# Challenges in Applications of Computational Intelligence in Industrial Electronics

Bogdan M. Wilamowski
Auburn University
wilam@ieee.org

***Abstract*-The presentation is focused on comparison of neural networks and fuzzy systems. Advantages and disadvantages of both technologies are discussed. Fuzzy systems are relatively easy to design but number of inputs in the system are significantly limited. It is very difficult to design neural networks so rather they have to be trained instead. Neural networks produce much smoother nonlinear mapping than fuzzy systems. When neural networks are selected then researchers are facing two dilemmas: what should be neural network architectures and how to train them. The presentation gives answers for both problems. It was found that Bridged Multilayer Preceptron BMLP are a much better architecture than popular MLP architecture. It is faster to train and more complex problems can be solved with fewer neurons. Training of neural networks is not easy. For example most of the existing software cannot train close to optimal neural network so networks with excessive number of neurons are being used by most researchers. Such networks indeed can be trained to very small errors using training patterns but they are not able to respond correctly for new patterns not used in training. A new NBN learning algorithm is presented in this work. This algorithm is not only up to 1000 times faster than the popular EBP algorithm, but it can train all neural network architectures. More importantly it can train close to optimum neural networks which were not able to be trained before.**

## I. INTRODUCTION

There are three major areas of computational intelligence: Neural networks, fuzzy systems, and evolutionary computation. The complexity of neural networks are such that humans are not able to design them. Instead we produce neural networks with random connections and then we train them so they can perform required function. Per analogy we as humans are not able to design dogs and we are not able to follow operations of dogs' neurons, but we are able to train them. On other hand, fuzzy systems are relatively simple and we can design them for required applications. The current approach for fuzzy systems allows us to design only systems with number of inputs limited to 3 or 4 and with one output only. Neural networks do not have these limitations and also the qualities of nonlinear mapping of neural networks are superior to fuzzy systems [1][2]. In evolutionary computation the design process is replaced by a selection process out of many populations of solutions. In this presentation we will focus on neural networks and fuzzy systems since applications of evolutionary algorithms are not used in real time applications.

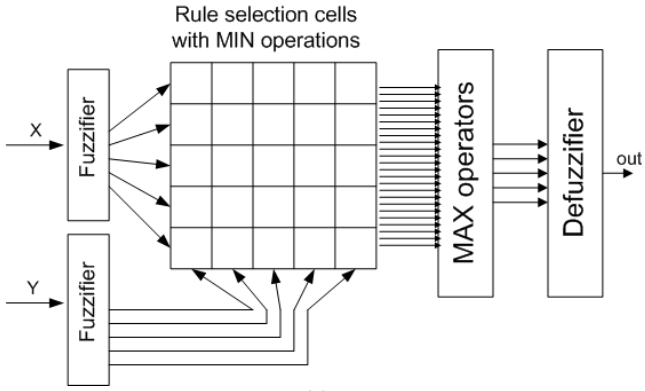
| MIN | | $A \cap B$ | MAX | | $A \cup B$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

(a)

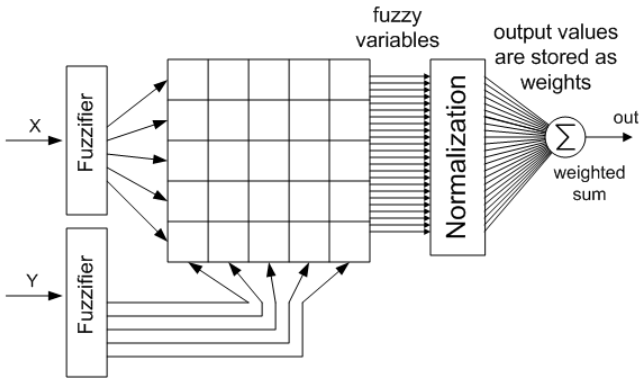
| MIN | | $A \cap B$ | MAX | | $A \cup B$ |
|---|---|---|---|---|---|
| 0.2 | 0.3 | 0.2 | 0.2 | 0.3 | 0.3 |
| 0.2 | 0.7 | 0.2 | 0.2 | 0.8 | 0.7 |
| 0.8 | 0.3 | 0.3 | 0.8 | 0.3 | 0.8 |
| 0.8 | 0.7 | 0.7 | 0.8 | 0.7 | 0.8 |

(b)

Fig. 1. Comparison of (a) Boolean and (b) Fuzzy logic.




