Neural Networks and Fuzzy Systems for Nonlinear Applications

Bogdan M. Wilamowski
Director of Alabama Microelectronics Science and Technology Center
Auburn University, Auburn AL 36849, USA
wilam@ieee.org

Abstract - **Nonlinear processes are difficult to control because there can be so many variations of the nonlinear behavior. 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 is not trivial.**

Advantages and disadvantages of fuzzy systems will be presented and compared, including Mamdani, Takagi-Sugeno and other approaches. In the conclusion, advantages and disadvantages of neural and fuzzy approaches are discussed with a reference to their hardware implementation.

I. INTRODUCTION

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 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. However, developments of neural or fuzzy systems are not trivial.

Neural networks are capable of approximating any multidimensional nonlinear functions and as such they can be very useful in nonlinear control [1][2]. But, until now there was no precise method for determination of proper neural network architecture. Currently this is more art than science. Also, all known algorithms often have difficulties to converge to a solution and this leads to a lot of disappointments and frustrations. As a result many researches were discouraged with neural networks.

In the case of fuzzy systems the system topology and parameters are easier to define, but also in this case this is more art than science. Fuzzy systems have very significant limitations of the number of inputs.

This presentation will describe the current status of neural networks and fuzzy systems. Comparison of various learning algorithms for neural networks will be presented and specific architectures, which are easy to train, will be described. Some of these NN architectures need not to be trained at all. In the case of fuzzy systems two practical approaches, Mamdani [3] and TSK [4][5], will be described. Some more advanced fuzzy systems will be also presented. Special fuzzy systems suitable for training will be also shown. We will try to bridge two words: neural networks which can be designed (need not to be trained) and fuzzy systems, which are easy to train.

II. NEURAL NETWORKS

Fig. 1 shows several simple McCulloch and Pitts [1943] neurons. These neurons are more powerful than logic gates. Each neuron can implement many logic functions and what is even more important is that without changing of network topology different logic function can be realized by adjusting weights and thresholds. For over a half of century we were recognizing a tremendous power of neural networks but we were not able to use them efficiently.

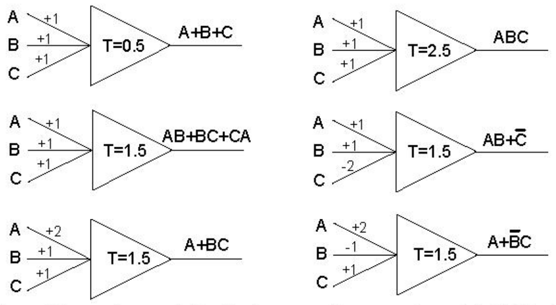

A B C +1 +1 +1 T=0.5 A+B+C
A B C +1 +1 +1 T=2.5 ABC
A B C +1 +1 +1 T=1.5 AB+BC+CA
A B C +1 +1 -2 T=1.5 AB+C̄
A B C +2 +1 +1 T=1.5 A+BC
A B C +2 -1 +1 T=1.5 A+B̄C

Fig. 1. Examples of logical operations using McCulloch-Pitts neurons

A significant breakthrough occurred in 1978 when Rumelhart [6] proposed the Error Backpropagation algorithm. The algorithm itself was not that important as the fact that instead a hard threshold functions and the soft sigmoid functions were used. This way neural networks became "transparent" for error signals and we could train multilayered neural networks. Unfortunately the learning algorithms now often are not converging to solutions. In order to undusted problem let us analyze some simple neural networks.

A single neuron can divide only linearly separated patterns. In order to select a shape of Fig. 2 (in two dimensions) four neurons have to be used. Fig. 2 shows the separation lines and equation for all four neurons. Fig. 3. shows neural network architectures with all network weights and sigmoidal activation functions with various gains.

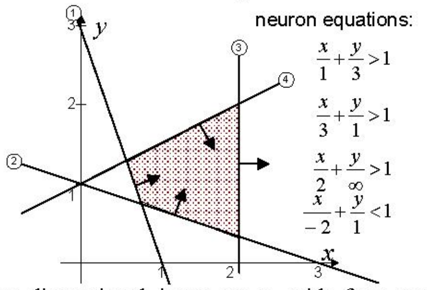

neuron equations:
$\frac{x}{1}+\frac{y}{3}>1$
$\frac{x}{3}+\frac{y}{1}>1$
$\frac{x}{2}+\frac{y}{\infty}>1$
$\frac{x}{-2}+\frac{y}{1}<1$
1 2 3 4
x y

Fig. 2. Two-dimensional input space with four separation lines representing four neurons.

1-4244-1148-3/07/$25.00 ©2007 IEEE.

