamdani type fuzzy controller.

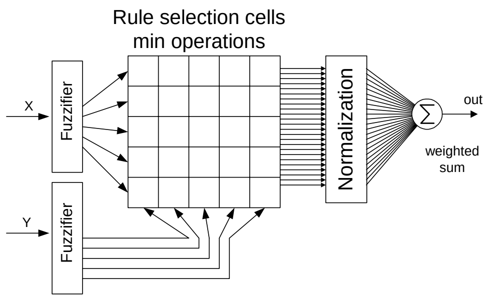


Fig. 3. TSK (Takagi-Sugeno-Kang) fuzzy architecture.

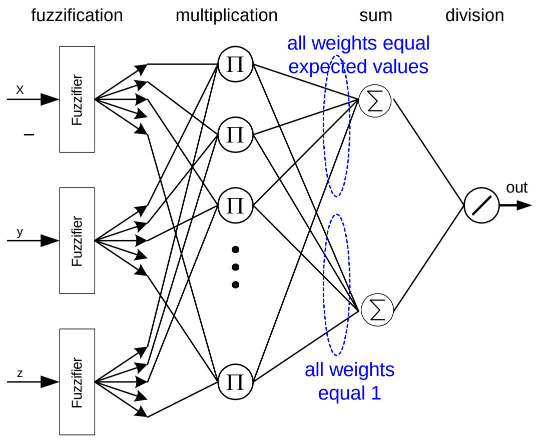


Fig. 4. Fuzzy neural networks [6]

2.1 Fuzzification

In fuzzy systems, at first all analog inputs are converted by fuzzifiers into sets of fuzzy variables. For each analog input, several fuzzy variables, with values between 0 and 1, are generated. Various types of fuzzification methods can be used as triangular, trapezoidal, or Gausian. One may consider mixing these techniques. The simplest implementation is with triangular membership functions and in most practical cases acceptable results are obtained with this simplest approach.

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.

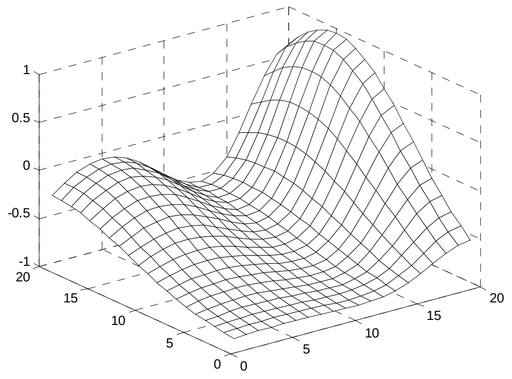


Fig. 5. Desired nonlinear control surface

Let us assume that the required nonlinear function has a shape as shown in Fig. 5. and let us compare results obtained with different fuzzification methods used assuming Mamdani type of architecture. Results are shown in Fig. 6 to 8 and Fig. 10.

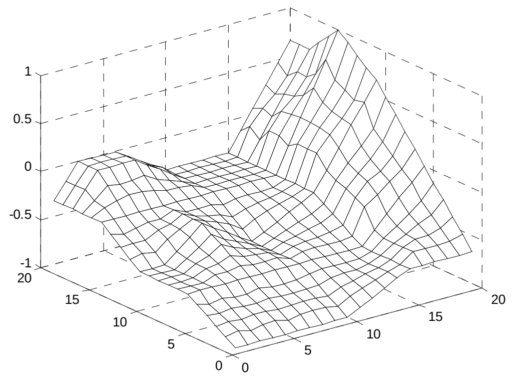


Fig. 6. Results for Mamdani architecture with fuzzifiers with triangular membership functions.

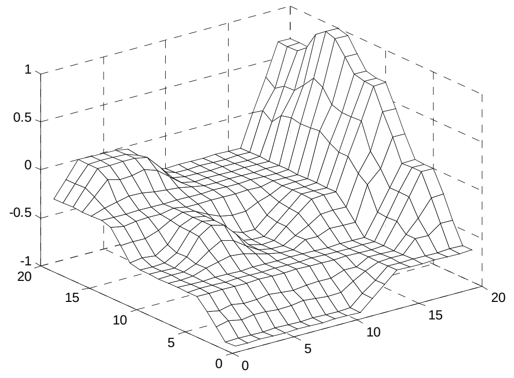


Fig 7. Results for Mamdani architecture with fuzzifiers with trapezoidal membership functions.