(a)




(b)

Fig. 2 Fuzzy architectures (a) Mamdani and (b) TSK

## II. FUZZY SYSTEMS

The fuzzy logic is similar to Boolean logic but instead of AND and OR operators, MIN and MAX operators are used. The concept is shown in Fig. 1. Sometimes MIN operator is replaced by product operator in order to obtain smoother response. As a consequence it is possible to perform logic operations on analog values in the range between 0 and 1. The key issue is how to design fuzzy systems so proper nonlinear mapping between input and output can be obtained.

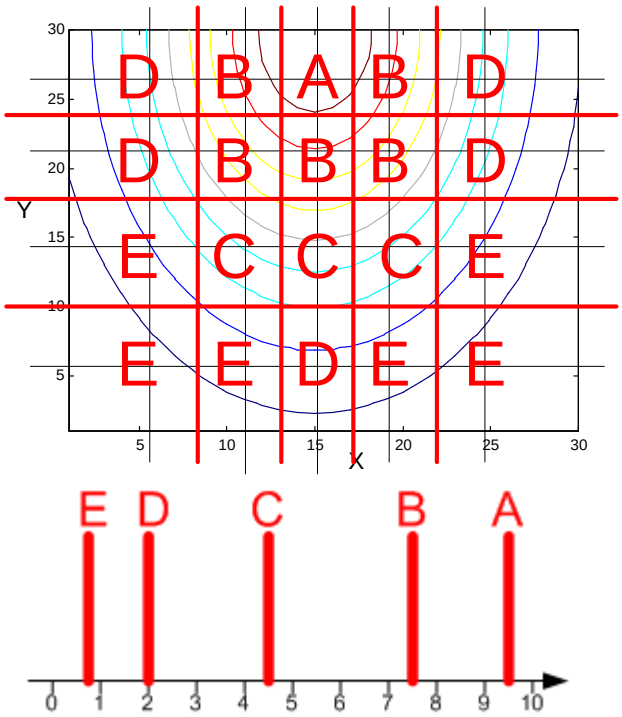



Fig. 3. Design process for Mamdani type fuzzy controller

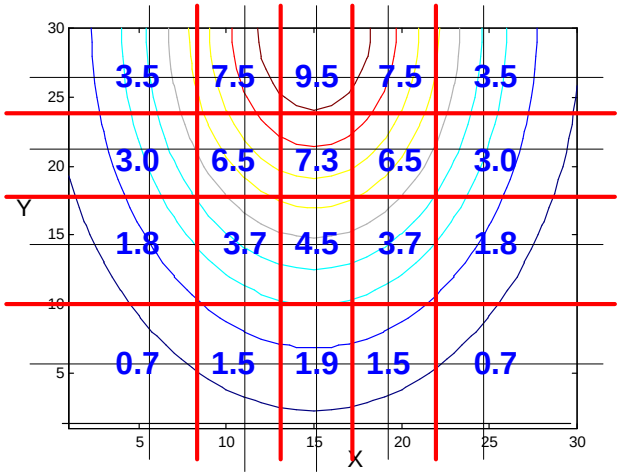



Fig. 4. Design process for TSK type fuzzy controller

There are two most commonly used approaches for development of fuzzy systems. Fig. 2.a shows architecture proposed by Mamdani [3] and Fig. 2.b shows TSK architecture developed by Takagi, Sugeno, and Kang [4][5]. The block diagram of the Mamdani approach [3] is shown in Fig. 2.a. 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.