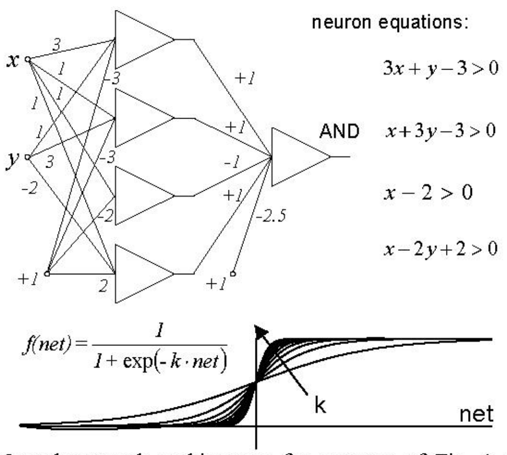


Fig. 3. Neural network architecture for patterns of Fig. 1 and used activation functions.

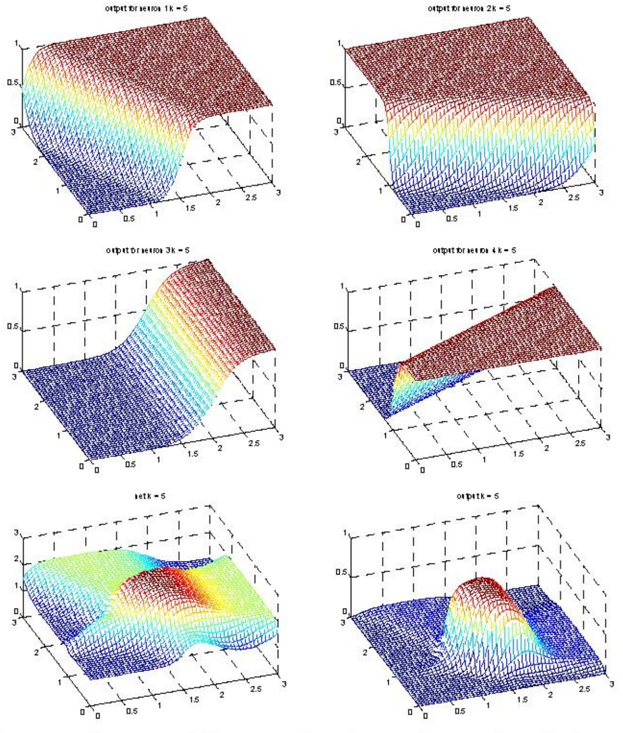
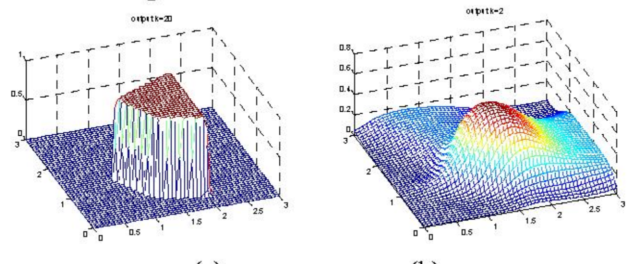


Fig. 4. Surfaces on different nodes of neural networks of Fig. 3 for neurons with the gain of activation function k=5.



(a) (b)

Fig. 5. Nonlinear mapping of the neural network of Fig. 3 for different values of the neuron gains (a) k=20, (b) k=2

Neural network may separate patterns (perform classifications) as shown in Fig. 5 (a) but also they can produce very complex nonlinear shapes (see Fig. 5 (b)) The question is how to design neural networks for arbitrary nonlinear mapping? Can we just select randomly chosen neural network architectures and train them?

III. NEURAL NETWORK LEARNING

Several methods to training a single neuron (read single layer of feed forward neural network) were developed such as:

- correlation learning rule $\Delta w_{ij} = c x_i d_j$
- instar learning rule $\Delta w_i = c(x_i - w_i)$
- outstar learning rule $\Delta w_{ij} = c(d_j - w_{ij})$
- perceptron fixed rule: $\Delta w_{ij} = c x_i \left(d_j - o_j\right)$
- perceptron adjustable rule: $\Delta w_{ij} = c \frac{net}{\|\mathbf{x}\|^2} x_i \left(d_j - o_j\right)$
- LMS (Widrow-Hoff) rule: $\Delta w_{ij} = c x_i \left(d_j - net_j\right)$
- delta rule: $\Delta w_{ij} = c x_i f_j' \left(d_j - o_j\right)$
- linear regression (pseudoinverse) rule : $\mathbf{w} = \left(\mathbf{x}^T \mathbf{x}\right)^{-1} \mathbf{x}^T \mathbf{d}$

where:
w_{ij} =weight from ith to jth neuron
Δw_{ij} =weight change for w_{ij}
c =learning constant
x_i =signal on the ith input

$$net_j = \sum_{i-1}^{n} x_{ij} w_{ij} + w_{i+1,j} \tag{1}$$

$o_j = f\left(net_j\right)$ output signal

In order to understand neural network learning let us consider a neuron with eight inputs with binary signals and weights such as
$\mathbf{x}$ = +1 -1 -1 +1 -1 +1 +1 -1
if weights $\mathbf{w} = \mathbf{x}$
$\mathbf{w}$ = +1 -1 -1 +1 -1 +1 +1 -1
then

$$net = \sum_{i=1}^{n} w_i x_i = 8 \tag{2}$$

and this is the maximum value $net=8$ can have for any other combinations net would be smaller. For example for the same weights and slightly different input pattern:
$\mathbf{x}$ = +1 +1 -1 +1 -1 +1 +1 -1

