Selection of Proper Neural Network Sizes and Architectures—A Comparative Study

David Hunter, Hao Yu, *Student Member, IEEE*, Michael S. Pukish, III, *Student Member, IEEE*, Janusz Kolbusz, and Bogdan M. Wilamowski, *Fellow, IEEE*

***Abstract*—One of the major difficulties facing researchers using neural networks is the selection of the proper size and topology of the networks. The problem is even more complex because often when the neural network is trained to very small errors, it may not respond properly for patterns not used in the training process. A partial solution proposed to this problem is to use the least possible number of neurons along with a large number of training patterns.**

The discussion consists of three main parts: first, different learning algorithms, including the Error Back Propagation (EBP) algorithm, the Levenberg Marquardt (LM) algorithm, and the recently developed Neuron-by-Neuron (NBN) algorithm, are discussed and compared based on several benchmark problems; second, the efficiency of different network topologies, including traditional Multilayer Perceptron (MLP) networks, Bridged Multilayer Perceptron (BMLP) networks, and Fully Connected Cascade (FCC) networks, are evaluated by both theoretical analysis and experimental results; third, the generalization issue is discussed to illustrate the importance of choosing the proper size of neural networks.

***Index Terms*—Architectures, learning algorithms, neural networks, topologies.**

I. Introduction

NEURAL networks are currently used in many daily life applications. In 2007, a special issue of TIE was published only on their application in industrial practice [1]. Further applications of neural networks continue in various areas. Neural networks are used for controlling induction motors [2]–[4] and permanent magnet motors [5]–[7], including stepper motors [8]. They are also used in controlling other dynamic systems [9]–[16], including robotics [17], motion control [18], and harmonic distortion [19], [20]. They are also used in industrial job-shop-scheduling [21], networking [22], and battery control [23]. Neural networks can also easily simulate highly complex dynamic systems such as oil wells [24] which are described by more than 25 nonlinear differential equations. Neural networks have already been successfully implemented in embedded hardware [25]–[29].

People who are trying to use neural networks in their research are facing several questions.

Manuscript received October 02, 2011; revised January 23, 2012; accepted January 31, 2012. Date of publication February 14, 2012; date of current version April 11, 2012. Paper no. TII-12-0052.

D. Hunter, H. Yu, M. S. Pukish, and B. M. Wilamowski are with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849-5201 USA (e-mail: dshunter1996@cableone.net; hzy0004@tigermail.auburn.edu; msp0005@tigermail.auburn.edu; wilam@ieee.org).

J. Kolbusz is with the Department of Distributed Systems, University of Information Technology and Management, Rzeszow 35-225, Poland (e-mail: jkolbusz@wsiz.rzeszow.pl).

Digital Object Identifier 10.1109/TII.2012.2187914

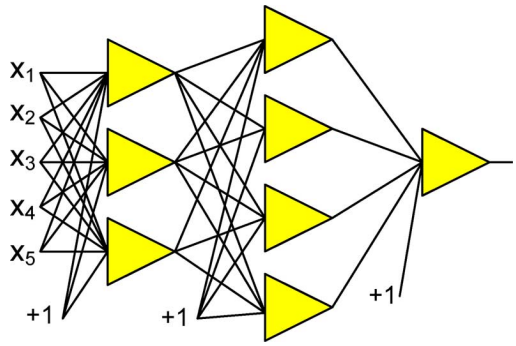

Fig. 1. Eight neurons in 5-3-4-1 MLP network.

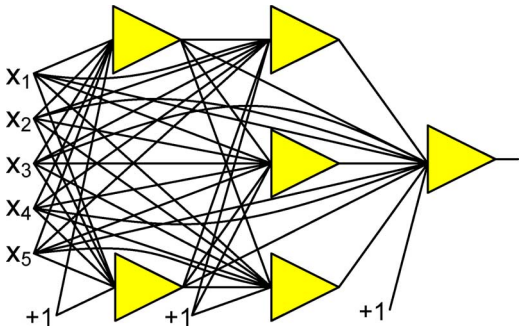

Fig. 2. Six neurons in $5 = 2 = 3 = 1$ BMLP network.