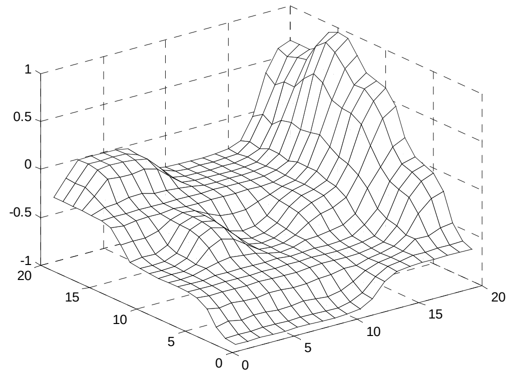


Fig. 8. Results for Mamdani architecture with fuzzifiers with Gaussian membership functions.

2.2. Rule evaluation with fuzzy logic

Once inputs are converted to fuzzy variables these variables 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. 9 shows fuzzy logic operations on zero-one Boolean variables (Fig. 9(a)) and on fuzzy variables (Fig. 9(b)).

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

(a)

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***

(b)

Fig. 9. Comparison of (a) Boolean and (b) Fuzzy logic.

The Mamdani [3] concept (see Fig. 2) 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. In the case of the fuzzy systems AND and OR operators are replaced by MIN and MAX operators respectively.

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. The TSK structure, as shown in Fig. 3, 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.

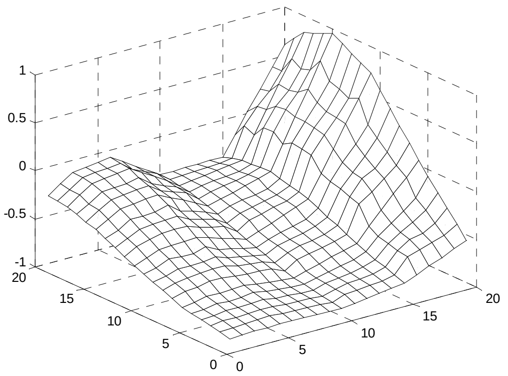


Fig. 10. Results for TSK architecture with fuzzifiers with trapezoidal membership functions.

When fuzzy neural networks are used (Fig.4) 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. 11 shows the surface obtained with product encoding, which is smoother than the surface of Fig. 6(a) which is obtained with a MIN encoding. Exactly the same surface can be obtained with TSK architecture (Fig. 3) when product encoding is used instead of MIN operators.

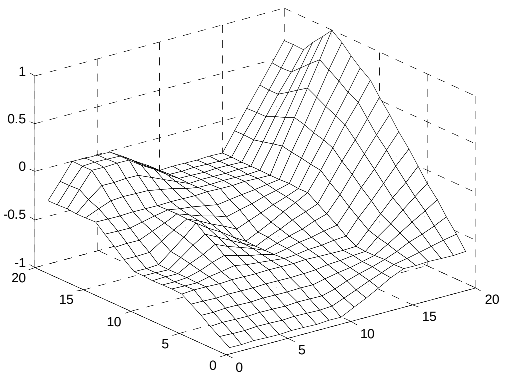


Fig 11. Results for Mamdani architecture with fuzzifiers with triangular membership functions and product encoding instead MIN encoding.

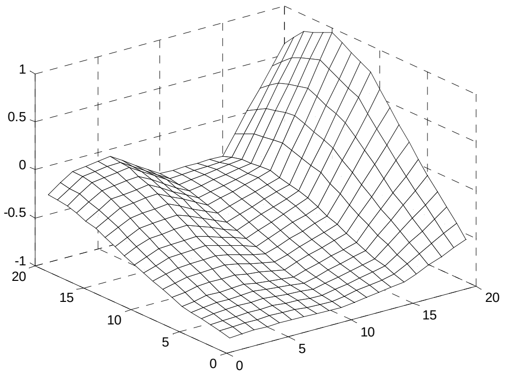


Fig 12. Results for fuzzy neural network architecture of Fig 4.

2.3. Defuzzification

As a result of fuzzy rules of Fig. 2 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.

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. 13. 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. 13 fuzzifiers and MIN operators used for region selection can be replaced by simple neural network architecture. Let us analyze Fig. 14 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. 15. 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 three of them are drawn on Fig. 15.