$$net = \sum_{i=1}^{n} w_i x_i = 6 \tag{3}$$

One may notice that the maximum net value is when weights are the same as the pattern (assuming that both weights and patterns are normalized). Therefore, one can draw a conclusion that during the learning process weights should be changed the same way as input signals to these weights

$$\Delta w_{ji} = c\, \Delta_j\, x_{ji} \tag{4}$$

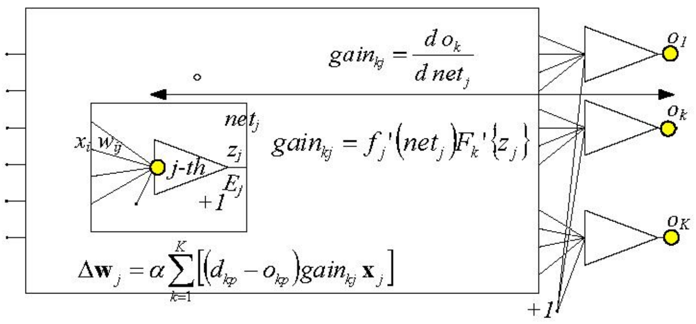
In the case of the delta rule for one neuron

$$\Delta_j = f_j'\left(d_j - o_j\right) \tag{5}$$

A. Error Back Propagation EBP

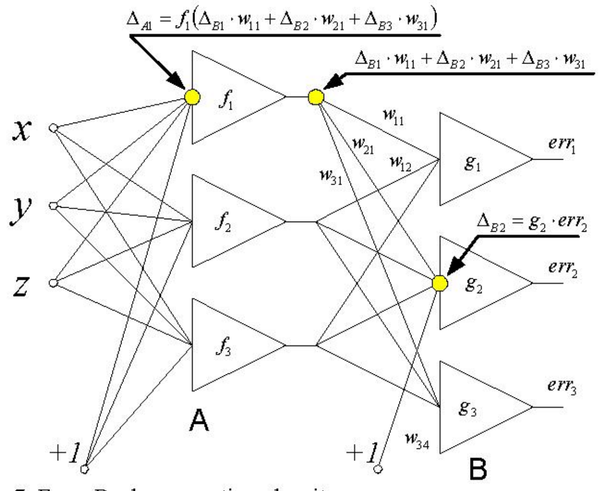
This rule can be extended for multiple layers as it is shown in Fig. 6. Instead of a gain f' through single neuron the signal gain through entire network can be used and this leads to the

Error Backpropagation (EBP) algorithm, as it is illustrated in Fig. 7.



$gain_{kj} = \frac{d\,o_k}{d\,net_j}$
$gain_{kj} = f_j'(net_j)F_k'\{z_j\}$
$\Delta \mathbf{w}_j = \alpha \sum_{k=1}^{K}\left[(d_{kp} - o_{kp})gain_{kj}\,\mathbf{x}_j\right]$
x_i
w_{ij}
net_j
z_j
E_j
j-th
+1
o_1
o_k
o_K
+1

Fig. 6. Extension of delta rule for a feed forward network with multiple layers.



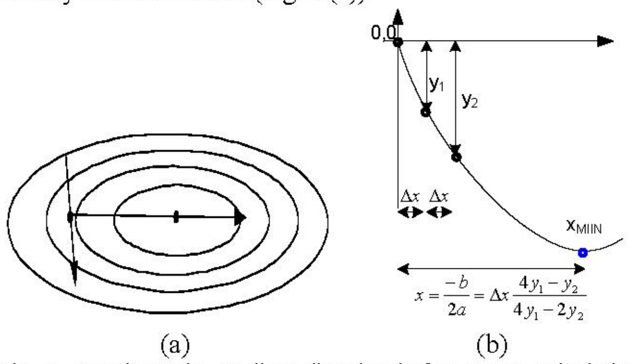
$\Delta_{A1} = f_1(\Delta_{B1}\cdot w_{11} + \Delta_{B2}\cdot w_{21} + \Delta_{B3}\cdot w_{31})$
$\Delta_{B1}\cdot w_{11} + \Delta_{B2}\cdot w_{21} + \Delta_{B3}\cdot w_{31}$
$\Delta_{B2} = g_2 \cdot err_2$
x
y
z
f_1
f_2
f_3
g_1
g_2
g_3
w_{11}
w_{21}
w_{12}
w_{31}
w_{34}
err_1
err_2
err_3
A
B
+1
+1

Fig. 7. Error Backpropagation algoritm.

Many other learning algorithms which are derivatives of EBP algorithm were developed. They are shortly described in following sections

B. Gradient direction search

The EBP algorithm can be significantly speeded up, when after finding components of the gradient, weights are modified along the gradient direction until a minimum is reached. One method to find a minimum along the gradient direction is the tree step process of finding error for three points along gradient direction and then, using a parabola approximation, jump directly to the minimum (Fig. 8 (b)).