1) What neural network architecture should be used?
2) How many neurons should be used?
3) How many patterns should be used in training?
4) What training algorithm should be used?

Unfortunately, there are no easy answers to these questions. Various neural network architectures are shown in Figs. 1, –3. The most popular, Multilayer Perceptron (MLP) architecture is shown in Fig. 1. The improved version with connections across layers the Bridged Multilayer Perceptron (BMLP) is shown in Fig. 2. The most powerful the Fully Connected Cascade (FCC) architecture is shown in Fig. 3.

A detailed comparison of learning algorithms is given in Section II. Section III presents a theoretical synthesis of various neural network architectures such as MLP networks (Fig. 1), BMLP networks (Fig. 2), and FCC networks (Fig. 3) to solve various, complex Parity-N problems. At the end of the section, a comparison of these topologies is given in Fig. 17 and Table III.

Section IV presents an experimental comparison of the efficiencies of different topologies for various problems. Generalization issues are discussed in Section V. This is a very important but frequently neglected issue. It turns out that because

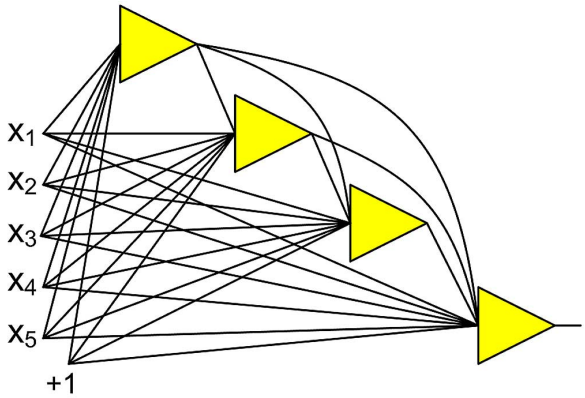

Fig. 3. Four neurons in $5 = 1 = 1 = 1 = 1$ FCC network.

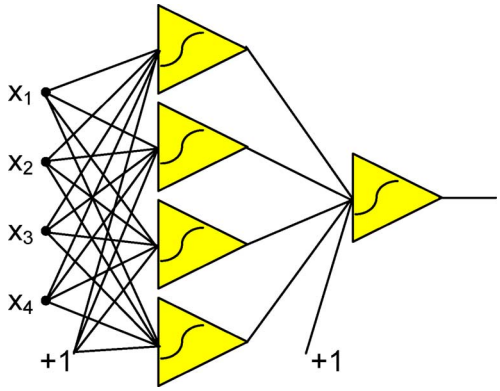

Fig. 4. Five neurons in $4 - 4 - 1$ MLP network.

it is easier to train neural networks with an excessive number of neurons, many researches are being lured into training large networks to very small errors, only to discover that these networks respond very poorly to patterns never used in the training process. Section VI presents a brief introduction on other networks, including radial basis function networks and deep neural networks. The final section presents a conclusion.

II. Learning Algorithms

Many algorithms for training neural networks have already been developed. The Error Back Propagation (EBP) [30], [31] algorithm is the most popular algorithm, but is very inefficient. Since development of the EBP algorithm, many improvements have been made to this algorithm [32]–[34]. Some of them are momentum [35], stochastic learning rate [36], [69], flat-spot elimination [37], [70], RPROP [38] and QUICKPROP [38]. They can reduce learning errors to small values, but often it does not mean that such train network has good generalization abilities.

