How to not get frustrated with neural networks

Bogdan M. Wilamowski
Auburn University, AMNSTC, Auburn, Alabama, USA
wilam@ieee.org

Abstract, **In the presentation major difficulties of designing neural networks are shown. It turn out that popular MLP (Multi Layer Perceptron) networks in most cases produces far from satisfactory results. Also, popular EBP (Error Back Propagation) algorithm is very slow and often is not capable to train best neural network architectures. Very powerful and fast LM (Levenberg- Marquardt) algorithm was unfortunately implemented only for MLP networks. Also, because a necessity of the inversion of the matrix, which size is proportional to number of patterns, the LM algorithm can be used only for small problems. However, the major frustration with neural networks occurs when too large neural networks are used and it is being trained with too small number of training patterns. Indeed, such networks, with excessive number of neurons, can be trained to very small errors, but these networks will respond very poorly for new patterns, which were not used for training. The most of frustrations with neural network can be eliminated when smaller, more effective, architectures are used and trained by newly developed NBN (Neuron-by-Neuron) algorithm.**

I. Introduction

The most popular neural network architecture is the MPL (Multi Layer Perceptron) topology. It will be shown in this presentation, the MLP neural network is not very effective. The MLP architectures require not only more neurons than other topologies, but it is also more difficult to train than neural networks where connections across layers are allowed. Therefore, if people are using not optimal network architectures then the results are usually not satisfactory. A comparison of different neural network architectures will be given in Section II.

Once a better architecture if found the next problem is how to train it? The issue is difficult because most of popular of neural network software can train only MPL architectures. The SNNS software [1] is the exception and it allows for training arbitrarily connected feed forward neural networks. Unfortunately, SNNS uses only first order learning algorithms such as EBP (Error Back Propagation) [1-3] and its derivatives. As a result, training is not only slow but often ineffective. Advanced second order algorithm such as LM (Levenberg- Marquardt)[4-5] are also implemented in the MATLAB Neural Network ToolBox [6] can train only MLP networks. As the consequence without proper learning software researchers have no other option but to use MLP architectures and obtained results are far from satisfactory. As a consequence more and more people are becoming frustrated with feed forward neural networks and its training process. Therefore, often researchers are reaching to other techniques such as fuzzy systems, Radial Basis Function (RBF) networks, or Support Vector Machines (SVM). Unfortunately these other approaches have their faults too.

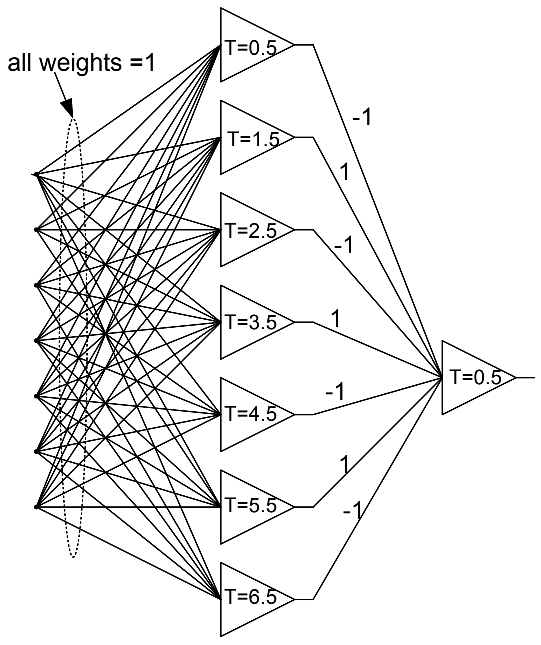
The good news is that recently developed NBN (Neuron by Neuron)[7-9] algorithm is not only fast and effective, but it can train any neural network with feed forward architecture. A comparison of effectives of various learning algorithm is given in section III.

It is known [8, 10,11] that it is always easier to obtain smaller error if large number of neuron is used in the network. Therefore many researchers are being trapped here. In order to speed up the learning process or reduce the training error an excessive number of neurons are often used and neural networks are losing their generalization abilities. In other words networks indeed can be trained to smaller errors, but such networks are then responding very poorly to new patterns not used for training. This is probably the main reason for a frustration with neural networks. The generalization issue will be discussed in details in Section IV.

