Neural Network Architectures and Learning Algorithms

How Not to Be Frustrated with Neural Networks

BOGDAN M. WILAMOWSKI

Neural networks are very powerful as nonlinear signal processors, but obtained results are often far from satisfactory. The purpose of this article is to evaluate the reasons for these frustrations and show how to make these neural networks successful. The following are the main challenges of neural network applications:

1) Which neural network architectures should be used?
2) How large should a neural network be?
3) Which learning algorithms are most suitable?

The multilayer perceptron (MLP) architecture (Figure 1) is unfortunately the preferred neural network topology of most researchers [1], [2]. It is the oldest neural network architecture, and it is compatible with all training softwares. However, it will be shown in the latter part of this article that MLP architectures seldom give positive results. The MLP topology is less powerful than other topologies such as bridged multilayer perceptron (BMLP), where connections across layers are allowed (marked as dotted lines in Figure 2).

Both MLP and BMLP architectures, as shown in Figures 1 and 2, have four layers, three input nodes, four neurons in the first hidden layer, three neurons in the second hidden layer, and one neuron in the output layer. Shorthand notation for

Digital Object Identifier 10.1109/MIE.2009.934790

1932-4529/09/$26.00©2009IEEE

these topologies are 3-4-3-1 and 3=4=3=1, where "=" characters replace "-" characters if the neural network has connections across layers. A comparison of several neural network architectures is given in the section "Comparison of Neural Architectures."

After neural network architecture is chosen, the next question is how large a neural network to be used. As will be demonstrated in the section "Use Minimum Network Size," it is much easier to secure training convergence of larger neural networks, but this success is often misleading, because neural networks with an excessive number of neurons do not have good interpolation abilities and cannot properly handle new patterns that were not used in the training process.

The error-back propagation (EBP) algorithm [3], [4] is the most popular learning algorithm, but it is very slow and seldom gives adequate results. The EBP training process requires 100–1,000 times more iterations than the more advanced algorithms such as Levenberg–Marquardt (LM) [5], [6] or neuron by neuron (NBN) [7], [8] algorithms. What is most important is that the EBP algorithm is not only slow but often it is not able to find solutions for close-to-optimum neural networks. The section "Case Study" describes and compares several learning algorithms.

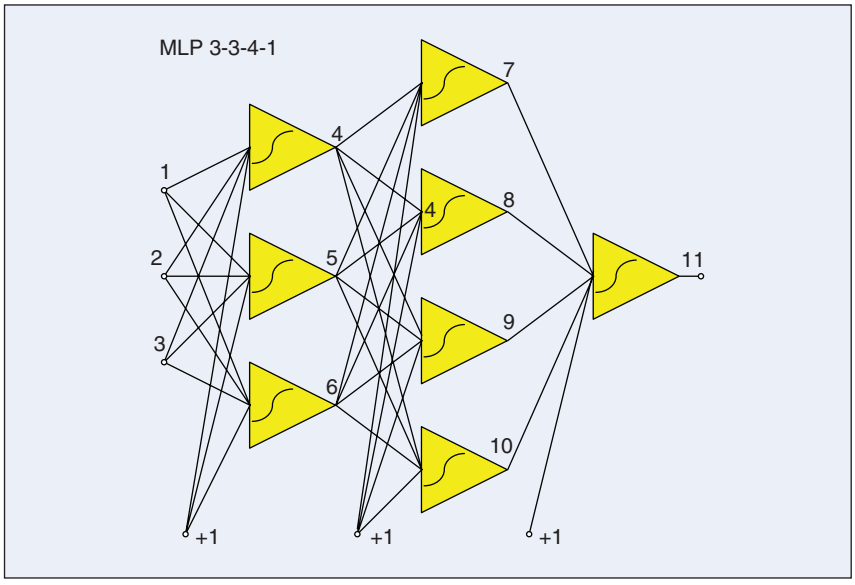

FIGURE 1 – The MLP-type architecture 3-3-4-1 (without connections across layers).

Comparison of Neural Architectures

There are several neural network architectures such as radial basis function (RBF), counterpropagation, or learning vector quantization (LVQ) networks [2], [9] that can be used for rapid prototyping. It is very easy to train them, but they require a large number of neurons (equal to the number of patterns or number of clusters). Also, in most cases, these architectures require additional signal-normalization processes. More recently, support vector machine (SVM) techniques [10] are often used to replace neural networks. In this presentation, we will focus on classical feed-forward neural networks with sigmoidal activation functions. In this traditional approach, neural network topologies/architectures are, in most cases, selected by a trial-and-error process. Often, success depends on a lucky guess; hence, the search process is started with larger architecture, and the network is pruned in a more or less organized way [11]. Unfortunately, most pruning algorithms are dealing with MLP architectures, and these architectures have limited abilities for neural signal processing. This section will show the advantages of other than MLP architectures.