Another drawback of the EBP algorithm and its modifications is that the search process follows just the gradient and can be trapped in local minima. Much better results can be obtained using second-order methods where a Hessian matrix is applied to evaluate the change of gradient. In this way, information about the shape of the error surface is used in the search process. Several second-order methods were adopted for neural network training, but the most efficient was the Levenberg Marquardt (LM) algorithm [40], [41]. The LM algorithm is very fast and efficient, but it has two major problems: 1) it was written [41] in such way that only MLP network architectures (Fig. 1) can be trained and 2) only relatively small problems with a limited number of patterns can be solved using this algorithm.

The training problem becomes even more complex because most neural network software can only train MLP architectures. For example, the popular MATLAB Neural Network Toolbox has both first- and second-order training algorithms, but it can only train MLP networks. As a consequence, without proper training software, researchers have no other option but to use MLP architectures and the obtained results are far from satisfactory [42], [43]. Both of these problems were solved in the recently developed Neuron by Neuron (NBN) training algorithm [44]–[47]. Although the NBN algorithm is not perfect, it can successfully compete with other algorithms in almost all cases, often producing improved results for the same parametric problems. It will be shown in this section that EBP is not only slow, but has difficulty finding solutions for close to optimum architectures. Let us compare the power of these algorithms with several experiments.

A. Parity-4 Problem

In this experiment, the EBP, LM, and NBN algorithms are compared. Each algorithm will be used to train the Parity-4 patterns using the MLP architecture shown in Fig. 4.

When testing the EBP algorithm, the learning constant is set to 1. In order to improve performance, the momentum technique was introduced with the value of 0.5. The maximum number of iterations is set to 100 000 for the EBP algorithm, and is set to 100 for both the LM and NBN algorithms. The Sum of Square Error (SSE) is calculated as a method to evaluate the training process and it is set to 0.001. For each algorithm, the training process is repeated 100 times with randomly generated initial weights in the range $[-1, 1]$.

Fig. 5 presents the training curves of the three different algorithms and Table I presents the comparison result. One can notice that, for the Parity-4 problem using the given MLP architecture, LM and NBN gives similar results, while EBP requires 500 times more iterations and learning time is about 200 times longer. Even improved by momentum, the success rate of the EBP algorithm is not as good as both the LM and NBN algorithms.

B. Two-Spiral Problem

One of the key benchmarks to evaluate neural network training and efficiency of architecture is the two-spiral problem shown in the Fig. 6. Separation of two groups of twisted points (green stars and red circles) by neural networks is not an easy task. Using a BMLP architecture (see Fig. 2), three hidden layers had to be used in a $2 = 5 = 5 = 5 = 1$ topology, requiring at least 16 neurons [48]. Using a standard MLP network with one hidden layer there are 34 neurons required to separate the two spirals [49]. The best results for the two-spiral problem are obtained by using the fully connected cascade (FCC) network. The well-known cascade correlation algorithm requires 12–19 hidden neurons in FCC networks [50].

In this experiment, the EBP and NBN algorithms are used to solve the two-spiral problem with a FCC network. The LM algorithm is not considered because it cannot handle FCC and