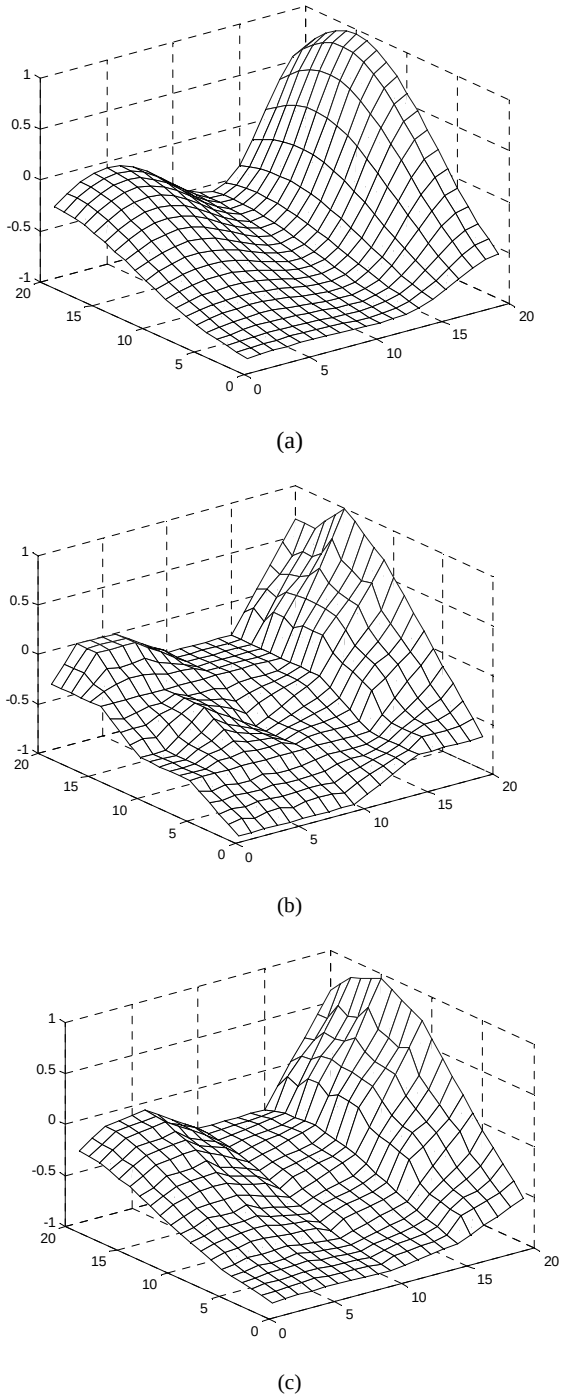



Fig. 5. Example of control surface obtained with fuzzy systems (a) desired surface, (b) surface obtained with Mamdani architecture, (c) surface obtained with TSK architecture

The popular TKS architecture follows well known concept of Look up Table (LUT) implemented in ROM and PLA memories. At first there is a selection of the area then a specific value is assigned to this area. Mamdani type of architectures are more restrictive because instead of assigning output values to each area, as in TKS, one fuzzy variable from the predefined output fuzzy set is assigned to each area. Then the defuzzification process is performed in order to obtain the output variable.

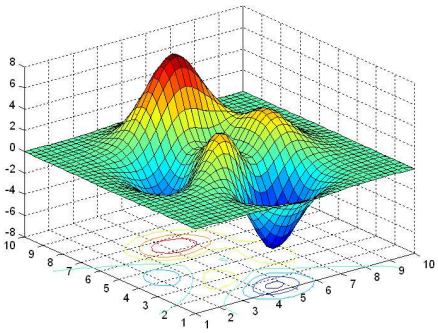
Let us assume that we have to design the Mamdani fuzzy controller having the required function shown in Fig. 5.a. Notice that for every area one specific fuzzy value is assigned of the output fuzzy set. This is marked on Fig. 3 by letters from A to F.

In TSK fuzzy architecture the defuzzification block was replaced with normalization and weighted average (Fig. 2.b). The TSK architecture does not require MAX operators, but a weighted average is applied directly to regions selected by MIN operators (Fig. 4). 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.

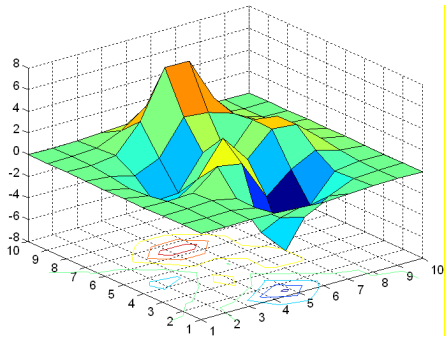
Samples of surfaces obtained with Mamdani and TSK architectures using triangular membership functions are shown in Fig. 5.

## III. NEURAL NETWORKS

The most important feature of neural networks is their generalization abilities. This means that neural networks should correctly respond to new patterns which were never used in the training. The number of neurons in such networks should be as small as possible. Unfortunately it is very difficult to train neural networks with good generalization abilities. In order to reduce number of neurons special network architectures have to be used. Also, more advanced learning algorithms than popular EBP algorithm should be used. These issues will be discussed in following sections.

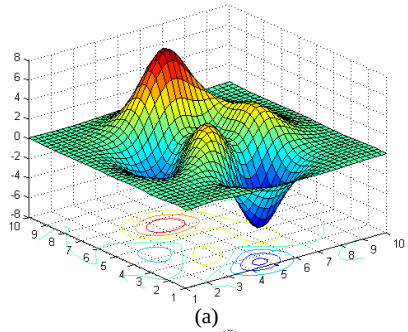


(a)

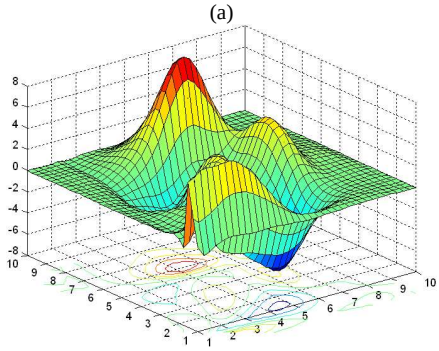


(b)

Fig. 6 Surface matching problem: (a) Required 2-D surface with *37×37=1,369* points, used for verification; (b) *10×10=100* training patterns extracted in equal space from (a), used for training.