The most common test bench for neural networks is the parity-*N* problem. Parity-*N* is considered to be the most difficult set of patterns for neural network training. The simplest parity-2 problem is also known as the exclusive-OR (XOR) problem. The larger the *N*, the more difficult it is to solve it. Even though parity-*N* problems are very complicated, it is possible to theoretically find neural network architectures and weight solutions [12]. Of course, depending on the neural network topology,

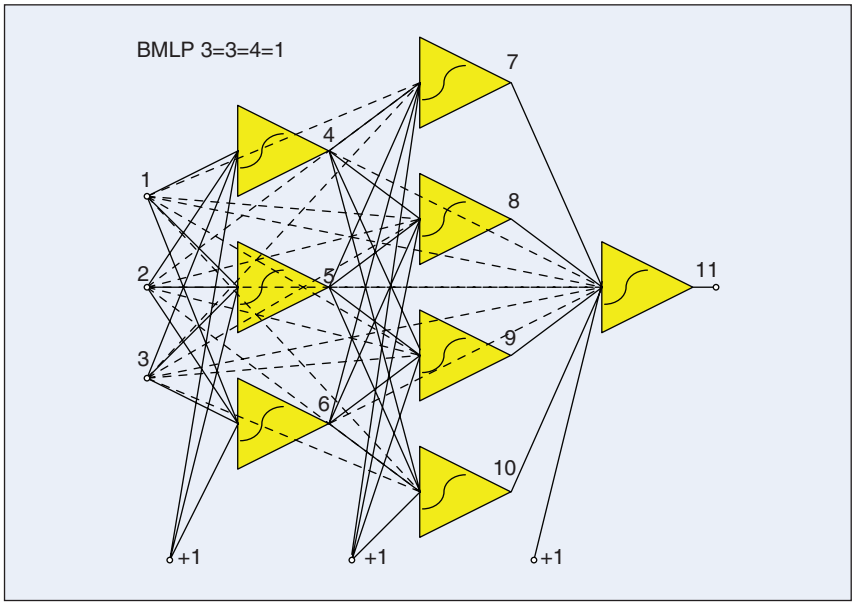

FIGURE 2 – The BMLP architecture 3=3=4=1 (with connections across layers marked by dotted lines).

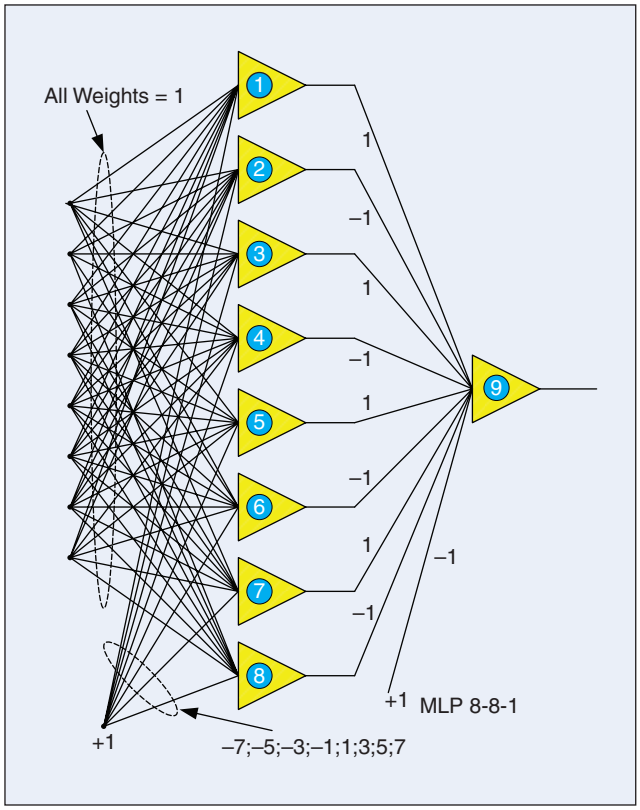

FIGURE 3 – Bipolar neural network for the parity-8 problem with MPL neural network with one hidden layer.

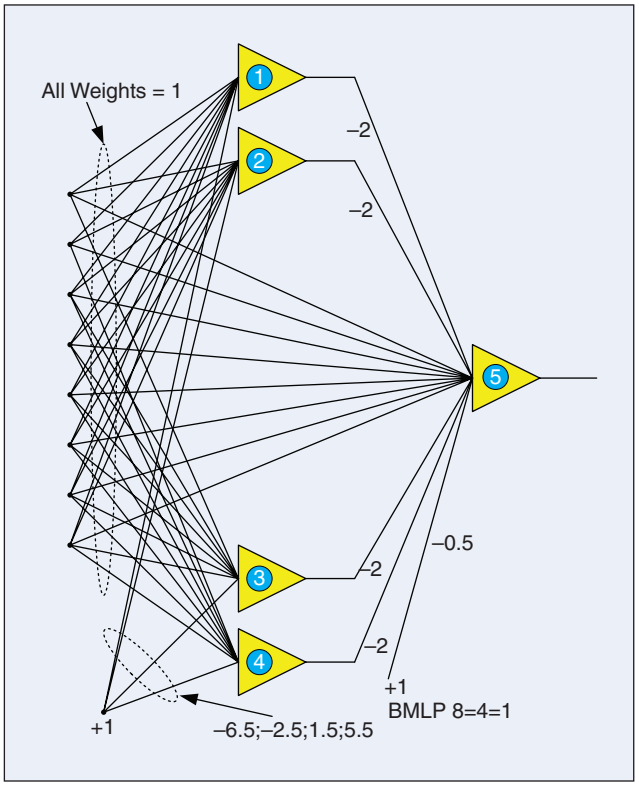

FIGURE 4 – Bipolar neural network for parity-8 problem with one hidden layer direct connections between output neuron and inputs of the network (BMLP).

different numbers of neurons and weights are required to solve the same problem using neural networks with bipolar activation functions. Figures 3–5 show several neural network topologies for the parity-8 problem. In the case of the most popular MLP architecture with one hidden layer (Figure 1), there are at least nine neurons required and $8 \times 9 + 9 = 81$ weights (Figure 3). For BMLP topology (Figure 2), where connections across layers are allowed, the same problem can be solved with only five neurons, and the total number of weights is $4 \times 9 + 8 + 4 + 1 = 49$ (Figure 4). For a fully connected cascade (FCC) architecture (Figure 6), only four neurons are required, and the total number of weights is $9 + 10 + 11 + 12 = 42$. For unipolar activation functions, neural network topologies would be identical, except that different values should be used for threshold-controlling weights [12].