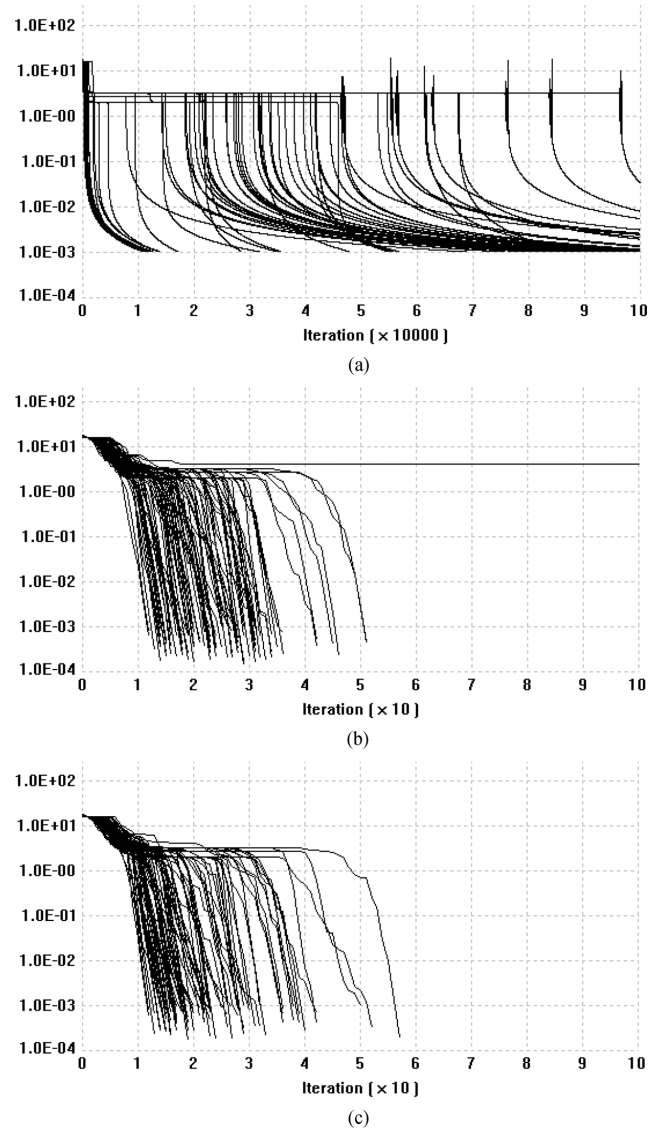

Fig. 5. Training curves of different algorithms for Parity-4 problem. (a) EBP algorithm. (b) LM algorithm. (c) NBN algorithm.

TABLE I
COMPARISON RESULTS OF PARITY-4 PROBLEM

Algorithms	Success Rate	Number of Iterations	Training Times (ms)
EBP	68%	12,036.01	5348.52
LM	100%	23.41	26.64
NBN	100%	23.21	25.64

BMLP networks. For the EBP algorithm, the learning constant is set to 0.005 and momentum is set to 0.5. The maximum number of iterations is set to 1 000 000 for the EBP algorithm and 1000 for the NBN algorithm. The desired training SSE is 0.01. For each case, the test is repeated 100 times with randomly selected initial weights in the range $[-1, 1]$.

As shown in Fig. 7, the training algorithm chosen affects the number of neurons required for successful training. Notice that the NBN algorithm can solve the two-spiral problem using 7 neurons and 44 weights [see Figs. 7(a) and 32(b)], while the EBP algorithm needs at least 12 neurons (102 weights). It takes only a few seconds to solve this problem with NBN while EBP needs about 10 min.

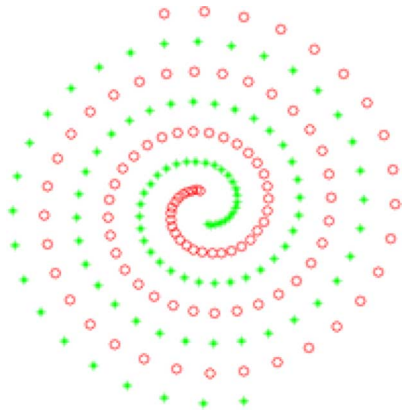

Fig. 6. Two spirals problem.