0,0
y_1
y_2
Δx
Δx
x_{MIIN}
$x = \frac{-b}{2a} = \Delta x \frac{4y_1 - y_2}{4y_1 - 2y_2}$
(a)
(b)

Fig. 8. Search on the gradient direction before a new calculation of gradient components.

C. Quickprop algorithm by Fahlman

The fast learning algorithm was proposed by Fahlman [7] and it is known as the *quickprop.*

$$\Delta w_{ij}(t) = -\alpha S_{ij}(t) + \gamma_{ij}\Delta w_{ij}(t-1) \qquad (6)$$

$$S_{ij}(t) = \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} + \eta w_{ij}(t) \qquad (7)$$

α learning constant
γ memory constant (small 0.0001 range) leads to reduction of weights and limits growth of weights
η momentum term selected individually for each weight

D. RPROP Resilient Error Back Propagation

RPROP [8] is very similar to EBP, but weights adjusted without using values of the propagated errors, but only its sign. Learning constants are selected individually to each weight based on the history

$$\Delta w_{ij}(t) = -\alpha_{ij}\,\mathrm{sgn}\left(\frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}(t)}\right) \qquad (8)$$

$$S_{ij}(t) = \frac{\partial E(\mathbf{w}(t))}{\partial w_{ij}} + \eta w_{ij}(t) \qquad (9)$$

$$\alpha_{ij}(t) = \begin{cases} \min(a\cdot\alpha_{ij}(t-1), \alpha_{\max}) & \text{for} \quad S_{ij}(t)\cdot S_{ij}(t-1) > 0 \\ \max(b\cdot\alpha_{ij}(t-1), \alpha_{\min}) & \text{for} \quad S_{ij}(t)\cdot S_{ij}(t-1) < 0 \\ \alpha_{ij}(t-1) & \text{otherwise} \end{cases}$$

E. Back Percolation

Error is propagated as in EBP and then each neuron is "trained" using an algorithm to train one neuron such as pseudo inversion. Unfortunately pseudo inversion may lead to errors, which are sometimes larger than 2 for bipolar or larger than 1 for unipolar

F. Delta-bar-Delta

For each weight the learning coefficient is selected individually. It was developed for quadratic error functions

$$\Delta\alpha_{ij}(t) = \begin{cases} a & \text{for} \quad S_{ij}(t-1)D_{ij}(t) > 0 \\ -b\cdot\alpha_{ij}(t-1) & \text{for} \quad S_{ij}(t-1)D_{ij}(t) < 0 \\ 0 & \text{otherwise} \end{cases} \qquad (10)$$

$$D_{ij}(t) = \frac{\partial E(t)}{\partial w_{ij}(t)} \qquad (11)$$

$$S_{ij}(t) = (1-\xi)D_{ij}(t) + \xi S_{ij}(t-1) \qquad (12)$$

G. Levenberg-Marquardt Algorithm (LM)

Newton method is the principal second order method

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \mathbf{A}_k^{-1}\mathbf{g} \qquad (13)$$

where $\mathbf{A}_k$ is Hessian and **g** is the gradient

$$\mathbf{A} = \begin{matrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_2 \partial w_1} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_1} \\ \frac{\partial^2 E}{\partial w_1 \partial w_2} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_n \partial w_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E}{\partial w_1 \partial w_n} & \frac{\partial^2 E}{\partial w_2 \partial w_n} & \cdots & \frac{\partial^2 E}{\partial w_n^2} \end{matrix} \qquad \mathbf{g} = \begin{matrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \\ \vdots \\ \frac{\partial E}{\partial w_n} \end{matrix} \quad (14)$$

The Newton algorithm works very well only for quasi linear system. In nonlinear systems it often fails to converge. Also, Computation of the Hessian is relatively expensive. Levenberg and Marquardt made several improvements for the Newton method. Instead of computing Hessian they are computing a Jacobian

$$\mathbf{J} = \begin{bmatrix} \frac{\partial e_{11}}{\partial w_1} & \frac{\partial e_{11}}{\partial w_2} & \cdots & \frac{\partial e_{11}}{\partial w_n} \\ \frac{\partial e_{21}}{\partial w_1} & \frac{\partial e_{21}}{\partial w_2} & \cdots & \frac{\partial e_{21}}{\partial w_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{M1}}{\partial w_1} & \frac{\partial e_{M1}}{\partial w_2} & \cdots & \frac{\partial e_{M1}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{1P}}{\partial w_1} & \frac{\partial e_{1P}}{\partial w_2} & \cdots & \frac{\partial e_{1P}}{\partial w_N} \\ \frac{\partial e_{2P}}{\partial w_1} & \frac{\partial e_{2P}}{\partial w_2} & \cdots & \frac{\partial e_{2P}}{\partial w_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial e_{MP}}{\partial w_1} & \frac{\partial e_{MP}}{\partial w_2} & \cdots & \frac{\partial e_{MP}}{\partial w_N} \end{bmatrix} \quad (15)$$

and then Hessians and gradients are calculated using:

$$\mathbf{A} = 2\mathbf{J}^T\mathbf{J} \quad (16)$$

and

$$\mathbf{g} = 2\mathbf{J}^T\mathbf{e} \quad (17)$$

Resulting in the modified Newton method [9][10]:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T\mathbf{J}_k\right)^{-1}\mathbf{J}_k^T\mathbf{e} \quad (18)$$