(a)



(b)

Fig. 7 Training results using 100 trials with (a) NBN algorithm, 8 neurons in FCC network (52 weights); maximum training iteration is 1,000; $SSE_{Train}$=0.0044, $SSE_{Verify}$=0.0080 and training time=0.37 s, (b) EBP algorithm, 13 neurons in FCC network (117 weights); maximum training iteration is 1,000,000; $SSE_{Train}$=0.0018, $SSE_{Verify}$=0.4909 and training time=635.72 s,

### A. Generalization Abilities

It is relatively easy to find neural network architectures so they can be trained to very small errors. However, it is more important to find an architecture which after training will respond correctly to patterns which were not used for training. Let us illustrate this problem using an example with the peak surface [7] shown in Fig. 6.a as the required surface and let us use equally spaced *10×10=100* patterns (Fig. 6.b) in training neural networks. The quality of trained networks is evaluated using errors computed for equally spaced *37×37=1,369* patterns. In order to make a valid comparison between training and verification error, the SSE, as defined in (1), is divided by 100 and 1,369 respectively.

TABLE I. Training Results of peak surface problem.

| Neurons | Success Rate | | Average Iteration | | Average Time (s) | |
|---|---|---|---|---|---|---|
| | EBP | NBN | EBP | NBN | EBP | NBN |
| 8 | 0% | 5% | / | 222.5 | / | 0.33 |
| 9 | 0% | 25% | / | 214.6 | / | 0.58 |
| 10 | 0% | 61% | / | 183.5 | / | 0.70 |
| 11 | 0% | 76% | / | 177.2 | / | 0.93 |
| 12 | 0% | 90% | / | 149.5 | / | 1.08 |
| 13 | 35% | 96% | 573,226 | 142.5 | 624.88 | 1.35 |
| 14 | 42% | 99% | 544,734 | 134.5 | 651.66 | 1.76 |
| 15 | 56% | 100% | 627,224 | 119.3 | 891.90 | 1.85 |

As the training results are shown in Table I, using the NBN algorithm, which can handle arbitrarily connected neural networks, it was possible to find the acceptable solution (Fig. 7a), $SSE_{Train}$=0.0044 and $SSE_{Verify}$=0.0080) with 8 neurons (52 weights). Unfortunately, with the EBP algorithm, it was not possible to find acceptable solutions in 100 trials within 1,000,000 iterations each. The best result out of the 100 trials with 1,000,000 iterations each was $SSE_{Train}$=0.0764 and $SSE_{Verify}$=0.1271. When the network size was significantly increased from 8 to 13 neurons (117 weights), the EBP algorithm was able to reach a similar training error as with NBN algorithm, but the network lost its ability to respond correctly for new patterns (between training points). Please notice that indeed with enlarged number of neurons, the EBP algorithm was able to train the network to small error $SSE_{Train}$=0.0018, but as one can see from Fig. 7b, the result is unacceptable with verification error $SSE_{Verify}$=0.4909. When reduced number of neurons are used the EBP algorithm can't converge to required training error. When the size of networks increase, the EBP algorithm can reach the required training error, but trained networks lose their generalization ability and can't process new patterns well (Fig. 7b). On the other hand, second order algorithms, such as a NBN algorithm [8][9][10], works not only significantly faster but it can find good solutions with close to optimal networks (Fig. 7a).