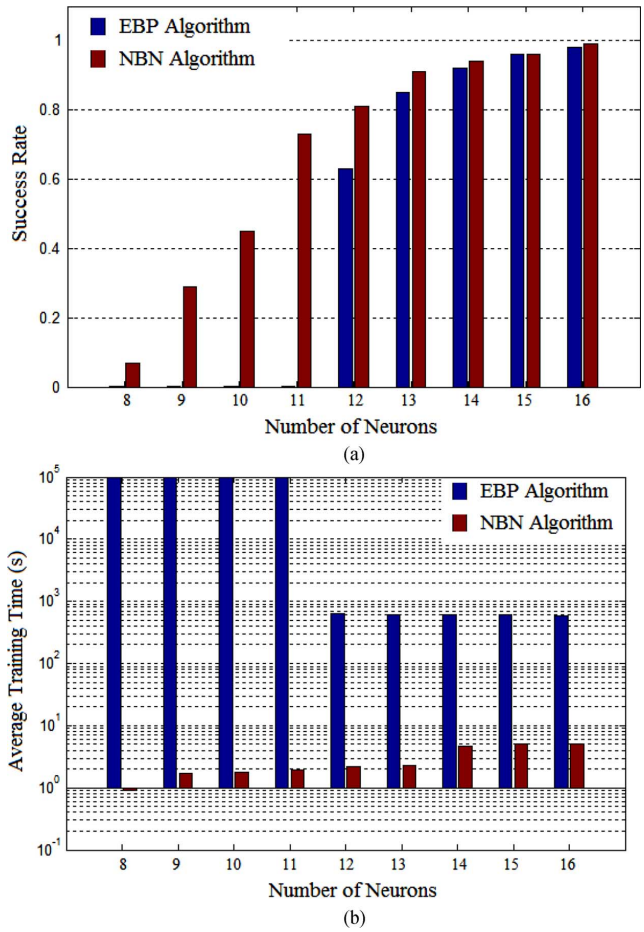

Fig. 7. Comparison results of the EBP and NBN algorithms for the two-spiral problem, using different number of neurons in fully connected cascade networks. (a) Success rate. (b) Training time.

One can also conclude that the EBP algorithm is only successful if an excessive number of neurons are used. Unfortunately, using large sized neural networks may cause an over-fitting problem. This issue will be illustrated with the example below and will be discussed more in Section V.

C. The Peak Problem

Let us consider the MATLAB peak surface [51] as the required surface and use equally spaced $10 \times 10 = 100$ patterns

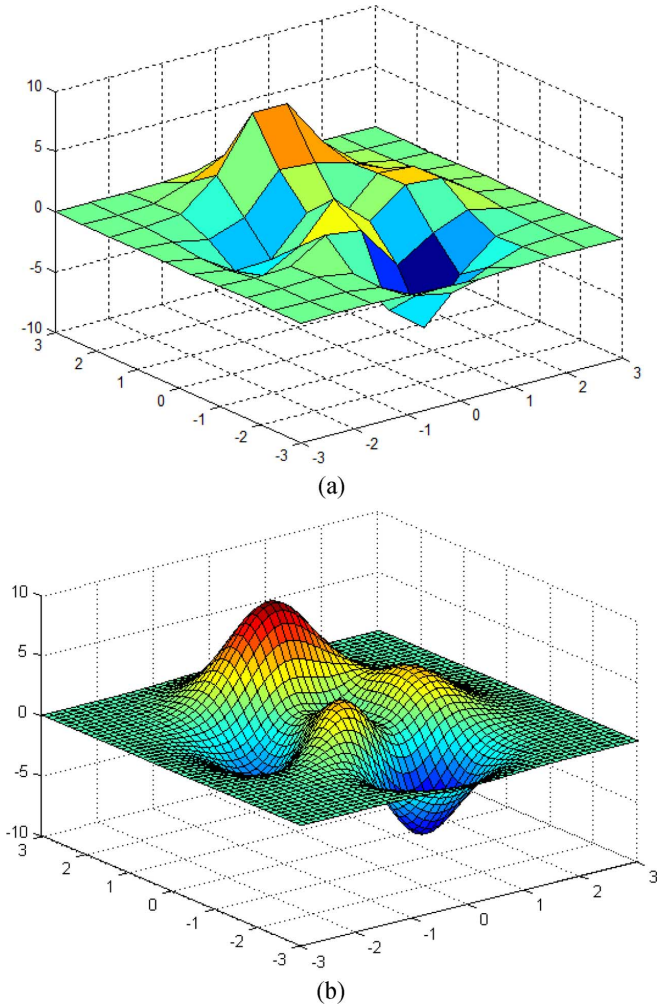

Fig. 8. Surface approximation using neural networks. (a) Training surface: $10 \times 10 = 100$ points. (b) Testing surface: $50 \times 50 = 2500$ points.