LM algorithm combines the speed of the Newton algorithm with the stability of the steepest decent method. The LM algorithm uses the following formula to calculate weights in subsequent iterations:

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \left(\mathbf{J}_k^T\mathbf{J}_k + \mu\mathbf{I}\right)^{-1}\mathbf{J}_k^T\mathbf{e} \quad (19)$$

For μ = 0 it becomes the Gauss-Newton method. For very large μ the LM algorithm becomes the steepest descent or the EBP algorithm. The *m* parameter is automatically adjusted at every iteration in order to secure convergence.

While EBP usually requires several thousands iterations the LM algorithm converges in several iterations. The LM algorithm requires computation of the Jacobian **J** matrix at each iteration step and the inversion of $\mathbf{J}^T\mathbf{J}$ square matrix. In the LM algorithm an *N* by *N* matrix must be inverted in every iteration. Therefore LM algorithm is not very efficient to train large neural networks.

IV. SPECIAL NEURAL NETWORK ARCHITECTURES FOR EASY LEARNING

Because difficulties with neural network training several special neural network architectures were developed which are easy to train. These special neural network architectures will be reviewed in following sections:

A. Functional Link 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 presented by Nilsson [11] and then was elaborated by Pao [12] using the functional link network are shown in Fig. 9. Using nonlinear terms with initially determined functions, the actual number of inputs supplied to the one-layer neural network is increased.

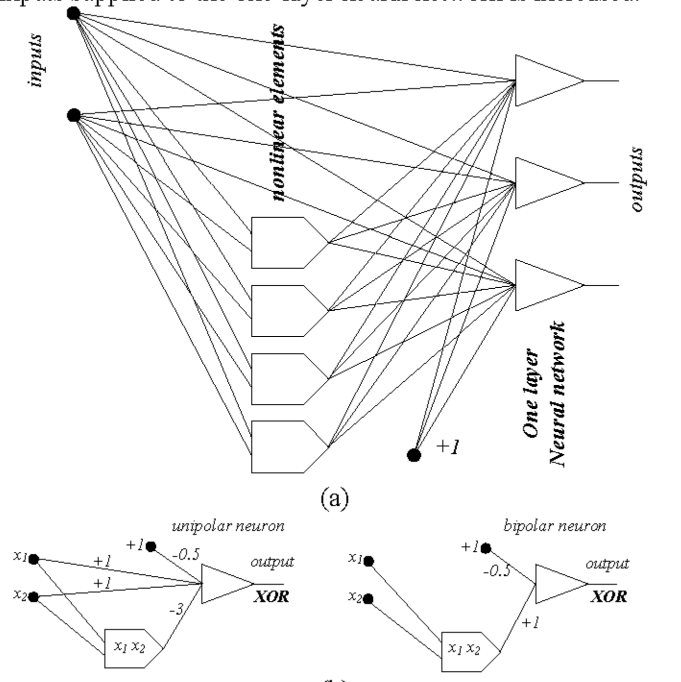


Fig. 9. Functional link networks (a) network architecture (b) used for solution of the *XOR* problem

The problem with the functional link network is that proper selection of nonlinear elements is not an easy task. In many practical cases, however, it is not difficult to predict what kind of transformation of input data may linearize the problem, and so the functional link approach can be used.

B. Polynomial Networks

Polynomial networks are a subclass of functional link networks, where for nonlinear functions polynomial terms are used.

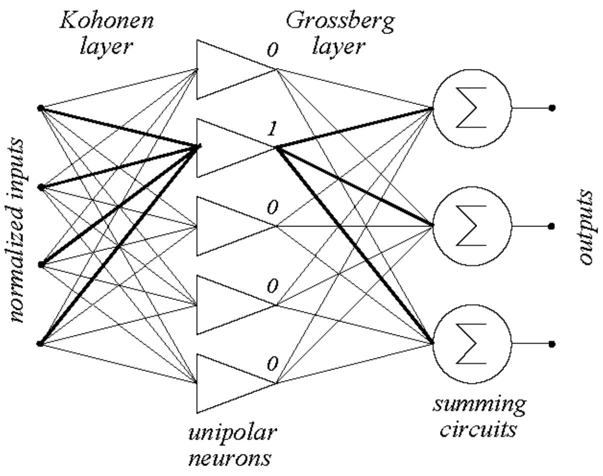
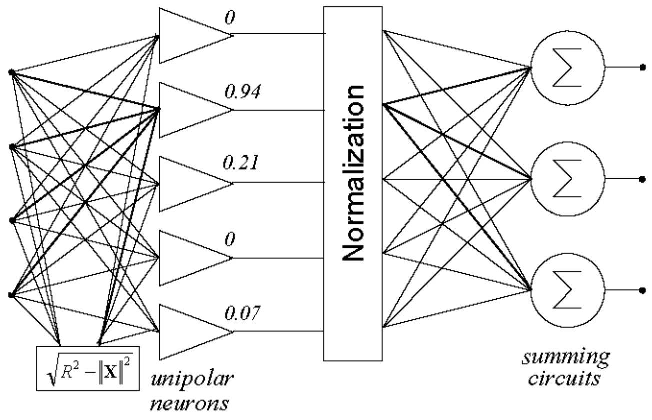