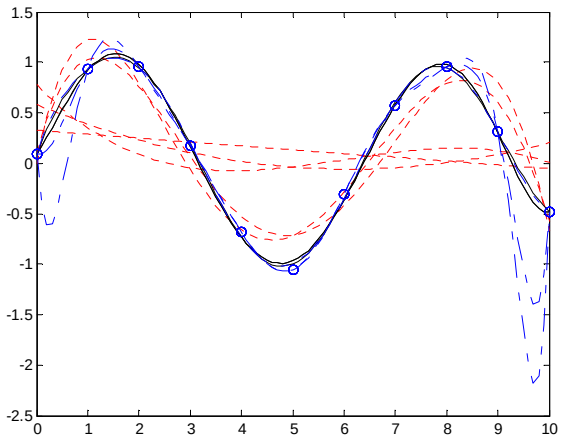



Fig. 8. Approximation of data by different orders of polynomials

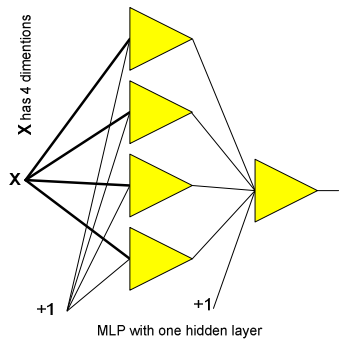



Fig.9. MPL-Multi Layer Perceptron topology with 4 inputs in one hidden layer

From the above analysis, one may notice that in order to sustain neural network generalization abilities the network should have as few neurons/weights as possible. This problem is very similar to function approximation by polynomials. If too high order of polynomial is used then errors for training points and values between points cannot be evaluated correctly. In the example on Fig. 8 only 5-th, 6-th, and 7-th order of polynomials are giving adequate results, while higher order polynomial can be tuned to smaller errors for given points they are useless to predict evaluated new points which were not used for training.

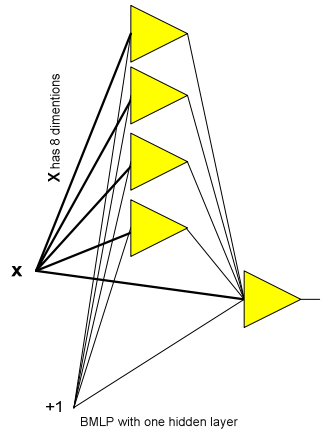



Fig 10. BMPL – Bridged Multi Layer Perceptron topology with 8 inputs and one hidden layer.

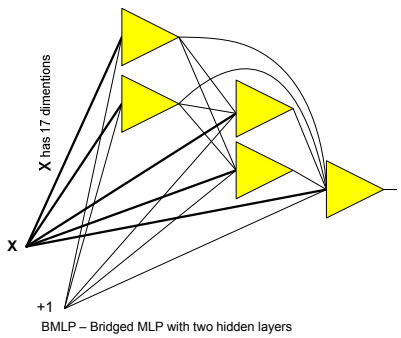



Fig. 11. BMPL – Bridged Multi Layer Perceptron topology with 17 inputs and two hidden layers

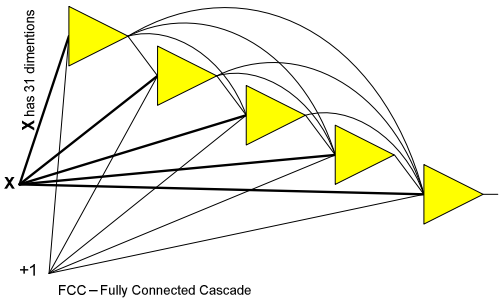


Fig. 12. FCC- Fully Connected Cascade with 5 neurons.

### B. *Neural Network Architectures*

There are many ways to connect neurons in feedforward networks. The most popular is MLP – Multilayer Perceptron (see Figs 9 through 12). The abbreviated description for these particular networks would be **4-4-1**. Much more powerful are BMLP Bridged Multilayer Perceptron networks (see Figs. 10 and 11). The abbreviated description for shown networks are **8=4=1** and **17=2=2=1.** The FCC – Fully Connected Cascade neural network (Fig. 12) is the special case of BMLP where there is only one neuron in each layer. The abbreviated description of the network of Fig. 12 would be **31=1=1=1=1=1**.

The most common test benches for neural networks are parity-N problems, which are considered to be the most difficult benchmark for neural network training. The simplest parity-2 problem is also known as the XOR problem. The larger the N, the more difficult it is to solve. It was shown [11][12] how to solve parity N problems using various neural network architectures such as: MLP with one hidden layer, Bridged Multilayer Perceptron (BMLP) with one hidden layer and FCC architectures. In the case of MLP with *n* neurons in the one hidden layer it is possible to solve Parity-N problem

$$[N-n-1] \quad \Rightarrow \quad N \le n \qquad (1)$$

In the case of BMLP with one hidden layer [10][11]

$$[N=n=1] \quad \Rightarrow \quad N \le 2(n+1)-1 = 2n+1 \qquad (2)$$

and in the case of FCC architecture

$$\left[N = \overbrace{1=1=\cdots=1}^{n}\right] \quad \Rightarrow \quad N \le 2^n - 1 \qquad (3)$$