II. Neural Network Architectures

The most common test benches for neural networks architecture are parity-N problems. The parity-N problems 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. Even though parity-N problems are very difficult it is possible to analytically design neural networks for these problems [9]. As a design example let us design neural networks for the parity-7 problem using different neural network architectures with unipolar neurons.

Fig. 1 shows the MLP architecture with one hidden layer. In order to properly classify patterns in parity-N problems the location of zeros and ones in input patterns are not relevant but it is only important how many ones are in the patterns. Therefore one may assume identical weights connected to all inputs equal +1. Depending on number of ones in the pattern net values of neurons of hidden layer calculated as sum of inputs times weights may vary from 0 to 7 and it would be equal to number of ones in an input pattern. In order to separate these 8 possible cases we need 7 neurons in the hidden layer with thresholds equal to: 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5. Let us assign positive (+1) and negative (-1) weights to outputs of consecutive neuron starting with +1. One may notice that the net value of the output neuron will be zero for patterns with odd number of ones and one with even number of ones. The threshold of +0.5 of the last neuron will just reinforce the same values on the output. The signal flow for this network is shown in the table on Fig. 1.



number of ones in a pattern	0	1	2	3	4	5	6	7
net1 through net7 (from inputs only)	0	1	2	3	4	5	6	7
out1 $T=0.5$ $w_1=1$	0	1	1	1	1	1	1	1
out2 $T=1.5$ $w_2=-1$	0	0	1	1	1	1	1	1
out3 $T=2.5$ $w_3=1$	0	0	0	1	1	0	1	1
out4 $T=3.5$ $w_4=-1$	0	0	0	0	1	1	1	1
out5 $T=4.5$ $w_5=1$	0	0	0	0	0	1	1	1
out6 $T=5.5$ $w_6=-1$	0	0	0	0	0	0	1	1
out7 $T=6.5$ $w_7=1$	0	0	0	0	0	0	0	1
$net8 = net1 + \sum_{i=1}^{7} w_i * out_i$	0	1	0	1	0	1	0	1
out8 (of output neuron) $T=0.5$	0	1	0	1	0	1	0	1

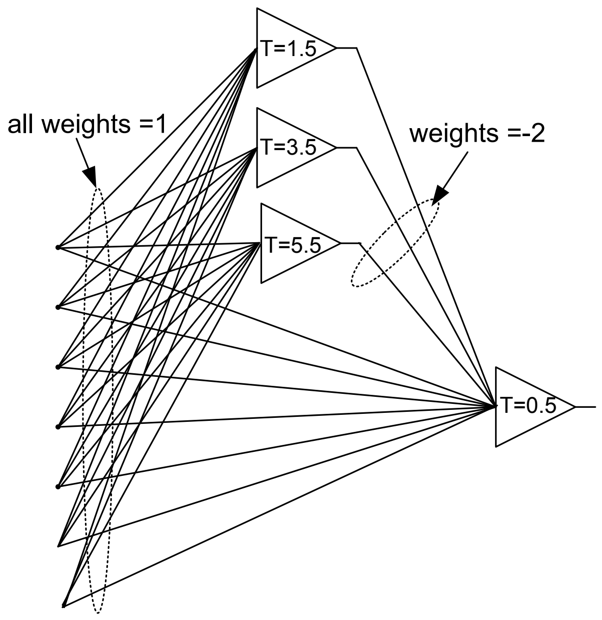
Fig. 1 Multi Layer Perceptron (MLP) architecture for parity-7 problem. The computation process of the network is shown in the table.

In the case of MLP neural network the number of neurons in the hidden layer is equal to N=7 and total number of neurons is 8. For MLP architecture with *n* neurons in one hidden layer maximum value of N of parity-N problem is [12]:

$$N = n \qquad (1)$$

Fig. 2 shows solutions with Bridged Multi Layer Perceptron (BMLP) with connections across layers. With this approach the neural network can be significantly simplified. Only 3 neurons are needed in the hidden layer with thresholds equal to 1.5, 3.5, and 5.5. In this case all weights associated with outputs of hidden neurons must be equal to -2 while all remaining weights in the network equal to +1

Fig. 2 shows solutions with Bridged Multi Layer Perceptron (BMLP) with connections across layers. With this approach the neural network can be significantly simplified. Only 3 neurons are needed in the hidden layer with thresholds equal to 1.5, 3.5, and 5.5. In this case all weights associated with outputs of hidden neurons must be equal to -2 while all remaining weights in the network equal to +1.