Fig. 10 Counterpropagation network

C. Counterpropagation networks

The counterpropagation network was originally proposed by Hecht-Nilsen [13] This network, which is shown in Fig. 10, requires numbers of hidden neurons equal to the number of input patterns, or more exactly, to the number of input clusters. The first layer is known as the Kohonen layer with unipolar neurons. In this layer only one neuron, the winner, can be active. The second is the Grossberg outstar layer.

A modification [14] of counterpropagation network is shown in Fig. 11. This modified network is can be easily designed. Weights in the input layer are equal to input patterns and weights in the output layer are equal to output patterns.



(a)

Inputs				Outputs		
same as input weights				*same as output weights*		
-2.3	3.12	-1	6.8	1.3	-1	4.1
0.2	-5.2	4.6	2.5	5.4	-4.5	0.3
5.23	-2.1	9.32	-1.32	-3.1	2.1	1.92
7.5	-0.34	-3.5	5.43	-2.4	4.87	-8.3
1.23	4.32	-8.1	-0.72	10.1	-2.1	-3.2

(b)

Fig. 11 Modification of the counter propagation network (network architecture, (b) sample weights, which are equal to input and output patterns.

D. Radial Basis Function Networks

The structure of the radial basis network is shown in Fig. 12 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 formula

$$h_i = \exp\left(-\frac{\|ix - is_i\|^2}{2\sigma^2}\right) \tag{20}$$

where:
ix =input vector
is_i =stored pattern representing the center of the i cluster
σ_i =radius of the cluster

The radial-based networks can be designed or trained. Training is usually carried out in two steps. In the first step, the hidden layer is usually trained in the unsupervised mode by choosing the best patterns for cluster representation. An approach, similar to that used in the WTA architecture can be used. Also in this step, radii σ_i must be found for a proper overlapping of clusters.

The second step of training is the error backpropagation algorithm carried on only for the output layer. Since this is a supervised algorithm for one layer only, the training is very rapid.

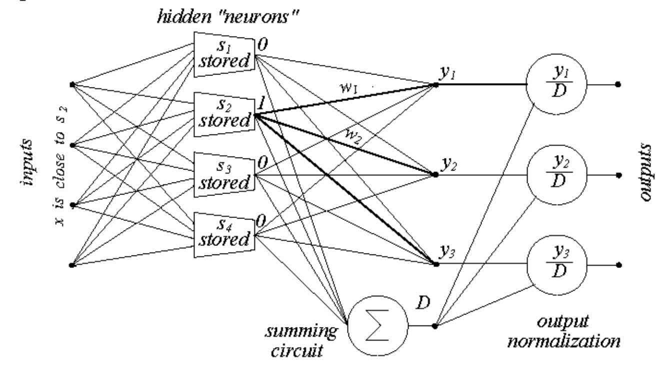


Fig. 12. Radial Basis Function Network

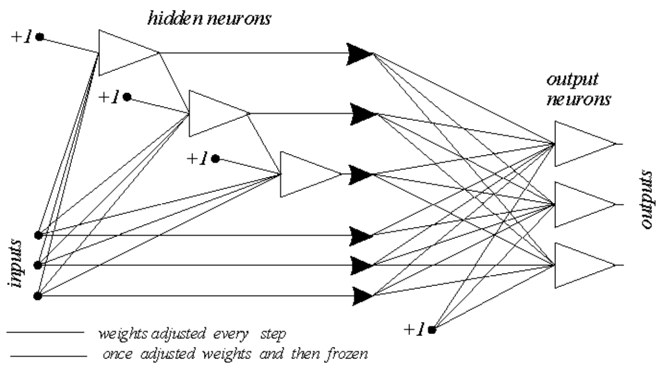

hidden neurons
+1
+1
+1
+1
output neurons
inputs
outputs
weights adjusted every step
once adjusted weights and then frozen

Fig. 13. Cascade correlation network

E. Cascade Correlation Networks

The cascade correlation architecture was proposed by Fahlman and Lebiere [15]. 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 the delta (backpropagation) algorithm. Each hidden neuron is trained just once and then its weights are frozen. The network learning and building process is completed when satisfied results are obtained. The learning process is limited to multiple one neuron (one layer) training and it is very fast.

V. FUZZY SYSTEMS

Fuzzy systems give us another solution of the nonlinear mapping problem. The principle of operation of the fuzzy controller significantly differs from neural networks. Analog values are at first converted into a set of fuzzy variables which are processed by a fuzzy logic and then a new set of fuzzy variables are converted back to analog value. Most commonly used algorithms are Mamdani [3] and Tagagi-Sugeno-Ken [4][5].