If a larger problem is considered, such as, for example, a parity-17 problem, the MLP architecture needs 18 neurons, the BMLP architecture with connections across hidden layers needs nine neurons, and the FCC architecture needs only five neurons. The minimum number of neurons required for parity-N problems are given by (1)–(3). In these equations, nn is the minimum number of neurons, and nw is the number of weights.

For traditional MLP architectures (Figure 3),

$$nn = N + 1 \text{ and}$$
$$nw = nn^2 = (N+1)^2. \qquad (1)$$

For BMLP architectures with additional connection through hidden layer (Figure 4),

$$nn = \left\lceil \frac{N}{2} \right\rceil + 1 \quad \text{and}$$
$$nw = nn(N+2) - 1. \qquad (2)$$

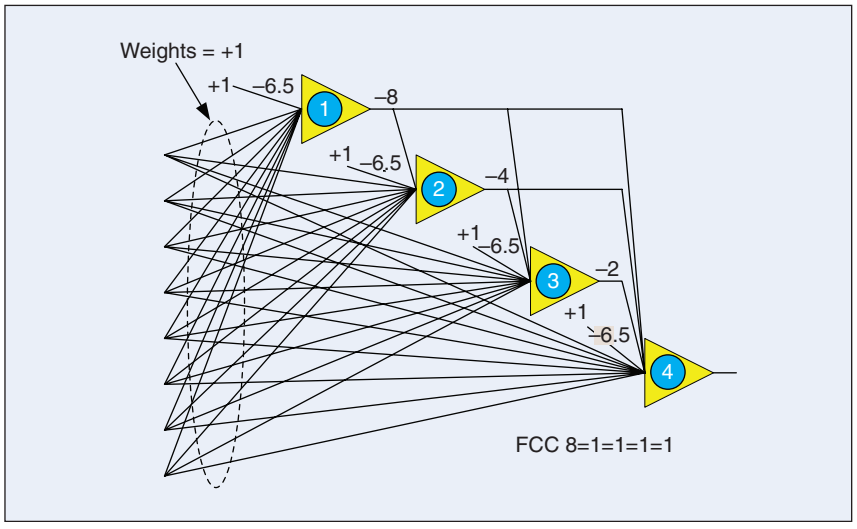

FIGURE 5 – Bipolar neural network for parity-8 problem in an FCC architecture.

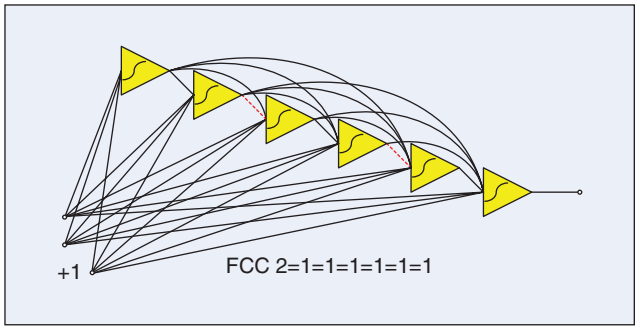

FIGURE 6 – An FCC topology with two inputs and six neurons.

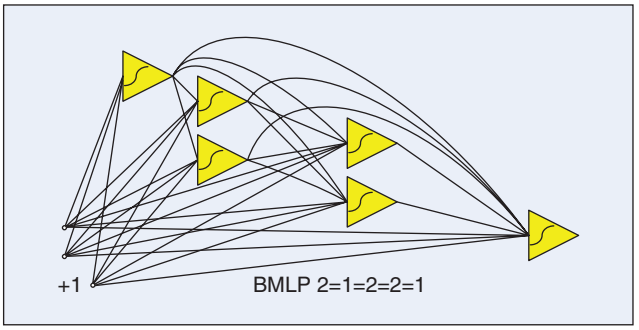

FIGURE 7 – The BMLP topology with two inputs and six neurons.

For FCC architectures (Figure 5),

$$nn = \lceil \log_2 (N+1) \rceil \quad \text{and}$$
$$nw = nn\left(N + 0.5 + \frac{nn}{2}\right). \qquad (3)$$

Table 1 shows the minimal number of neurons/weights required for different parity problems using various neural network architectures. One can easily conclude from Table 1 that the FCC architecture (Figure 5) can solve parity-*N* problems with the lowest number of neurons and weights.

There is another advantage of architectures with connections across layers (Figures 4 and 5). With these additional connections, neural networks are more transparent for signal propagation, and they are easier to train. In typical MLP architectures, the forward and backward propagating signals must pass more nonlinear elements (neurons) than in other network topologies. Unfortunately, it is much easier to write the training software for simple MLP architecture than for arbitrarily connected neural networks. For example, the very popular MATLAB Neural Network Toolbox [13] is not able to handle arbitrarily connected feed-forward neural network architectures. For more efficient neural network architectures, it is often difficult to find the training software. The exceptions are as follows:

- the Stuttgart Neural Network Simulator (SNNS) [14] that uses first-order algorithms such as EBP and its derivatives
- the NBN [8], [15] (where both first- and second-order learning methods are implemented).