number of ones in a pattern	0	1	2	3	4	5	6	7
net1 (from inputs only)	0	1	2	3	4	5	6	7
out1 $T=1.5$	0	0	1	1	1	1	1	1
out2 $T=3.5$	0	0	0	0	1	1	1	1
out3 $T=5.5$	0	0	0	0	0	0	1	1
net1 = ne1 - 2(out1+out2+out3)*	0	1	0	1	0	1	0	1
out4 (of output neuron) $T=0.5$	0	1	0	1	0	1	0	1

Fig. 2 Bridged Multi Layer Perceptron (BMLP) architecture for parity-7 problem. The computation process of the network is shown in the table.

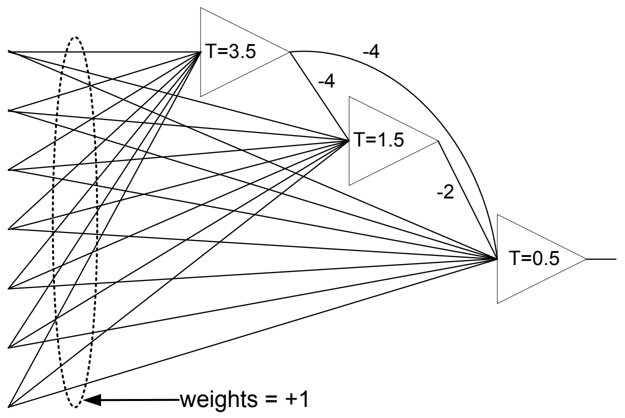
Signal flow in this BMLP network is shown in the table on Fig. 2. With bridged connections across layers the number of hidden neurons was reduced to (N-1)/2=3 and total number of neurons is 4. For BMLP architecture with *n* neurons in one hidden layer maximum value of N of parity-N problem is [12]:

$$N = 2(n+1) - 1 \qquad (2)$$

Fig. 3 shows solution for Fully Connected Cascade (FCC) architecture for the same parity-7 problem. In this case only 3 neurons are needed with thresholds 3.5, 1.5, and 0.5. The first neuron with threshold 3.5 is inactive (out=0) if number of ones in an input pattern is less than 4. If number of ones in input pattern is 4 or more then the first neuron becomes active and with -4 weights attached to its output it subtracts -4 from nets of neurons 2 and 3. Instead of [0 1 2 3 4 5 6 7] these neurons will see [0 1 2 3 0 1 2 3]. The second neuron with threshold of 1.5 and the -2 weight associated with its output makes that the last neuron will see [0 1 0 1 0 1 0 1] instead [0 1 2 3 0 1 2 3]. For other parity-N problems and FCC architecture:

$$N = 2^n - 1 \qquad (3)$$

where *n* is number of neurons used in the FCC network.



number of ones in a pattern	0	1	2	3	4	5	6	7
net1 (from inputs only)	0	1	2	3	4	5	6	7
out1 (of first neuron) T=3.5	0	0	0	0	1	1	1	1
*net2 = net1 - 4*out1*	0	1	2	3	0	1	2	3
out2 (of second neuron) T=1.5	0	0	1	1	0	0	1	1
*net3 = net2 - 2*out2*	0	1	0	1	0	1	0	1
out3 (of output neuron) T=0.5	0	1	0	1	0	1	0	1

Fig. 3 Fully Connected Cascade (FCC) architecture for parity-7 problem. The computation process of the network is shown in the table.

A similar analysis for BMLP networks with various number of hidden layers. For BMLP network with two hidden layers the maximum value of N is:

$$N = 2(n+1)(m+1) - 1 \qquad (4)$$

where n and m are the number of neurons in two consecutive hidden layers.

For BMLP network with three hidden layers the maximum value of N is:

$$N = 2(n+1)(m+1)(k+1) - 1 \qquad (5)$$

where n, m and k are the number of neurons in three consecutive hidden layers. Table I shows comparisons of minimum number of neurons required for these several architectures and various parity-N problems.