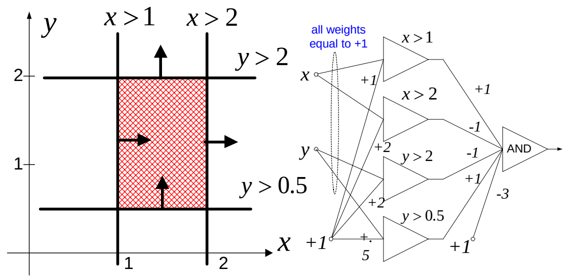


Fig. 13. Separation of the rectangular area on a two dimensional input space and desired neural network to fulfill this task.

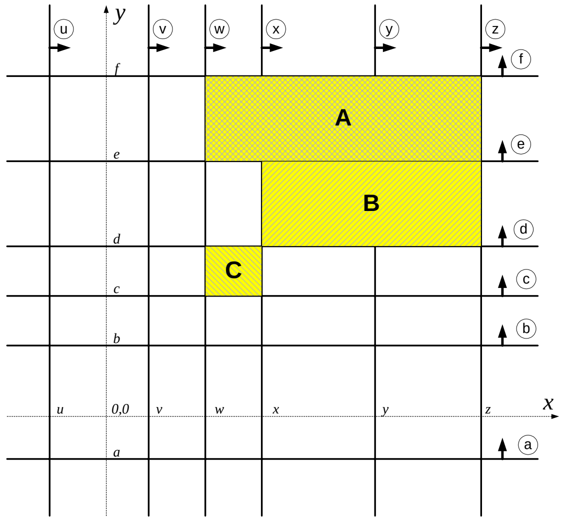


Fig. 14. Two-dimensional input plane separated vertically and horizontally by six neurons in each direction.

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. 3.

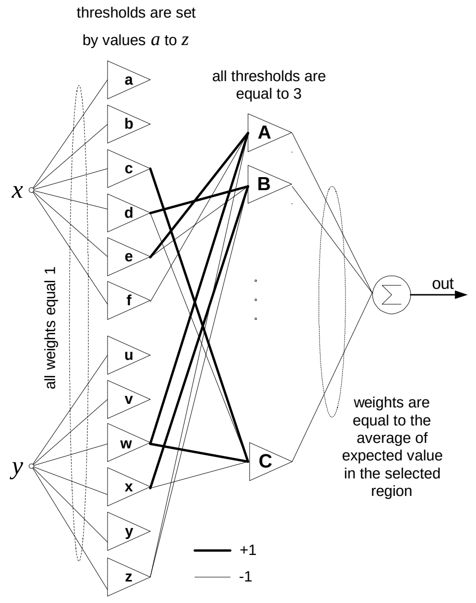

thresholds are set by values a to z
all thresholds are equal to 3
all weights equal 1
x
y
a
b
c
d
e
f
u
v
w
x
y
z
A
B
C
out
weights are equal to the average of expected value in the selected region
+1
-1

Fig. 15. Simple neural networks performing the function of TSK fuzzy system.

It was shown above that a simple neural network of Fig. 15 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.

The most commonly used learning algorithm such as EBP – Error Back Propagation or LM - Levenberg-Marquardt, were developed for layer by layer type feedforward neural networks. Note that feedforward networks can become much more powerful if weight connections across layers are also allowed. Unfortunately only very few software packages are capable of training fully connected neural networks.

Since it is not easy to train neural networks, several special neural networks were developed where no training or limited training is only required. In the following section these special neural network architectures are shortly described. More detailed information, with network diagrams, can be found at [2].

3.1. 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 presented by Pao [7] using the functional link network, where additional inputs to one layer networks are created by arbitrary selection of nonlinear terms. If these nonlinear terms are generated using a polynomial function then this network is known as a polynomial network. These networks are easy to train, but it is usually not known which nonlinear terms are best suited for specific problems. Polynomial networks have a more generalized approach, but with an increase in the dimensionality of the problem the number of polynomial terms grow exponentially and these networks become impractical.

3.2. Counterpropagation networks

Counterpropagation networks were originally proposed by Hecht-Nilsen [8]. This architecture requires several hidden neurons which are equal to the number of input patterns.
When binary input patterns are considered, then the input weights can 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. In the case of binary or normalized patterns the weights in the input layer should be equal to the input patterns. Weights in the output layer should be equal to the output patterns. A disadvantage of the counterpropagation network is that the number of neurons in the hidden layer must be equal to the number of training patterns and this number is sometimes excessively large.