The NBN algorithm [7] is an improved version of the LM algorithm [6], where a second-order algorithm is used for arbitrarily connected feed-forward neural networks. The network topology is entered to the system in a similar way as in the SPICE program. The node numbering is organized in the following way: First, node numbers are reserved for input nodes and then for output nodes of neurons in natural order in feed-forward direction. The last nodes are associated with network outputs. Each line has a node number of a neuron, name of the model of activation function, and the list of all input nodes. Table 2 shows the topology files for the networks of Figures 1 and 2. Both networks use the bipolar activation function with a gain of three and the same data training set in trainingset.dat. file. The training set consists of only numerical data, with the number of rows equal to the number of patterns, and the first columns are associated with inputs and the remaining columns are for outputs. The node number of the first neuron *n*1 indicates that the number of inputs in the training set is (*n*1 – 1). More detailed instructions can be found in [8] and [15]. Table 2 shows the sample topology files.

One may notice that if the number of neurons is held constant and all

TABLE 1–NUMBER OF NEURONS/WEIGHTS REQUIRED FOR DIFFERENT PARITY PROBLEMS USING NEURAL NETWORK ARCHITECTURES.

ARCHITECTURE	PARITY-3	PARITY-7	PARITY-15	PARITY-31	PARITY-63
MLP	**4**/16	**8**/64	**16**/256	**32**/1024	**64**/4096
BMLP	**3**/14	**5**/44	**9**/152	**17**/560	**33**/2144
FCC	**2**/9	**3**/27	**4**/70	**5**/170	**6**/399

TABLE 2–TOPOLOGY FILES OF NEURAL NETWORKS OF FIGURES 1 AND 2.

/TOPOLOGY OF FIGURE 1	//TOPOLOGY OF FIGURE 2
*n*4 mbip 1 2 3	*n*4 mbip 1 2 3
*n*5 mbip 1 2 3	*n*5 mbip 1 2 3
*n*6 mbip 1 2 3	*n*6 mbip 1 2 3
*n*7 mbip 4 5 6	*n*7 mbip 1 2 3 4 5 6
*n*8 mbip 4 5 6	*n*8 mbip 1 2 3 4 5 6
*n*9 mbip 4 5 6	*n*9 mbip 1 2 3 4 5 6
*n*10 mbip 4 5 6	*n*10 mbip 1 2 3 4 5 6
*n*11 mbip 7 8 9 10	*n*11 mbip 1 2 3 4 5 6 7 8 9 10
.model mbip fun=bip, gain=3	.model mbip fun=bip, gain=3
datafile=trainingset.dat	datafile=trainingset.dat

For optimum performance, neural networks should have as few neurons as possible.

possible feed-forward connections are implemented that there is only one possible cascade topology. An example is the FCC network with two inputs and six neurons, as shown in Figure 6. An abbreviated description of FCC architecture, shown in Figure 6, is 2=1=1=1=1=1=1, which indicates two inputs and six neuron layers with one neuron in each layer. Another benefit of the FCC topology is that it is relatively easy to find an optimal size for the neural network, without searching through large number of possibilities given by MLP or BMLP topologies. For a limited number of neurons, FCC neural networks are the most powerful architectures, but this does not mean that they are the only suitable architectures. Often, similar results can be obtained with slightly simplified architectures, as in removing some weights from FCC networks. For example, if the two weights marked by red dotted lines in Figure 6 are removed, then the FCC 2=1=1=1=1=1=1 architecture is converted to a BMLP 2=1=2=2=1 architecture, as shown in Figure 7. This BMLP architecture (Figure 7) will be only slightly less powerful than the FCC architecture (Figure 6), but it has other significant advantages. The signal has to be propagated by fewer layers, and as a result, the network is more transparent for training. The traditional MPL topology does not have any advantages other than that it is easier to find the training software.

Use Minimum Network Size

It is not enough to develop a neural network so that it properly responds to all training patterns. The main purpose of practical usage of neural networks is to be able to receive a close-to-optimum answer for all patterns that were never used in training. Therefore, to verify the quality of the developed neural networks, different patterns are used for training and verification. If errors obtained with verification patterns are satisfactory, then the neural network architecture is acceptable. In the case of having only a limited number of patterns to check the suitability of neural network size and architecture, it is a relatively tedious process. In the first step, all but one pattern is used for training, and then the error for the pattern that was not used for training is evaluated. This process is repeated until all patterns are excluded from training, and their errors are evaluated.