Fig. 4 shows comparisons of the efficiency of various neural network architectures [12-14]. By looking on Fig. 4 one may notice that the most commonly used architecture of MLP with one hidden layer gives worst results. For example with 10 neurons it is possible to solve only Parity-9 problem. With the same 10 neurons, when FCC architecture is used it is possible to solve as large problems as Parity-1023. Various BMLP networks depending on the depth of the network entertain a powers between MLP and FCC networks.

One may withdraw the conclusion that with BMLP architectures the capabilities of neural networks rapidly increase with the depth of the network and that the FCC architecture is the most powerful. Unfortunately most researchers are using MLP networks with one hidden layer which is the least powerful architecture. One reason why more powerful architectures are not used in the practice is that

TABLE I. Minimum number of neurons required for various parity-N problems

	Parity-8	Parity-16	Parity-32	Parity-64
# inputs	8	16	32	64
# patterns	256	65536	4.294e+9	1.84e+19
MLP	**8**	**16**	**32**	**64**
BMLP-1	**5**	**9**	**17**	**33**
BMLP-2	**4**	**5**	**8**	**11**
BMLP-3	**4**	**5**	**7**	**8**
FCC	**4**	**5**	**6**	**7**

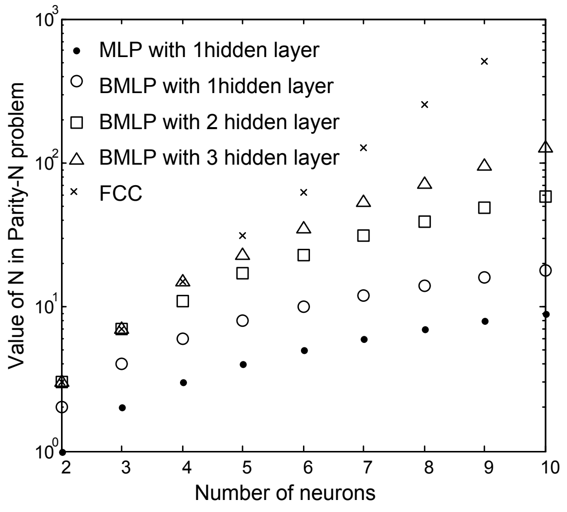


Fig. 4. Abilities of solving Parity-N problems as function of number of neurons.

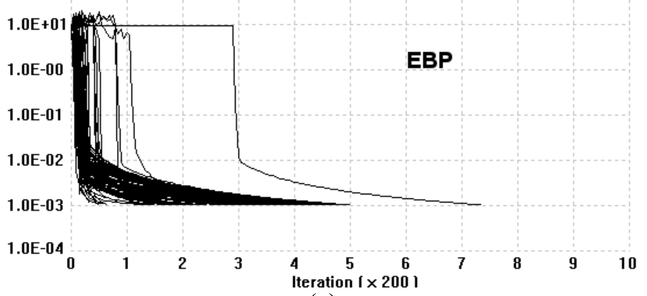
all popular neural network software is not able to train efficiently these more powerful neural networks.

III. Learning Algorithms

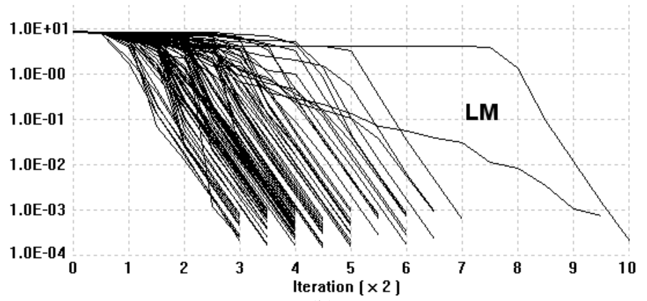
The Error Back Propagation (EBP)[2,3] algorithm is the most popular algorithm but it is very slow and seldom gives good results. The EBP training process requires significantly more iterations and more time than more advanced algorithms such as Levenberg- Marquardt (LM) [4,6] or Neuron by Neuron (NBN) [6-8,14-16] algorithms. (see Table II) and Figure 5) What is most important is that EBP is not only slow but it is not able to find solutions for close to optimum architectures (see Fig. 4 and Table III). Figure 5 shows the training errors as a function of number of iterrations for three algoritms: EBP, LM, and NBN. In the case of EBP and NBN optimal FCC neural network architecture were used with two neurons. Since the LM algorithm was not implemented for this optimal architecture the MLP topology with one hidden layer with three neurons was used.