3.3. 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. 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.

3.4. Cascade correlation architecture

The cascade correlation architecture was proposed by Fahlman and Lebiere [9]. 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 weights are adjusted to maximize the magnitude of the correlation between this neuron output and the residual error signal on the network output. 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.

3.5. RBF - Radial Basis Function networks

The RBF network usually has only one hidden layer with special "neurons". These "neurons" respond only to the input signals close to the stored pattern characteristic for each neuron. 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 but the Euclidean distance between the input and stored pattern is computed.

3.6. Sarajedini and Hecht-Nielsen network

The Sarajedini and Hecht-Nielsen [10] network of Fig. 16 is capable of calculating Euclidean distances between input pattern and stored pattern using only a neuron with linear activation function and information about the square of the input vector length. The network is using the following analytical formulas:

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

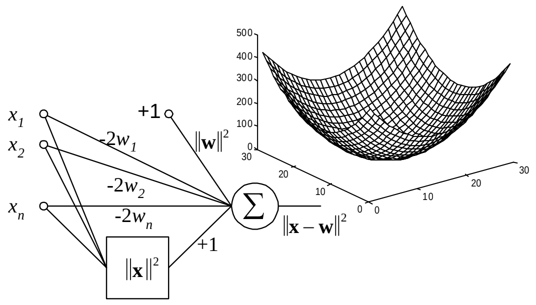


Fig. 16. Sarajedini and Hecht-Nielsen neural network and obtained surface for 2-dim case.

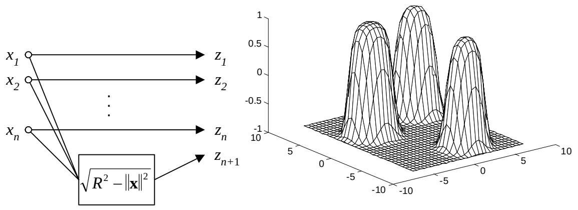


Fig. 17. Network with increased input dimensionality by input pattern transformation and an example of three cluster separation.

3.7. 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 [11][12]. One way to generate this additional input is to use the formula:

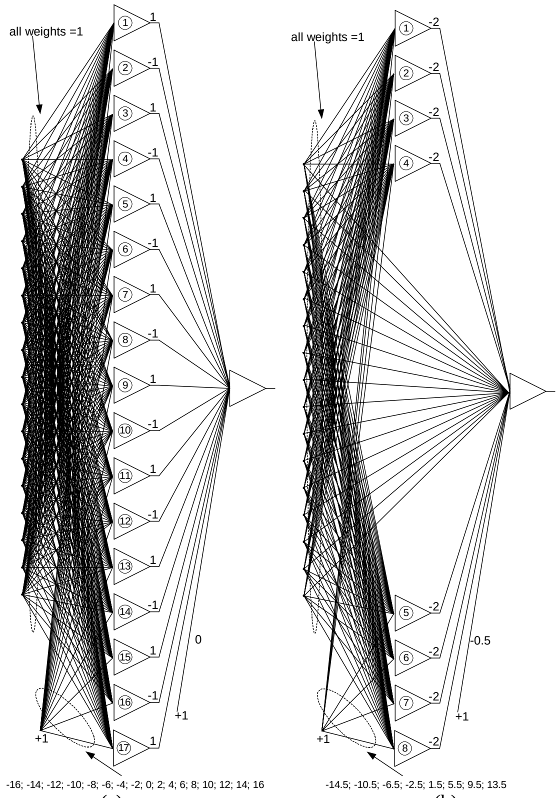
$$z_{n+1} = \sqrt{R^2 - \|\mathbf{x}\|^2} \tag{3}$$

This way all input patterns are projected on a hyper sphere with a radius R and round clusters could be separated by hyper planes (traditional sigmoidal neurons).

Fig. 21 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 hypersphere.

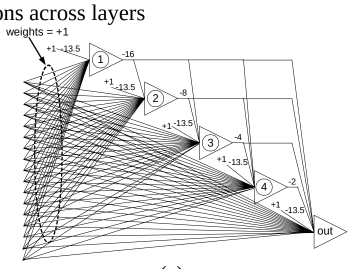
3.8. Comparison of neural network architectures

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 [13]. Let us compare different neural network architectures to solve the parity-17 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 18 neurons required and 17*18+18=324 weights. See Fig. 18(a) for the network architecture.