[Fig. 8(a)] to train neural networks. The quality of trained networks is evaluated using the errors computed for equally spaced $50 \times 50 = 2500$ patterns [Fig. 8(b)] in the same range. In order to make a valid comparison between training and verification error, the sum squared error (SSE) is divided by 100 and 2500, respectively.

Again, FCC networks are applied in the experiment and both EBP algorithm and NBN algorithm are selected for training. LM algorithm is not considered because of the architecture limitation. For EBP algorithm, learning constant is 0.0005 and momentum is 0.5. The maximum iteration is 1 000 000 for EBP algorithm and 1000 for LM algorithm. The desired training SSE is set as 0.5. There are 100 trials for each case. The training results are presented in Table II.

One may notice that, using the NBN algorithm, it was possible to find the acceptable solution (Fig. 9) with eight neurons (52 weights). Using the same network architecture with the EBP algorithm, the training process cannot converge to desired error level within 1 000 000 iterations. Fig. 10 shows the best testing out of the 100 trials using the EBP algorithm. When the network size was increased from 8 to 13 neurons (117 weights), the EBP algorithm was able to reach the similar training error as with the NBN algorithm, but the network lost its ability to respond correctly for new patterns (between training points). Please notice

TABLE II
TRAINING RESULTS OF THE PEAK PROBLEM

Neurons	Success Rate		Average Iteration		Average Time (s)	
	EBP	NBN	EBP	NBN	EBP	NBN
8	0%	5%	Failing	222.5	Failing	0.33
9	0%	29%	Failing	214.6	Failing	0.58
10	0%	61%	Failing	183.5	Failing	0.70
11	0%	76%	Failing	177.2	Failing	0.93
12	0%	90%	Failing	149.5	Failing	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

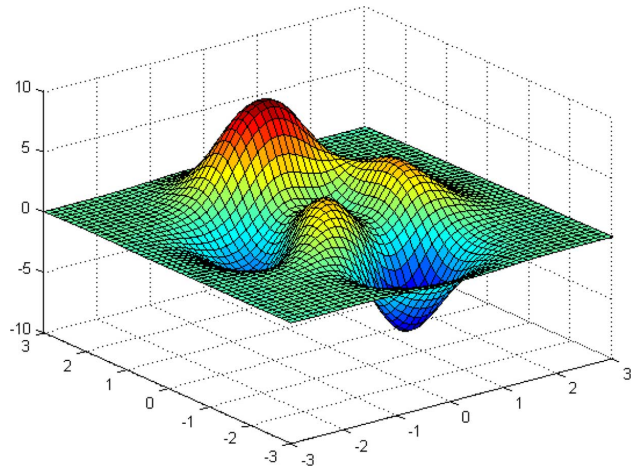

Fig. 9. The best training result in 100 trials, using NBN algorithm, eight neurons in FCC network (52 weights); maximum training iteration is 1000; $\mathrm{SSE}_{\mathrm{Train}} = 0.0015$ and $\mathrm{SSE}_{\mathrm{Verify}} = 0.0125$.