TABLE II Average data for parity-3 training

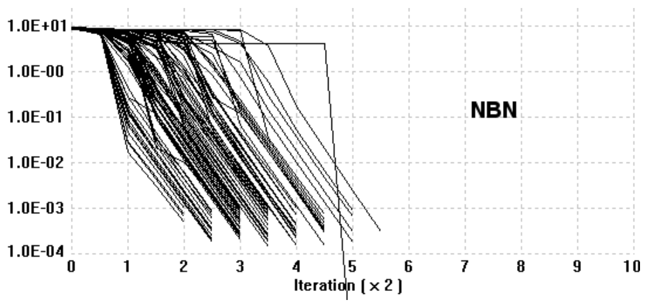
	EBP	LM	NBN
Number of iterations	658.1	8.7	6.8
Training time[ms]	898.37	21.67	16.6
Success rate	100%	100%	100%



(a)



(b)



(c)

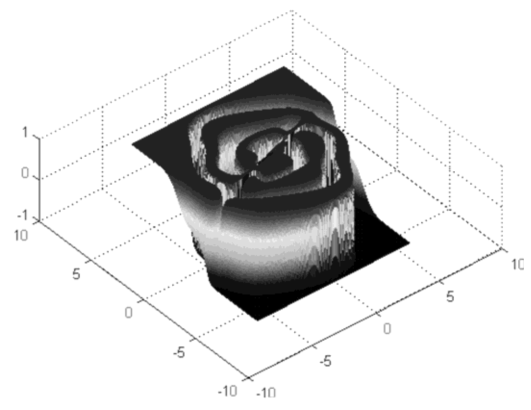
Fig. 5. Learning error as function of number of iterations for three algorithms (a) EBP, (b) LM, (c) NBN. Because LM algorithm is not able to train BMLP architecture with 2 neurons in the case (b) the MLP with 4 neurons were used.

The two-spiral problem is considered as a good evaluation of training algorithms [14-17]. Depending on the neural network architecture, different numbers of neurons are required for successful training. For example, using standard MLP networks with one hidden layer, 34 neurons are required for two-spiral problem [18]; while with FCC architecture, it can be solved with only 8 neurons. Second order algorithms are not only much faster but they can train reduced size networks which can't be handled by the EBP algorithm (See Table III). Notice that the LM algorithm was not used in this comparison because it is not capable to train FCC or BMLP neural networks.

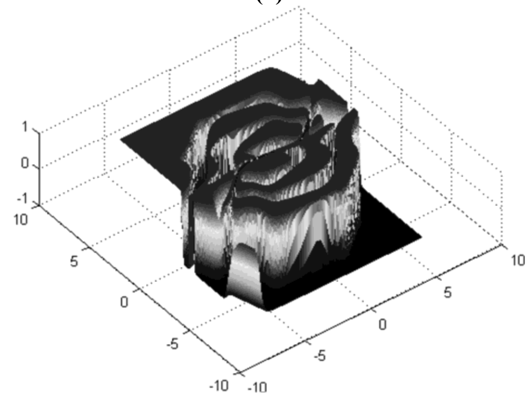
Table III presents the training results of the two-spiral problem using FCC networks with different number of neurons [11]. NBN algorithm can solve the two-spiral problem, using 8 neurons (52 weights) in nearly 290 iterations (Fig. 6(a)). The EBP algorithm can solve the two-spiral problem only when larger networks are used. When the number of neurons is increased to 12 (102 weights), EBP algorithm can solve it in about 400,000 iterations. The result (the best one in 100 trials), shown in Fig. 6(b), is not as good as the result (Fig. 6(a)) from NBN algorithm with much simpler architecture. One can conclude that the EBP algorithm is only successful if excessive number of neurons is used, but then the neural network generalization abilities are lost (poor response for patterns not used in training)

TABLE III. Training results of two-spiral problem.