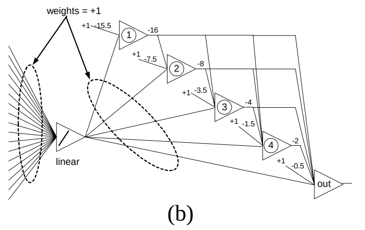


(a) (b)

Fig.18. Parity-17 problem with feedforward bipolar neural network with one hidden layer (a) without and (b) with connections across layers



(a)



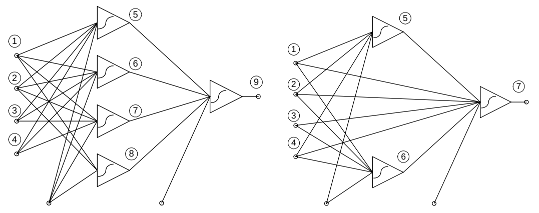
(b)

Fig. 19. Bipolar implementation of a fully connected cascade neural network for the parity-17 problem: (a) five neuron architecture and (b) six neuron architecture

In the case of when connections across layers are allowed, as shown in Fig. 28 (b), the number of neurons in the hidden layer can be reduced from 17 to 8 and the total number of weights is 8*18+18+8=170. When neurons are connected in a cascade, as shown in Fig. 19(a), then only 5 neurons are required and the total number of weights is 5*18+1+2+3+4+5=105.

Note, that in the case of the parity problem (due to the symmetry of inputs) each of networks shown in Fig. 18 and Fig. 19(a) can be further simplified by adding an additional neuron with linear activation function (summator) to the front. This way, the network of Fig. 19(a) can be simplified to the architecture shown in Fig. 19(b) with 6 neurons and 17+5=1+2+3+4+5=37 weights.

One may conclude that the cascade network (Fig. 19) 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. 18(b)) has a slightly larger number of neurons than cascade architecture (Fig. 19(b)) but it is easier to train and in most cases this would be the preferred choice.



(a) (b)

Fig. 20. Feedforward neural networks (a) layered and (b) fully connected.

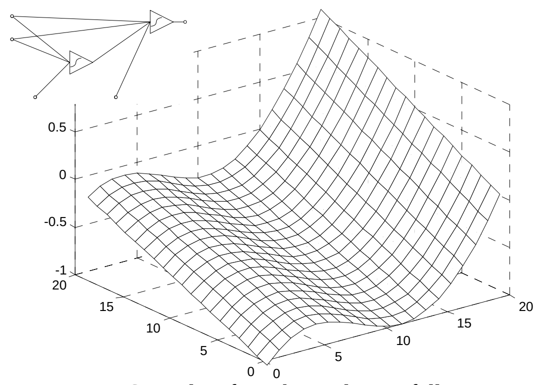


Fig.21. Control surface obtained using fully connected neural network with one hidden neuron.

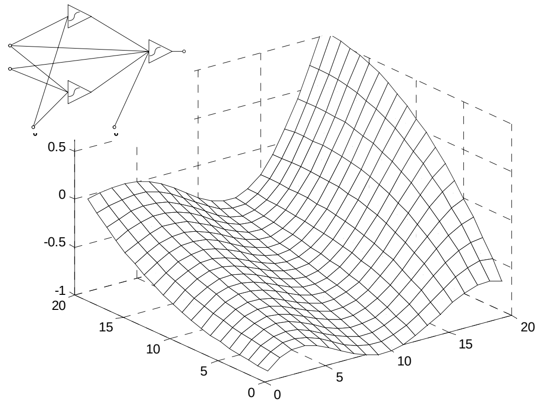


Fig.22. Control surface obtained using fully connected neural network with two hidden neurons.

3.9. 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) [14], which can handle fully connected architectures, but the LM - Levenberg-Marquardt [15] 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 hundreds or thousands times more iterations to reach a solution. An additional advantage of the LM algorithm is its fast convergence to the solution, while the EBP reaches the solution only asymptotically.