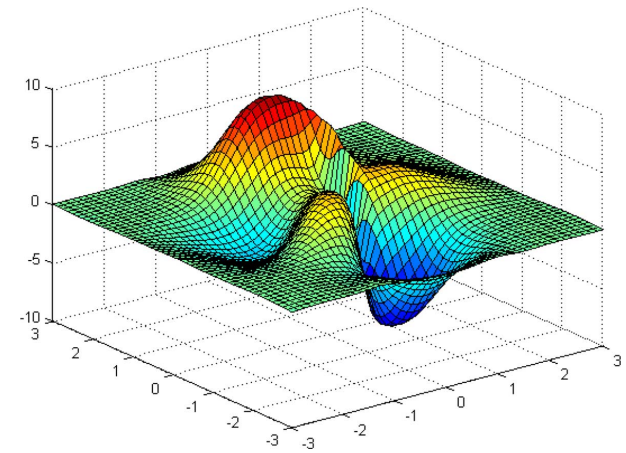

Fig. 10. The best training result in 100 trials, using EBP algorithm, eight neurons in FCC network (52 weights); maximum training iteration is 1 000 000; $\mathrm{SSE}_{\mathrm{Train}} = 0.0205$ and $\mathrm{SSE}_{\mathrm{Verify}} = 0.2942$.

that indeed with an increased number of neurons (13 neurons), the EBP algorithm was able to train the network to a small error $\mathrm{SSE}_{\mathrm{Train}} = 0.0005$, but as one can see from Fig. 11, the result is unacceptable with verification error $\mathrm{SSE}_{\mathrm{Verify}} = 0.3231$ (larger errors between training points).

From the example presented, one may notice that often in simple (close to optimal) networks, the EBP algorithm cannot converge to the required training error (Fig. 10). When network size increases, the EBP algorithm can reach the required

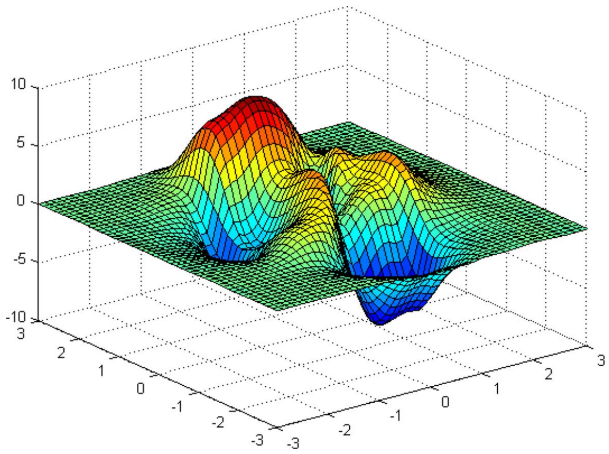

Fig. 11. The best training result in 100 trials, using EBP algorithm, 13 neurons in FCC network (117 weights); maximum training iteration is 1 000 000; $\mathrm{SSE}_{\mathrm{Train}} = 0.0005$ and $\mathrm{SSE}_{\mathrm{Verify}} = 0.3231$.

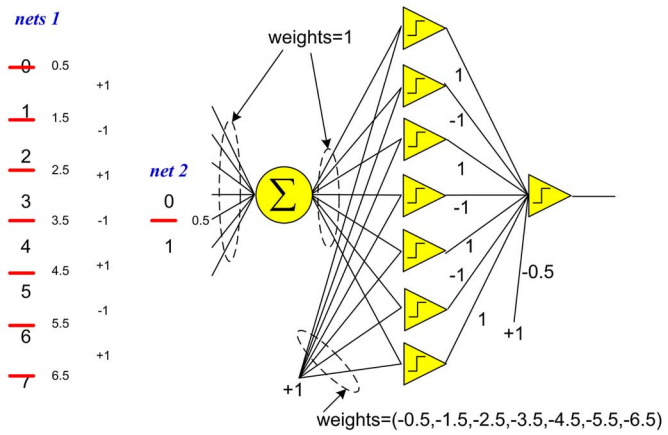

Fig. 12. MLP with one hidden layer for solving the Parity-7 problem.

training error, but the trained networks lose their generalization ability and cannot process new patterns well (Fig. 11). On the other hand, second-order algorithms, such as the NBN algorithm, work not only significantly faster, but can find good solutions with close to optimal networks (Fig. 9).