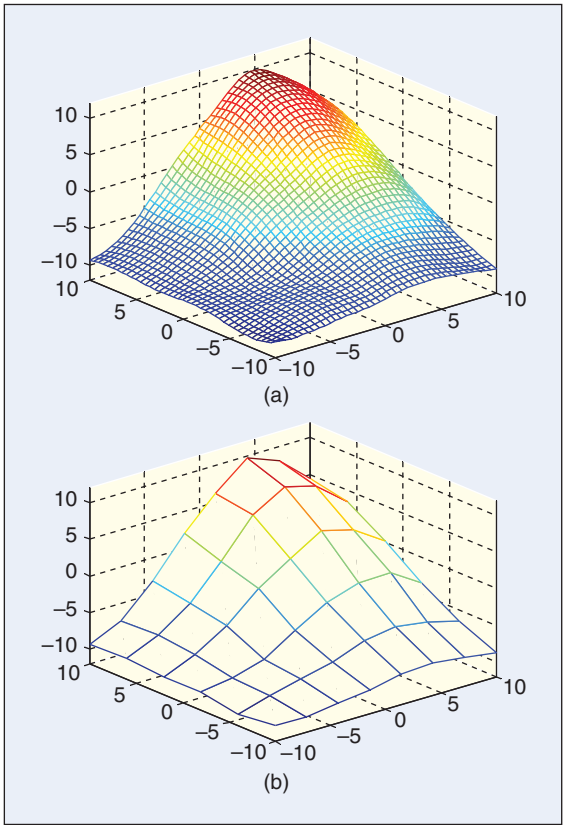

FIGURE 8 – Control surface of the TSK fuzzy controller: (a) required control surface; (b) 8*6 = 48 defuzzification rules.

The method of selecting the best architecture by removing one training instance at a time is very time consuming, especially if many neural network architectures must be tried and efficient training algorithms are not used. Many researchers are often frustrated when a neural network can be trained well on the training patterns and then perform poorly on verification patterns. It means that the neural network lost its generalization abilities.

The major hint is that with a smaller number of neurons, the neural network should have better generalization abilities. If too many neurons are used, then the network can be overtrained 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 results for new patterns. The training error for neural networks is often defined as mean square error (MSE)

$$\text{Error} = \frac{1}{N}\sum_{i=1}^{N}(d_i - o_i)^2, \quad (4)$$

where N is the number of patterns and d_i and o_i are the desired and actual output for the ith pattern.

As it was discussed in the section "Comparison of Neural Architectures," the FCC topology (Figure 6) seems to be the most powerful and, for a given number of neurons, there is always a unique, precisely defined, FCC architecture. In the case of other neural network architectures, such as MLP, there are always many possible topologies with which to experiment. With FCC architectures, the choices are limited, and the best neural network structure can be found very quickly. The only question then is how many neurons have to be used to achieve the best results and, typically, only two to four trials are enough to find the best solution.

Case Study

Let us try to find the best neural network architecture to replace a fuzzy controller. Figure 8 shows the required control surface and the defuzzification rules for the Takagi, Sugeno, Ken (TSK) fuzzy controller [16], [17]. Figure 9 shows the control surface obtained with TSK fuzzy controllers using trapezoidal and triangular membership functions.

To train the developed neural controller, we may use the TSK defuzzification rules as the training patterns. Let us select an FCC neural network architecture and try to find solutions for using different number of neurons. Figure 10(a) shows the results of a neural network with three neurons (12 weights), and Figure 10(b) shows the results with four neurons and 18 weights. However, when the size of the network increases (Figure 11), the results become worse instead of better, even though learning errors decrease with the increase of neural network size.

One may notice that the best results were obtained with a four-neuron architecture [Figure 10(b)]. With more neurons, we are obviously able to reduce the training error, but the neural network loses its generalization ability. One may notice that for all training patterns [Figure 8(b)] a very small error was obtained (1.1 E−05), but between training points, the eight-neuron cascade architecture produces undesirable results, as one can see on Figure 11(b). It is less noticeable, but even with a five-neuron architecture [Figure 11(a)], results are not as good as with the four-neuron architecture [Figure 10(b)]. The conclusion is that for optimum performance, neural networks should have as few neurons as possible.

Which Learning Algorithms Should Be Used?

Neural network training software can be found and used with little effort. What many people are not aware of is that not all popular algorithms can train every neural network. Surprisingly, the most popular EBP algorithm [3], [4] cannot handle more complex problems, while other more advanced algorithms [6], [7] can.

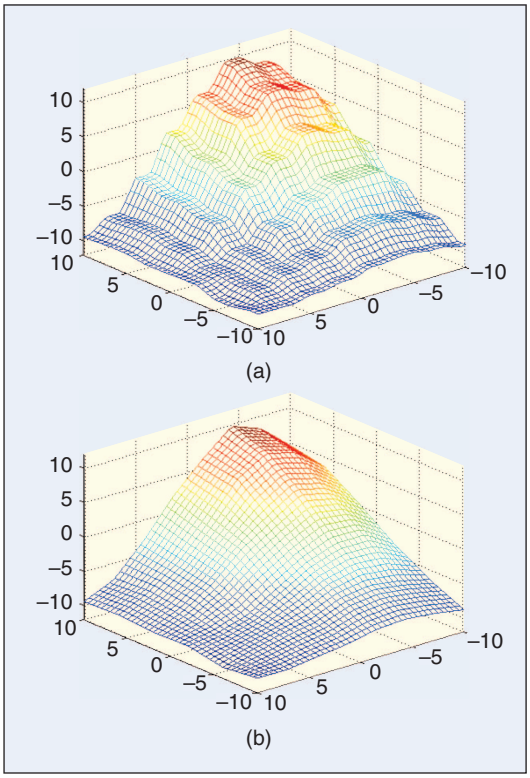

FIGURE 9 – Control surface of the TSK fuzzy controller with equally spaced membership function 8 in x direction and 6 in y direction: (a) trapezoidal membership functions, (b) triangular membership functions.