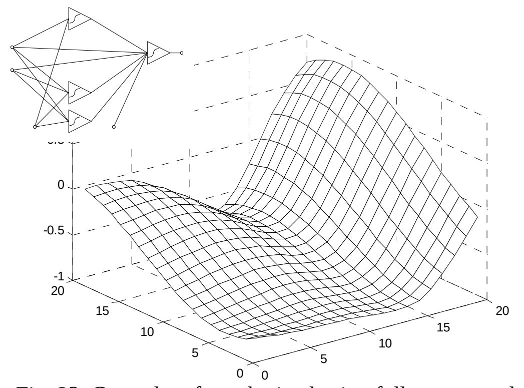


Fig. 23. Control surface obtained using fully connected neural network with three hidden neurons.

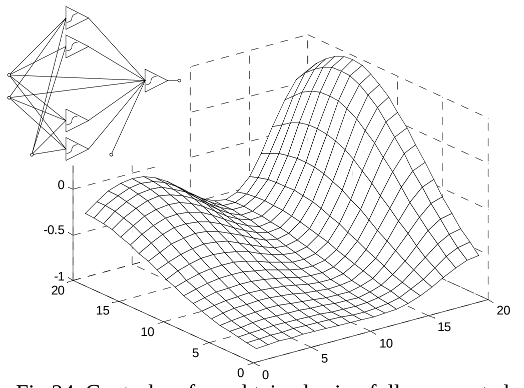


Fig.24. Control surface obtained using fully connected neural network with four hidden neurons.

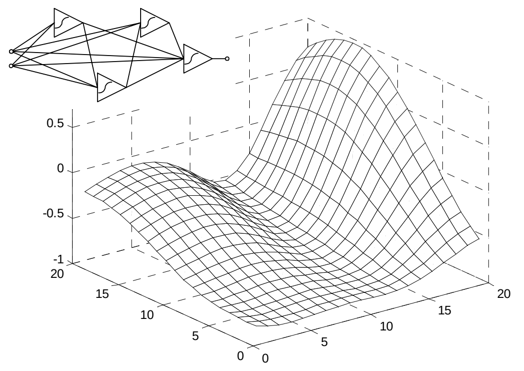


Fig.25. Control surface obtained using neural network with four neurons connected in cascade.

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 [16]. 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 20(a) 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. 20(b) 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. Figures 21 to 25 show results obtained with different neural network architectures trained to the same required function, which was used for fuzzy systems (Fig. 5). Note that when neural network approach is used then better results are obtained than in the case of fuzzy systems. Neural networks require simple hardware and work faster. In [17] a practical comparison of various neural and fuzzy architectures, implemented on the HC11 Motorola microcontroller, were presented. Again neural systems had shorter and faster assembly codes.

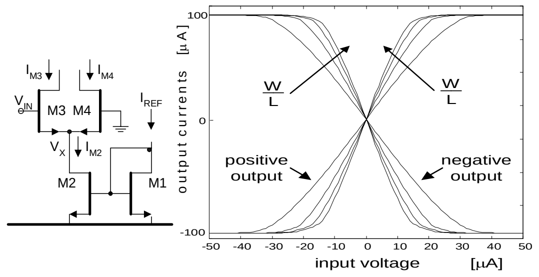
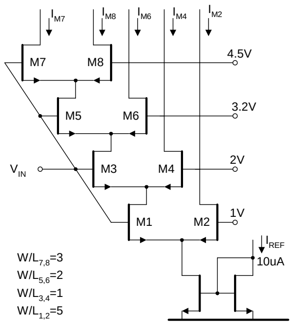
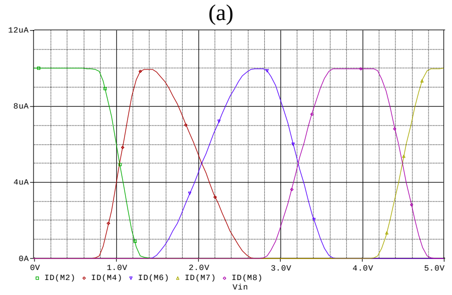


Fig. 26. Simple VLSI implementation of neuron with a differential pair: (a) circuit diagram (b) result of SPICE simulation.



(a)



(b)

Fig. 27. VLSI implementation of fuzzifier block: (a)

circuit diagram, (b) result of SPICE simulations with five membership functions plotted.