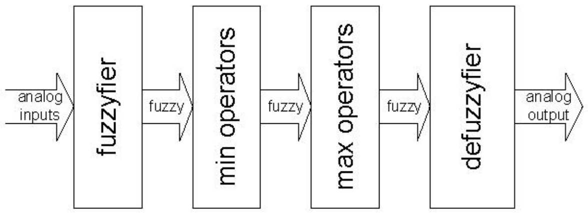


Fig. 14. Architecture of Mamdani fuzzy controller

A. Mamdani fuzzy controller

The block diagram of the Mamdani approach [3] is shown in Fig. 14. 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. 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.

In some implementations, MIN operators are replaced by product operators (signals are multiplied). The rightmost block of the diagram represents defuzzification, where the output analog variable is retrieved from a set of output fuzzy variables. The most common is the centroid type of defuzzification.

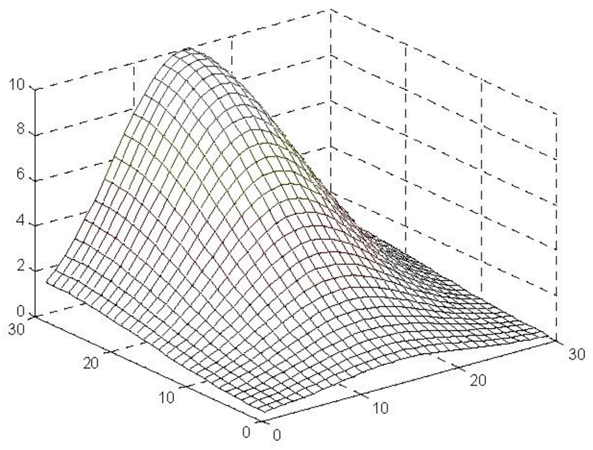


Fig. 15. Required control surface

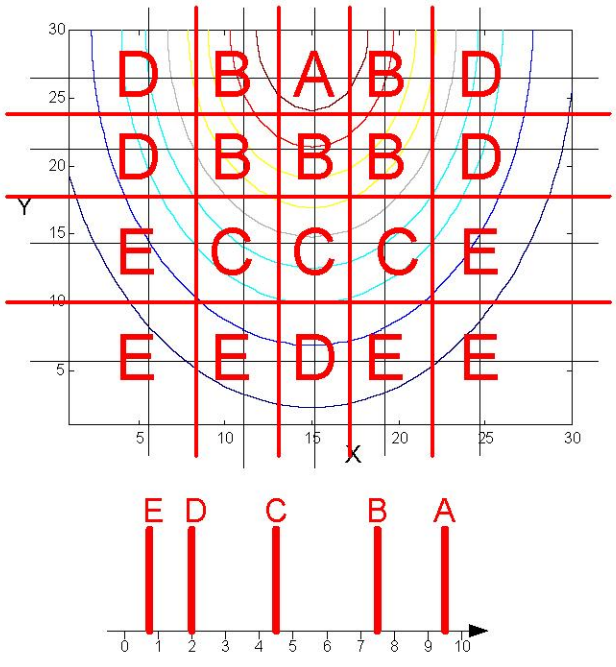


Fig. 16. Design process for Mamdani type fuzzy controller

Let us assume that we have to design the Mamdani Fuzzy controller having the required function shown in Fig. 15. A contour plot of the same surface is shown in Fig. 16. In order to design a fuzzy controller at first characteristic break points should be selected for input variables. In our example we selected 8, 13, 17, and 22 for X variable and 10, 18, and 24 for Y variable. For higher accuracy, more membership functions should be used. A similar process has to be done for the output variable where values 0.8, 2, 4.5, 7.5, and 9.5 were selected. The last step is to assign to every area one specific membership function of the output variable. This is marked on Fig. 16 by letters from A to F. The described above process can be redefined by assigning different values for both inputs and outputs.

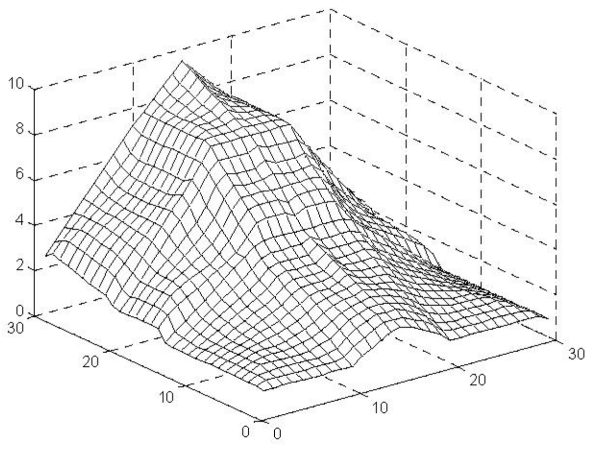


Fig. 17. Control surface obtained with Mamdani Fuzzy controller.