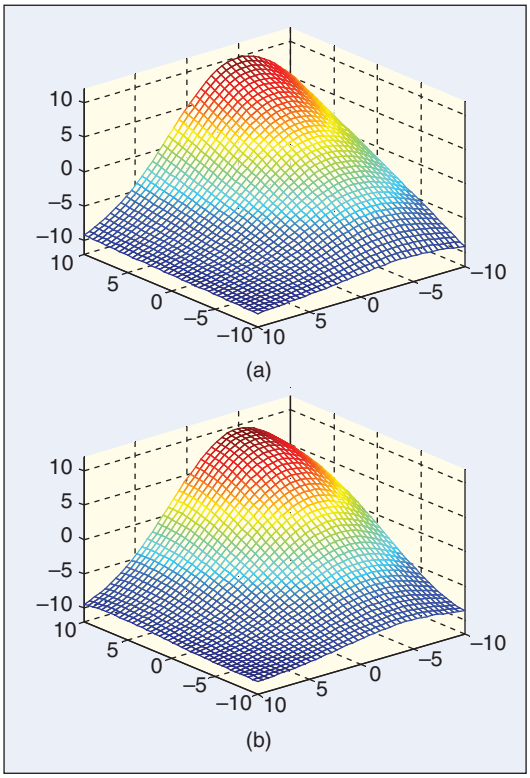

FIGURE 10 – Control surface obtained with neural networks: (a) three neurons in cascade (12 weights) training error = 0.21049; (b) four neurons in cascade (18 weights) training error = 0.049061.

Let us use the parity-3 problem with a simple two-neuron FCC architecture to illustrate the properties of first- and second-order algorithms. Figure 12 shows the training error as a function of the number of iterations. One may notice the asymptotical character of EBP [Figure 12(a)], which may not let the process converge to very small errors. The NBN algorithm can train neural networks 1,000 times faster than the EBP algorithm. With large neural networks, the advantages of NBN algorithm diminish, because for every iteration, it has to invert the square matrices of size equal to the number of weights. The practical limit for NBN or LM algorithms on PC computers is about 500 weights in the network.

One of the most difficult problems for neural networks, besides parity-N problems, is the Wieland two-spiral problem, where two interlacing spirals have to be separated. The two-spiral problem has an advantage, because it can be easily visualized. Using the cascade correlation algorithm/architecture, this problem can be solved by the FCC topology, using 16–20 neurons [18]. When the recently developed NBN algorithm [7], [8] is used, the same problem can be solved with as little as eight neurons and 52 weights [8] (Figure 13). The NBN algorithm can easily handle feed-forward neural networks with arbitrarily connected neurons [19], which was not possible with the originally developed LM algorithm [6], [11]. Note that, using the popular EBP algorithm, with the same FCC topology and 12 neurons [twice as many weights (102) is required to solve the

same two-spiral problem (Figure 14)]. The processing overhead to solve the two-spiral problem with EBP is about 300,000 iterations in about 6 min (Figure 14). There are, of course, countless numbers of improvements to the EBP algorithm, such as momentum [20], resilient error-back propagation (RPROP) algorithm [21], adaptive learning rate [22] and as long as the first-order gradient method is used, these improvements are not dramatic. In comparison, the NBN algorithm reached the solution with fewer than 300 iterations and fewer than 1 s (Figure 13).

One may draw the conclusion that advanced algorithms such as the NBN can not only find a solution more than 100–1,000 times faster but also solve problems for which the EBP algorithm is not very useful. Interestingly, not all learning algorithms are able to train neural networks with minimal number of neurons. Please note

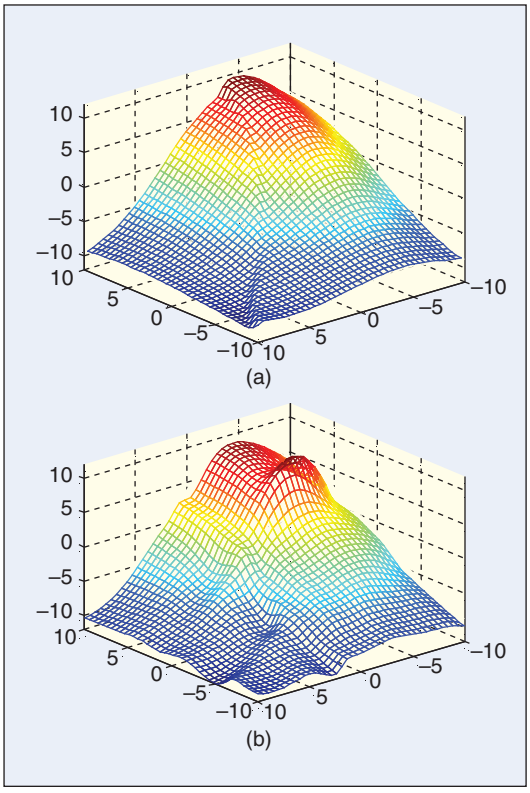

FIGURE 11 – Control surface obtained with neural networks: (a) five neurons in cascade (25 weights) training error = 0.023973; (b) eight neurons in cascade (52 weights) training error = 1.118 E–005.