VLSI implementations

In the case of neural networks, a sigmoidal type of activation function can be implemented using a simple differential pair (Fig 26). 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. 27(a), may act as fuzzifiers. Fig. 27(b) shows membership functions implemented by the circuit of Fig. 27(a). Fig. 28 shows a simple solution of digitally programmed 6-bit weights. More detailed review of VLSI implementation of neural and fuzzy systems are in [18].

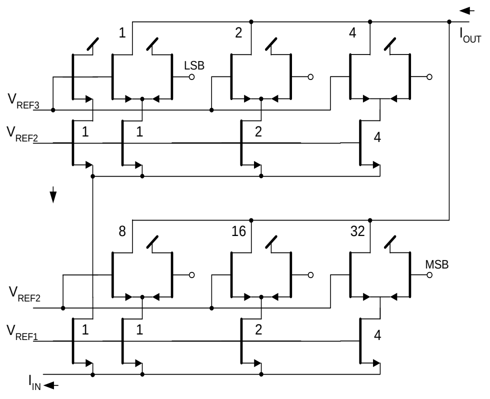


Fig. 28. Programmable current multipliplier by digital weights.

4. 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. 10) required 6+6+36= 48 values to be stored. In the case of fully connected neural network with three hidden neurons (Fig. 23), 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 have to be used and optimization is not desired, neural networks can also replace fuzzy systems as was shown in Fig. 15.

REFERENCES

[1] W. Duch, J. Korbicz, L. Rutkowski, R. Tadeusiewicz Sieci Neuronowe Akademicka Oficyna Wydawnicza EXIT, Warszawa 2000.

[2] B. M.Wilamowski, "Neural Networks and Fuzzy Systems", chapter 32 in Mechatronics Handbook edited by Robert R. Bishop, CRC Press, pp. 33-1 to 32-26, 2002.

[3] E. H. Mamdani, "Application of Fuzzy Algorithms for Control of Simple Dynamic Plant," IEEE Proceedings, Vol. 121, No. 12, pp. 1585-1588, 1974.

[4] Sugeno and G. T. Kang, "Structure Identification of Fuzzy Model," Fuzzy Sets and Systems, Vol. 28, No. 1, pp. 15-33, 1988.

[5] T. Takagi and M. Sugeno, "Fuzzy Identification of Systems and Its Application to Modeling and Control," IEEE Transactions on System, Man, Cybernetics, Vol. 15, No. 1, pp. 116-132, 1985.

[6] D. Rutkowska, Y. Hayashi "Neuro-fuzzy systems approaches" *Int. J. Advanced Computational Intelligence*, vol. 3, no 3, pp. 177-185.

[7] Y. H. Pao, Adaptive Pattern Recognition and Neural Networks, Reading, Mass. Addison-Wesley Publishing Co. 1989

[8] Hecht-Nielsen, R. 1987. "Counterpropagation networks" Appl. Opt. 26(23):4979-4984.

[9] S.E Fahlman,.and C. Lebiere, "The cascade-correlation learning architecture" nn D. S.Touretzky, Ed. Advances in Neural Information Processing Systems 2, Morgan Kaufmann, San Mateo, Calif., (1990),524-532.

[10] A. Sarajedini, R. Hecht-Nielson, "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 pp. 905-910 vol.3

[11] Y. Ota and B. M. Wilamowski, "Input Data Transformation for Better Pattern Classification with Fewer Neurons," proceedings of *Word Congress on Neural Networks*, San Diego, California, USA, vol. 3, pp. 667-672, June 4-9, 1994.

[12] B. M. Wilamowski, and R. C. Jaeger, "Implementation of RBF Type Networks by MLP Networks," *IEEE International Conference on Neural Networks*, Washington, DC, June 3-6, 1996, pp. 1670-1675.

[13] B. Wilamowski, D. Hunter, "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.

[14] University of Tübingen. [Online]. Available: http://www-ra.informatik.uni-tuebingen.de/SNNS/

[15] 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.

[16] Auburn University [Online]. Available: http://www.eng.auburn.edu/users/wilambm/BMW.zip.

[17] B.M. Wilamowski and J. Binfet, "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.

[18] B. M. Wilamowski, J.Y. Hung, and R. Gottiparthy "Digitally Tuned Analog VLSI Controllers" ISIE'05 IEEE International Symposium on Industrial Electronics, Dubrovnik Croatia, June 19-22, 2005