In the case of BMLP architectures the situation is slightly more complicated but BMLP with 2 hidden layers with *n* neurons in the first layer and *m* neurons in the second layer

$$[N=n=m=1] \quad \Rightarrow \quad N \le 2(n+1)(m+1)-1 \qquad (4)$$

For 3 hidden layers in BMLP architecture with *m*, *n*, *k* neurons in each layer the maximum possible parity problem is:

$$[N=n=m=k=1] \Rightarrow N \le 2(n+1)(m+1)(k+1)-1 \qquad (5)$$

Please notice that in FCC architecture there is only one neuron in each layer (n=1; m=1; k=1) and one neuron in the output layer. So the equations (3) and (5) give identical results:

$$N \le 2(1+1)(1+1)(1+1)-1 = 15 = 2^4 - 1 \qquad (6)$$

Fig. 13 show comparisons of the efficiency of various neural network architectures. For example with 8 neurons MLP with one hidden layer the largest problem which can be solved is the Parity-7. With the same 8 neurons using BMLP various parity problems can be solved starting with Parity-15 for the case of one hidden layer (**15=7=1**) to Parity=72 three hidden layers when are used (**72=3=2=2=1** or **72=2=3=2=1** or **72=2=2=3=1** ). With 8 neurons using FCC topology (which can be also considered as BMLP with 7 hidden layers) the neural network can solve as large problem as Parity-255 using (**255=1=1=1=1=1=1=1=1**) topology.

Another example is the popular test bench with Wieland two spiral problem which can be solved (Fig.14) with second order algorithm using cascade architecture with 8 neurons but in order to solve the same problem with the EBP algorithm (Fig 15) at least 13 neurons and weights in cascade architecture are needed. With only 12 neurons in cascade, the NBN algorithm can produce a very smooth response (Fig. 16) with less than 150 iterations but we were not able to solve this 12 neuron problem with EBP algorithm despite many trials with 1,000,000 iterations limit [13].

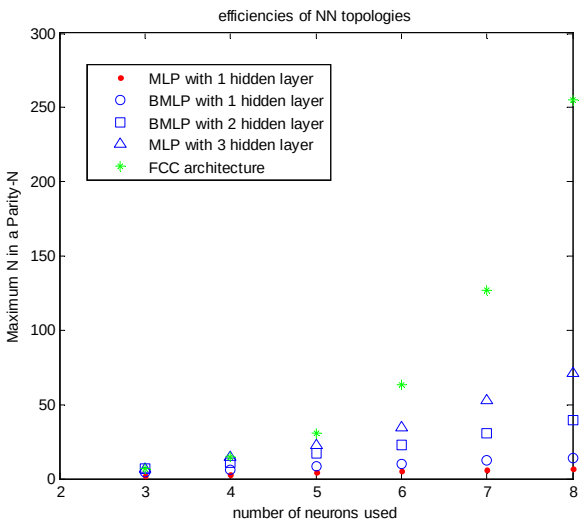


Fig.13. Abilities of solving Parity-N problems as function of number of neurons.

### C. *Neural Network Training Algorithms*

The most common training algorithm is EBP – Error Back Propagation [13][14]. It is relatively simple and it does not

require a lot of computer resources. This algorithm however seldom leads to a good solution and is extremely slow. Much better results can be obtained with second order algorithms such as LM – Levenberg-Marquardt algorithm [15] or NBN – Neuron by Neuron algorithm [7][8].

Tables II and III show a comparison of these algorithms for the parity-4 problem and the MLP architecture (**4-3-3-1**) (Table II) and for the FCC architecture (**4=1=1=1**) (Table III). The LM was not developed for arbitrarily connected neural networks [16]. Please notice that the popular MATLAB Neural Network Toolbox [11] where LM algorithm is implemented can also handle only MLP architectures. The SNNS software [15] which can handle arbitrarily connected neural networks unfortunately uses only first order algorithms. The Neuron by Neuron (NBN) algorithm, implemented in NNT software package [10], was developed in order to eliminate most disadvantages of the LM algorithm and can handle arbitrarily connected neural networks.

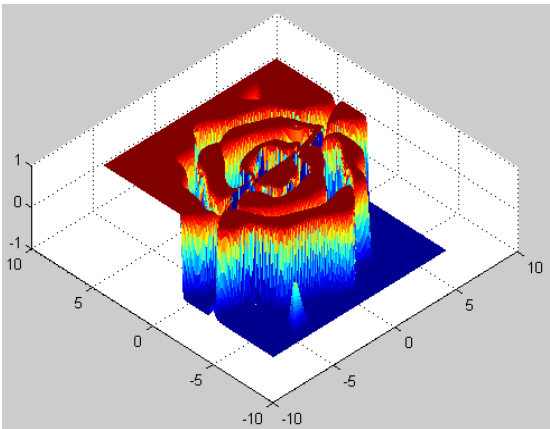



Fig. 14. Solution of the two spiral problem with NBN algorithm [4] using fully connected architecture with 8 neurons and 52 weights.