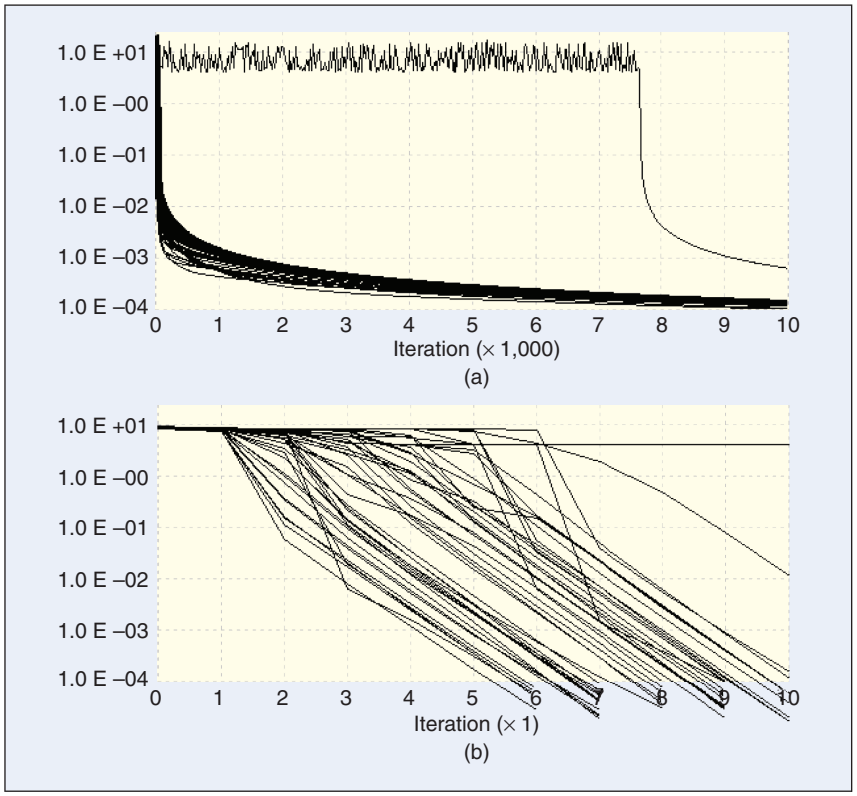

FIGURE 12 – Training error as the function of number of iterations, using ten trials to the desired error of 10^{-4}: (a) EBP algorithm (1% success rate, average solution time of 4.2 s, and average 4,188.3 iterations); (b) NBN algorithm (98% success rate, average solution time of 2.4 ms, and average 5.73 iterations).

that the popular EBP algorithm was unable to find a solution for smaller than the 12-neuron network, while the NBN was able to train the two-spiral problem with as few as eight neurons. The conclusion is that with a better learning algorithm the same problem can be solved with much smaller neural networks, and as discussed in the sections "Use Minimum Network Size" and "Case Study" for not losing generalization abilities, the neural network should be as small as possible.

To be successful in the development of a good neural network, one has to follow several golden rules:

1) When possible, use neural network architectures with connections across layers, such as FCC or BMLP architectures. Such networks are not only more powerful but also easier to train (assuming that proper training software is used).
2) To prevent overtraining, try to use networks with a minimum number of neurons. The problem is that for these reduced networks an advanced learning algorithm must be used, as first-order algorithms may not have the ablility to train them.
3) The EBP is not only very slow, but it may have the ability to find an optimal solution for the architecture with a reduced number of neurons.
4) Second-order algorithms such as LM and NBN have difficulties handling very large neural networks, because at each iteration, they have to invert a $nw \times nw$ matrix, where nw is the number of weights. From a practical viewpoint, this is not a significant limitation, as to be successful, the smallest possible neural networks should be used anyway. When the NBN algorithm is used, then 500 weights would be a practical upper limit in current Windows-based computers.
5) Finally, the powerful second-order LM algorithm adopted in the

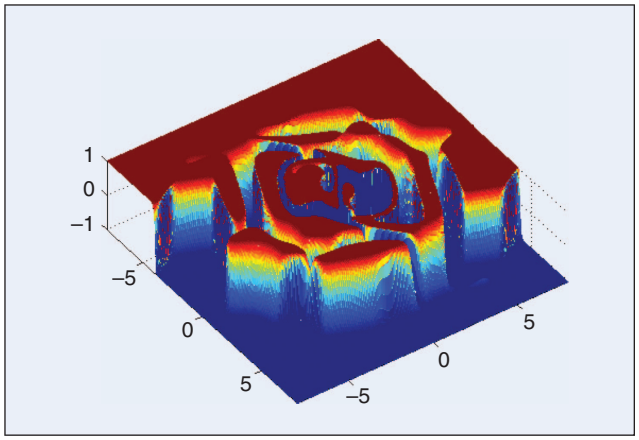

FIGURE 13 – Solution of the two-spiral problem with NBN algorithm [4] using FCC architecture with eight neurons and 52 weights. To reach the solution, 244 iterations and 0.913 s were required.