Neurons	Success Rate		Average Iteration		Average Time (s)	
	EBP	NBN	EBP	NBN	EBP	NBN
8	0%	13%	/	287.7	/	0.88
9	0%	24%	/	261.4	/	0.98
10	0%	40%	/	243.9	/	1.57
11	0%	69%	/	231.8	/	1.62
12	63%	80%	410,254	175.1	633.91	1.70
13	85%	89%	335,531	159.7	620.30	2.09
14	92%	92%	266,237	137.3	605.32	2.40
15	96%	96%	216,064	127.7	601.08	2.89
16	98%	99%	194,041	112.0	585.74	3.82



(a)



(b)

Fig. 6. Best results of two-spiral problem in 100 trails: (a) 8 neurons in FCC network (52 weights), using NBN algorithm and training time=0.82 s; (b) 12 neurons in FCC network (102 weights), using EBP algorithm and training time=694.32 s

As one can see from the 2 spiral example that the EBP algorithm can't converge to required training error unless a significant number of excessive neurons are used. 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 The newly developed NBN algorithm [7-9, 14-17] works not only significantly faster than EBP (or even faster than LM algorithm) and it can find good solutions with close to optimal networks which are very difficult to train. The NBN algorithm eliminates most deficiencies of the LM algorithm; it can be used to train neural networks with arbitrarily connected neurons (not just MLP architecture). It does not require to compute and to store large Jacobians so it can train problems with basically an unlimited number of patterns [8]. Error derivatives are computed only in forward pass, so backward computation process is not needed. It is equally fast, but in the case of networks with multiple outputs faster than LM algorithms. It can train close to optimal feed forward networks which are impossible to train with other algorithms.

IV. The Generalization Issue

With increased number of neurons it is easy to train neural networks. The most common mistake made by many researchers is to increase number of neurons in the system to secure faster convergence. Indeed such larger networks can be trained faster and to smaller errors, but this "success" is very misleading. Such networks with excessive number of neurons are most likely losing their generalization abilities. It means that that will respond very poorly for new patterns never used in the training.

The problem is similar to curve fitting using polynomial interpolation as illustrated in Fig. 7. Notice that if the order of polynomial is too low there is a poor approximation everywhere. When order is too high there is a perfect fit at the given data points, but the interpolation between points is rather poor. In the case of neural networks we have a similar situations with excessive number of neurons it is easy to train the network to very small error at the training data, but this may lead to very poor and frustrating results when this trained neural network is used for to process a new patterns.

In order to have good generalization abilities the number of neurons in the network should be as small as possible to obtain a reasonable training error.

The generalization abilities also depend on the number of training patterns. With a large number of training patterns a larger number of neurons can be used, while generalization abilities are preserved To illustrate the problem with neural networks let us try to find the best neural network architecture to replace a fuzzy controller. Fig. 8 shows the required control surface and the defuzzyfication rules for the TSK (Tagagi, Sugeno, Ken) fuzzy controller [10, 19, 20].

In order to train the developed neural controller we may use TSK defuzzyfication rules as the training patterns. Let us select FCC neural network architecture and try to find solutions for different number of neurons used. Fig. 9 (a) shows results for neural network with 3 neurons (12 weights)

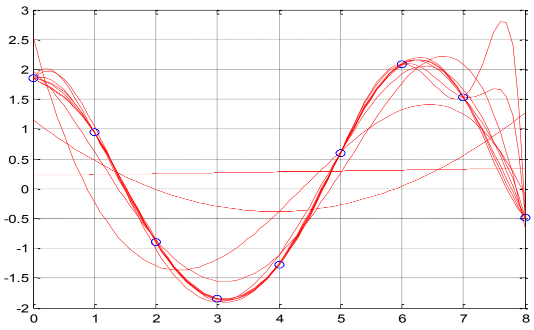


Fig. 7 Approximation of measured points by polynomials with a different orders stating with first through 9-th order.

.

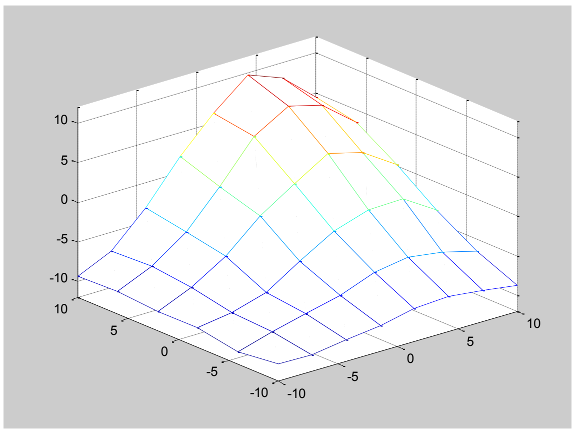


Fig. 8. Control surface of TSK fuzzy controller with 8*6=48 defuzzyfication rules

and Fig. 9 (b) shows results for 4 neurons and 18 weights. However, when the size of the network increases, the results become worse instead of better, even learning errors decrease with the increase of the neural network size.