B. TSK fuzzy controller

More recently Mamdani architecture was replaced by TSK [4][5] (Takagi, Sugeno, Kang) architecture where the defuzzification block was replaced with normalization and weighted average The block diagram of TSK approach is shown in Fig. 18 The TSK architecture, as shown in Fig. X (b), 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. The TSK fuzzy system works as a lookup table.

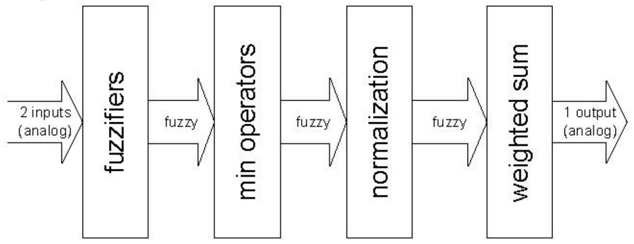


Fig. 18 Architecture of TSK fuzzy controller

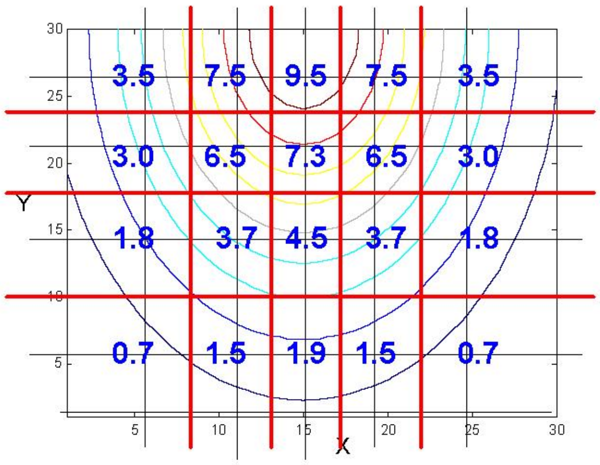


Fig. 19. Design process for TSK type fuzzy controller

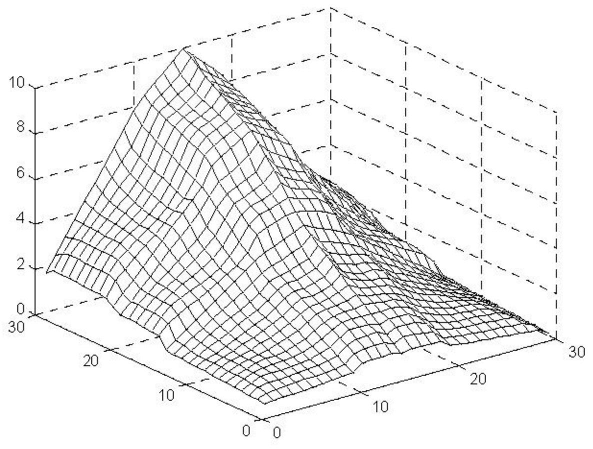


Fig. 20. Control surface obtained with TSK Fuzzy controller.

VI. CONCLUSION

Various learning methods of neural networks including supervised and unsupervised methods are presented and illustrated with examples. General learning rule as a function of the incoming signals is discussed. Other learning rules such as Hebbian learning, perceptron learning, LMS - Least Mean Square learning, delta learning, WTA – Winner Take All learning, and PCA - Principal Component Analysis are presented as a derivation of the general learning rule. The presentation focuses on various practical methods such as Quickprop, RPROP, Back Percolation, Delta-bar-Delta and others. More advance gradient-based methods including pseudo inversion learning, conjugate gradient, Newton and LM - Levenberg-Marquardt Algorithm are illustrated with examples.

Special neural architectures for easy learning such as cascade correlation networks, functional link networks, polynomial networks, counterpropagation networks, and RBF-Radial Basis Function networks are described.

REFERENCES

[1] J.M. Zurada, *Artificial Neural Systems*, PWS Publishing Company, St. Paul, MN, 1995.

[2] Wilamowski, B.M, 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] Rumelhart, D. E., Hinton, G. E. and Wiliams, R. J, "Learning representations by back-propagating errors", *Nature,* vol. **323**, pp. 533-536, 1986.

[7] Scott E. Fahlman. Faster-learning variations on back-propagation: An empirical study. In T. J. Sejnowski G. E. Hinton and D. S. Touretzky, editors, *1988 Connectionist Models Summer School*, San Mateo, CA, 1988. Morgan Kaufmann.

[8] M Riedmiller and H Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)*, 1993.

[9] M.T. Hagan, M.B. Menhaj. "Training feedforward networks with the Marquardt algorithm." *IEEE Transaction onNeural Networks* 1994; NN-5:989–993.

[10] B. M.Wilamowski, S. Iplikci, M. O. Kayank and M. O.Efe, "An Algorithm for Fast Convergence in Training Neural Networks ", *International Joint Conference on Neural Networks (IJCNN'01)* pp. 1778-1782, Washington DC, July 15-19, 2001.

[11] N. J. Nilson, *Learning Machines: Foundations of Trainable Pattern Classifiers,* New York: McGraw Hill 1965.

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

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

[14] 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.

[15] 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.