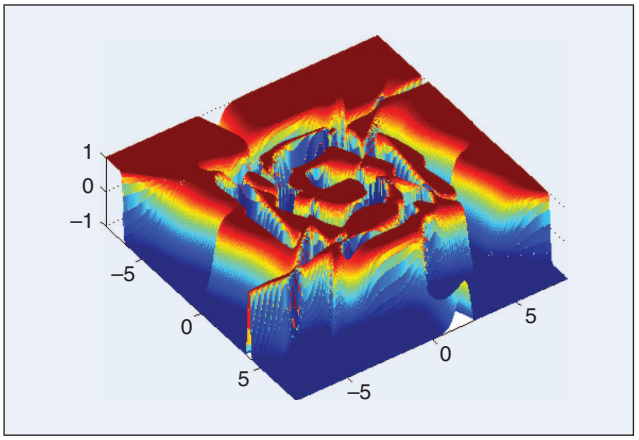

FIGURE 14 – Solution of the two-spiral problem with EBP algorithm using an FCC architecture with 16 neurons and 168 weights. To reach the solution, 308,325 iterations and 342.7 s were required.

popular MATLAB Neural Network Toolbox [13] can handle only MLP topologies, without connections across layers, and these topologies are far from optimal. The NBN algorithm does not have this limitation, and very fast C++ version can be downloaded from [15].

The importance of the proper learning algorithm was emphasized, because with advanced learning algorithms, we can train those networks that cannot be trained with simple algorithms. When simple training algorithms such as EBP are used, neural networks with a larger number of neurons must be used to fulfill the task. As a consequence, an EBP algorithm neural network learns the training patterns, but it loses the generalization abilities. In other words, the neural network may give incorrect answers for patterns that were not used in the training set.

Biography

Bogdan M. Wilamowski (wilam@ieee.org) received his M.S., Ph.D., and D.Sc. degrees in 1966, 1970, 1977, respectively. He was with the Technical University of Gdansk, Poland, University of Wyoming, and University of Idaho. Since 2003, he has been professor and director of the Alabama Microelectronics Science and Technology Center at Auburn University. He also works for WSliZ, Rzeszów, Poland. He is the author of four textbooks, more than 300 refereed publications, and holds 27 patents. He has been involved in the neural networks research area since 1966. He was the cofounder of the IEEE Neural Networks Society and IEEE Computational Intelligence Society. He was associate editor for *IEEE Transactions on Neural Networks*, *Journal of Intelligent and Fuzzy Systems*, and *Journal of Computing*. He is currently the editor-in-chief of *IEEE Transactions on Industrial Electronics*. He is a Fellow of the IEEE.

References

[1] B. K. Bose, "Neural network applications in power electronics and motor drives—An introduction and perspective," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 14–33, Feb. 2007.

[2] B. M. Wilamowski, "Neural networks and fuzzy systems for nonlinear applications," in *Proc. 11th INES 2007–11th Int. Conf. Intelligent Engineering Systems*, Budapest, Hungary, June 29–July 1, 2007, pp. 13–19.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Wiliams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 9, 1986.

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

[5] K. Levenberg, "A method for the solution of certain problems in least squares," *Quart. Appl. Math.*, vol. 2, pp. 164–168, 1944.

[6] M. T. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[7] B. M. Wilamowski, N. J. Cotton, O. Kaynak, and G. Dundar, "Computing gradient vector and Jacobian matrix in arbitrarily connected neural networks," *IEEE Trans. Ind. Electron.*, vol. 55, no. 10, pp. 3784–3790, Oct. 2008.

[8] H. Yu and B. M. Wilamowski, "Efficient and reliable training of neural networks," in *Proc. 2nd Conf. Human System Interaction*, Catania, Italy, May 21–23, 2009, pp. 109–115.

[9] B. M. Wilamowski, "Special neural network architectures for easy electronic implementations," in *Proc. Int. Conf. Power Engineering, Energy and Electrical Drives 2009*, Lisbon, Portugal, Mar. 18–20, 2009, pp. 17–22.

[10] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[11] N. Fanieh, F. Fanieh, B. W. Jervis, and M. Cheriet, "The combined statistical stepwise and iterative neural network Pruning algorithm," *Intell. Automat. Soft Comput.*, vol. 15, no. 4, pp. 573–589, 2009.

[12] B. Wilamowski, D. Hunter, and A. Malinowski, "Solving parity-*n* problems with feedforward neural network," in *Proc. IJCNN'03 Int. Joint Conf. Neural Networks*, Portland, OR, July 20–23, 2003, pp. 2546–2551.

[13] MATLAB Neural Network Toolbox [Online]. Available: http://www.mathworks.com/products/neuralnet/

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

[15] NNT—Neural Network Trainer [Online]. Available: http://www.eng.auburn.edu/~wilambm/nnt/

[16] M. Sugeno and G. T. Kang, "Structure identification of fuzzy model," *Fuzzy Sets Syst.*, vol. 28, no. 1, pp. 15–33, 1988.

[17] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modeling and control," *IEEE Trans. Syst., Man, Cybern.*, vol. 15, no. 1, pp. 116–132, 1985.

[18] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 524–532.

[19] H. Yu and B. M. Wilamowski, "Efficient and reliable training of neural networks," in *Proc. 2nd IEEE Human System Interaction Conf., HSI 2009*, Catania, Italy, May 21–23, 2009, pp. 109–115.

[20] V. V. Phansalkar and P. S. Sastry, "Analysis of the back-propagation algorithm with momentum," *IEEE Trans. Neural Networks*, vol. 5, no. 3, pp. 505–506, Mar. 1994.

[21] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. Int. Conf. Neural Networks*, San Francisco, CA, 1993, pp. 586–591.

[22] C.-T. Kim and J.-J. Lee, "Training two-layered feedforward networks with variable projection method," *IEEE Trans. Neural Networks*, vol. 19, no. 2, pp. 371–375, Feb. 2008.