One may notice that the best results were obtained for the 4 neuron architecture (Fig. 9(b)) With more neurons we obviously are able to reduce the training error, but the neural network loses its generalization ability (see Fig. 10).

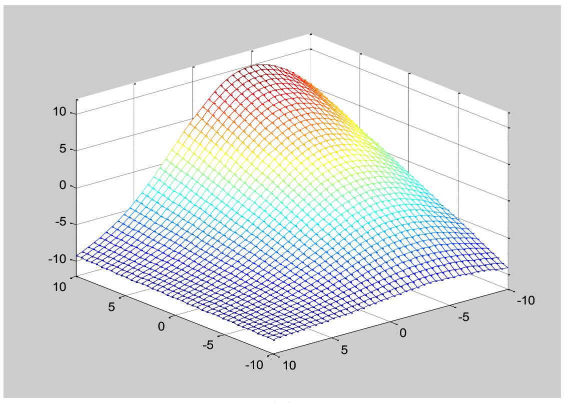
The conclusion is that for optimum performance neural networks should have as few neurons as possible. The software which uses NBN algorithm can be downloaded from [19].

V Conclusion

There are several reasons for frustration with neural networks:

[1] In most cases the relatively inefficient MLP architectures with one hidden layer are used instead of more powerful topologies with connections across layers.

[2] When popular first order learning software is used, such as EBP, the training process is not only very time consuming, but frequently the correct solution is not obtained. EBP algorithm often is not able to find solutions for small and close to optimal neural networks.


(a)

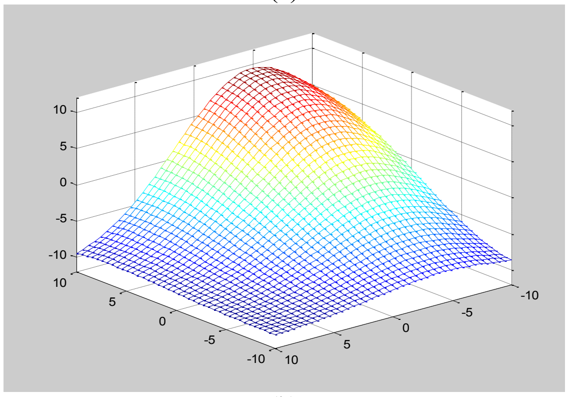
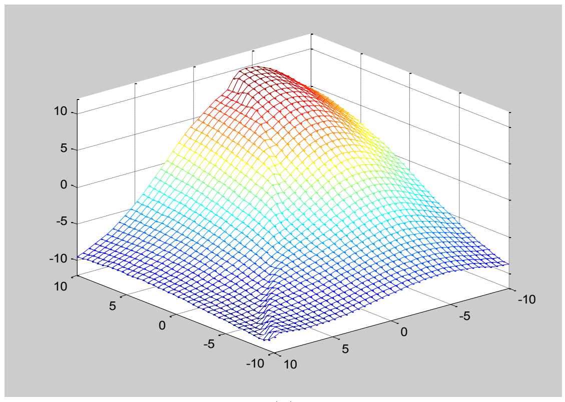

(b)

Fig. 9 Control surface obtained with neural networks (a) 3 neurons in cascade (12 weights) Error=0.21049 (b) 4 neurons in cascade (18 weights) Error=0.049061


(a)

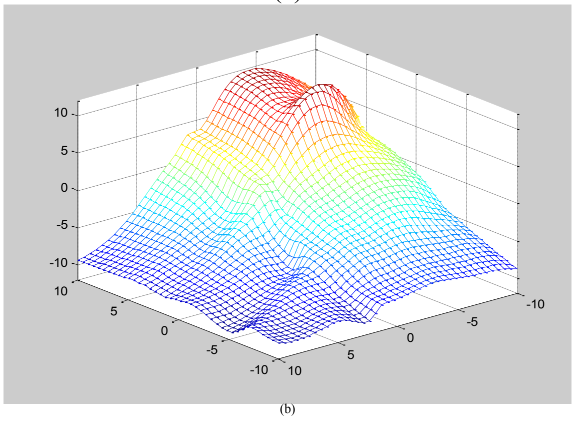

(b)