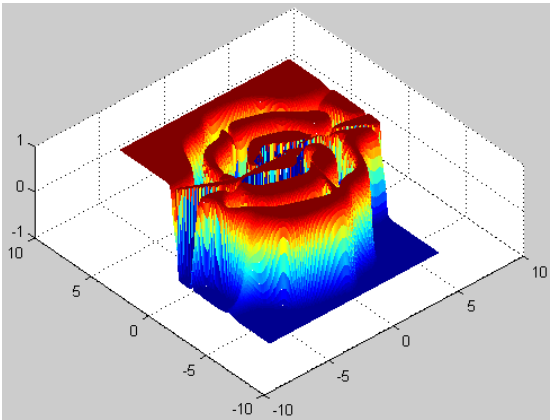



Fig. 15. Solution of the two spiral problem with EBP algorithm using fully connected architecture with 16 neurons and 168 weights

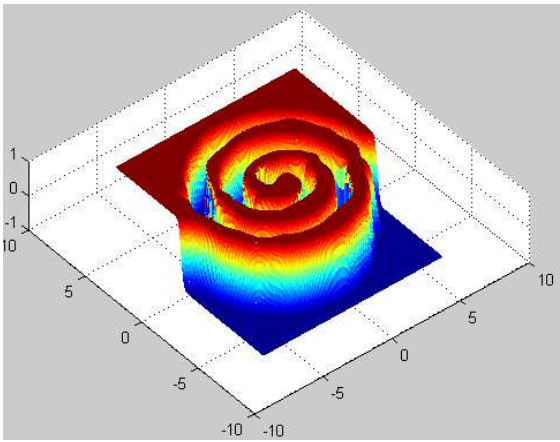



Fig. 16. Solution of the two spiral problem with NBN algorithm [4] using fully connected architecture with only 12 neurons and 102 weights

Table IV shows training results for parity-4 problems and FCC architecture, but the size of the network is increased. In this case both EBP and NBN algorithms can easily find solutions.

TABLE II Average data for parity-4 training and MLP architecture 4-3-3-1

| | EBP | LM | NBN |
|---|---|---|---|
| *Number of iterations* | 2438.9 | 19.7 | 19.6 |
| *Training time[ms]* | 931.9 | 19.6 | 19.6 |
| *Success rate* | 90% | 72% | 82% |

TABLE III Average data for parity-4 training and FCC architecture 4=1=1=1

| | EBP | LM | NBN |
|---|---|---|---|
| *Number of iterations* | 7568.8 | NA | 14.6 |
| *Training time[ms]* | 2985.8 | NA | 8.7 |
| *Success rate* | 88% | NA | 98% |

TABLE IV Average data for parity-4 training and FCC architecture 4=1=1=1=1

| | EBP | LM | NBN |
|---|---|---|---|
| Number of iterations | 3977 | NA | 12.4 |
| Training time[ms] | 1582 | NA | 8.15 |
| Success rate | 98% | NA | 100% |

Fig. 17 shows surfaces produced by neural networks for the same desired surface as it was used with fuzzy systems (Fig. 5). One may notice that neural networks produces much smoother and more accurate surfaces.

## IV. CONCLUSION

As was already mentioned the success rate increases with the increased size of neural network but such networks with excessive number of neurons are losing their generalization abilities. If too many neurons are used, then the network can be over-trained on the training patterns, but it will fail on patterns never used in training. With a smaller number of neurons, the network cannot be trained to very small errors, but it may produce much better approximations for new patterns. The most common mistake made by many researchers is that in order to speed up the training process and to reduce the training errors they use neural networks with larger number of neurons than required. Such networks would perform very poorly for new patterns not used for training [9].

Neural networks exhibit superior performance in comparison to fuzzy systems but there are several reasons for frustration of people trying to adapt neural networks for their research:

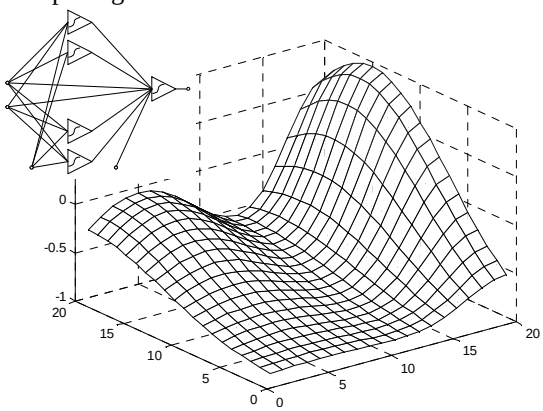
(1) In most cases the relatively inefficient MLP architecture is used instead of more powerful topologies which allow connections across layers.

(2) When popular learning software is used, such as EBP, the training process is not only very time consuming, but EBP is often not able to find solutions for the neural network with reduced number of neurons.

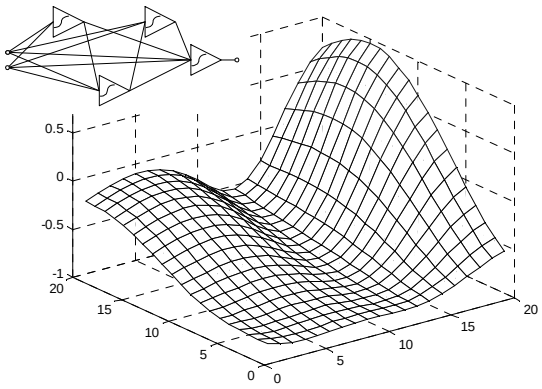
(3) As neural network complexity increases neural networks can be over-trained to the training data, losing its ability for generalization; therefore, it is not able to correctly process new patterns which were not used for training.

(4) In order to find solutions for close to optimal architectures, second order algorithms such as NBN or LM should be used. Unfortunately, the LM algorithm adopted in popular MATLAB NN Toolbox can handle only MLP topology without connections across layers and these topologies are far from optimal.

With the recently developed advanced learning algorithm [6] it is possible to train those networks, which cannot be trained with simple algorithms.




(a)




(b)

Fig.17. Example of control surface obtained with neural systems (a) with 5 neurons in BMLP architecture , (b) with 4 neurons using FFC architecture. Required surface is shown in Fig. 5.a.

## REFERENCES


[1] B. M. Wilamowski, “Neural Networks and Fuzzy Systems for Nonlinear Applications” (keynote) *11th INES 2007 -11th INES 2007 -International Conference on Intelligent Engineering Systems*, Budapest, Hungary, June 29 2007-July 1 2007, pp. 13-19.

[2] Bodgan M. Wilamowski, “Methods of Computational Intelligence for Nonlinear Control Systems” (keynote talk) ICCAE’ 05 International Conference on Control, Automation and System, June 2-5, 2005, KINTEX, Gyeonggi-Do, Korea, pp. P1-P8

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

[4] 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.

[5] Mike McKenna and Bogdan Wilamowski, " Implementing a Fuzzy System on a Field Programmable Gate Array ", *International Joint Conference on Neural Networks (IJCNN'01)*, pp. 189-194, Washington DC, July 15-19, 2001

[6] B. M. Wilamowski and H. Yu, "Improved Computation in Levenberg Marquardt Training," *IEEE Trans. on Neural Networks* (accepted for publication).

[7] Wilamowski, B.M. Cotton, N.J. Kaynak, O. Dundar, G., “Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks”, *IEEE Trans. on Industrial Electronics*, vol. 55, no. 10, pp. 3784-3790, Oct. 2008.

[8] B. M. Wilamowski, "Neural Network Architectures and Learning Algorithms," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 56-63, Dec. 2009

[9] Hao Yu and B. M. Wilamowski, “C++ Implementation of Neural Networks Trainer”, 13-th *International Conference on Intelligent Engineering Systems, INES-09*, Barbados, April 16-18, 2009

[10] B. M. Wilamowski, D. Hunter, A. Malinowski, "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

[11] B. M. Wilamowski, "Efficient neural network architectures and advanced training algorithms", ICIT-10, 3-rd International Conference on Information Technology, Gdansk, Poland, June 28-30, 2010.

[12] B. M. Wilamowski, “Special Neural Network Architectures for Easy Electronic Implementations” *POWERENG.2009*, Lisbon, Portugal, March 18-20, 2009, pp. 17-22Rumelhart D. E., G. E. Hinton, R. J. Williams, “Learning representations by back-propagating errors”. *Nature*, vol. 323, pp. 533-536, 1986.

[13] Werbos P. J., “Back-propagation: Past and Future”. *Proceeding of International Conference on Neural Networks*, San Diego, CA, 1, 343-354, 1988.

[14] Stuttgart Neural Network Simulator SNNS http://www.ra.cs.uni-tuebingen.de/SNNS/

[15] M. T. Hagan, M.B. Menhaj, “Training feedforward networks with the Marquardt algorithm”. *IEEE Trans. on Neural Networks*, vol. 5, no. 6, pp. 989-993, Nov. 1994.