Fig. 10 Control surface obtained with neural networks (a) 5 neurons in cascade (25 weights) Error=0.023973 (b) 8 neurons in cascade (52 weights) Error=1.118e-005

(3) With increased complexity of neural network (often needed so EBP algorithm can converge) the neural network loses its ability of generalization; therefore, it is not able to process correctly new patterns which were not used for training.

(4) In order of find solutions for close to optimal architectures the second order algorithms such as NBN or LM should be used. Unfortunately, LM algorithm adopted in popular Matlab NN Tool Box [6] can handle only MLP topology without connections across layers and these topologies are far from optimal.

The importance of the proper learning algorithm was emphasized because with advanced learning algorithm we can train these networks, which cannot be trained with simple algorithms. When simple training algorithms, such as EBP are used, neural networks with larger number of neurons must be used to fulfill the task and often generalization abilities of neural networks are lost.

REFERENCES

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

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

[3] 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.

[4] K. Levenberg, "A method for the solution of certain problems in least squares". *Quarterly of Applied Machematics*, 5, pp. 164-168, 1944.

[5] 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

[6] MATLAB Neural Network ToolBox http://www.mathworks.com/products/neuralnet/

[7] B. M. Wilamowski, N. J. Cotton, O. Kaynak, G. Dundar, "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, H. Yu, "Improved Computation for Levenberg Marquardt Training," *IEEE Trans. on Neural Networks*, vol. 21, no. 6, pp. 930-937, June 2010

[9] B. M. Wilamowski and H. Yu, "Neural Network Learning Without Backpropagation," *IEEE Trans. on Neural Networks*, vol. 21, no.11, pp. 1793 - 1803 Nov. 2010

[10] B. M. Wilamowski, " Neural Network Architectures and Learning algorithms", *IEEE Industrial Electronics Magazine,* vol 3, no 4, pp.56-63, (2009)

[11] Bogdan M. Wilamowski "Efficient Neural Network Architectures and Advanced Training Algorithms", *Gdańsk University of Technology Faculty of ETI Annals,* Vol 18, pp. 345-352, 2010

[12] B. 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, 200.

[13] B. M. Wilamowski, Hao Yu, and Kun Tao Chung "Parity-*N* Problems as a Vehicle to Compare Efficiency of Neural Network Architectures" *Industrial Electronics Handbook, vol. 5 – Intelligent Systems,* 2nd Edition, chapter 10, pp. 10-1 to 10-8, CRC Press 2011.

[14] B. M. Wilamowski, " Challenges in Applications of Computational Intelligence in Industrial Electronics" *ISIE10 - International Symposium on Industrial Electronics,* Bari, Italy, July 4-7, 2010, pp. 15-22.

[15] B. M. Wilamowski, N. J. Cotton, O. Kaynak,, and G. Dundar, "Method of computing gradient vector and Jacobean matrix in arbitrarily connected neural networks" *ISIE 2007- IEEE International Symposium on Industrial Electronics*, Vigo, Spain, 4-7 June 2007, pp. 3298-3303

[16] B. M. Wilamowski, N. Cotton, J. Hewlett, and O. Kaynak, "Neural Network Trainer with Second Order Learning Algorithms", 11th *INES 2007 -International Conference on Intelligent Engineering Systems,* Budapest, Hungary, June 29 2007-July 1 2007, pp. 127-132H.

[17] Yu and B. M. Wilamowski, "Fast and efficient and training of neural networks," in *Proc. 3nd IEEE Human System Interaction Conf. HSI 2010*, Rzeszow, Poland, May 13-15, 2010, pp. 175-181

[18] Jian-Xun Peng, Kang Li, G.W. Irwin, "A New Jacobian Matrix for Optimal Learning of Single-Layer Neural Networks," IEEE Trans. on Neural Networks, vol. 19, no. 1, pp. 119-129, Jan 2008.

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

[20] 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.

[21] NBN training software *http://www.eng.auburn.edu/~wilambm/nnt/NBNTrainer210.zip*