As it was shown in the three examples above, the popular EBP algorithm is 100 to 1000 times slower (depending on required accuracy) than second-order algorithms. Speeds of second-order algorithms such as LM or NBN are comparable. However, the LM algorithm as presented in [41] has several restrictions in comparison to the NBN algorithm.

- The NBN algorithm can handle any arbitrarily connected [45], feed forward neural network while the LM algorithm can only handle MLP networks [41].
- In the NBN algorithm, there is no need to calculate and store the Jacobian matrix with a size that is proportional to the number of training patterns. This advantage allows the NBN algorithm to be used for problems with essentially an unlimited number of patterns [46].
- The NBN algorithm addresses the "flat spot" problem [52], [71], resulting in a slightly better success rate than the LM algorithm.
- The NBN algorithm uses only a single, forward computational scheme making it more efficient, especially for neural networks with multiple outputs [47].
- The NBN algorithm makes it easier to find a close to optimal architecture by starting with a FCC topology (Fig. 3), which can then be pruned to a BMLP topology (Fig. 2) [46].

III. Efficiencies of Neural Network Architectures With Hard Activation Functions

Neural network architectures vary in complexity and efficiency. Research has thus far shown that some of the most efficient and reliable networks have fewer neurons, but also require multiple layers with connections between all layers [53]. When a hard threshold activation function is assumed, it can be shown that the popular MLP architecture is less efficient than other architectures [54]. Let us compare the performance of various architectures using well know Parity-N benchmarks. Notice that only the number of inputs is important and their locations are irrelevant. Therefore, in order to simplify analysis and the network drawing, we will introduce a summing operator in front of the network, as shown in Fig. 12.

A. MLP—Multilayer Perceptron Architecture With One Hidden Layer

For the MLP architecture with one hidden layer, a generalized solution for all Parity-N cases requires n neurons

$$n = N + 1 \tag{1}$$

where N is the parity number as well as the number of neurons in the hidden layer [54]–[56]. The output neuron performs the AND operation on all neuron outputs from the hidden layer. Fig. 12 shows the unipolar implementation of a Parity-7 MLP network with one hidden layer. While this particular implementation is easy to train, it requires an excessive number of neurons for large Parity-N problems.

B. BMLP—Bridged Multilayer Perceptron Architecture With One Hidden Layer

For the BMLP architecture using n hidden neurons in one hidden layer, a generalized solution for the largest Parity-N that can be solved is [55], [56]

$$N = 2n + 1. \tag{2}$$

The output neuron performs the AND operation on all neuron outputs from the hidden layer. Fig. 13 shows the unipolar implementation of a Parity-7 BMP network with one hidden layer.

C. BMLP—Bridged Multilayer Perceptron Architecture With Two or More Hidden Layers

With two hidden layers, as seen in Fig. 14, the largest Parity-N problem that can be solved by this network is defined by

$$N = 2(n_1 + 1)(n_2 + 1) - 1 \tag{3}$$

where n_1 and n_2 are the number of neurons in the first and second hidden layers, respectively [55]. Fig. 14 shows networks

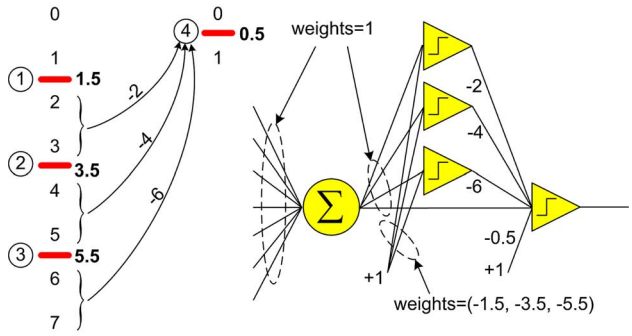

Fig. 13. BMLP with one hidden layer for solving the Parity-7 problem.

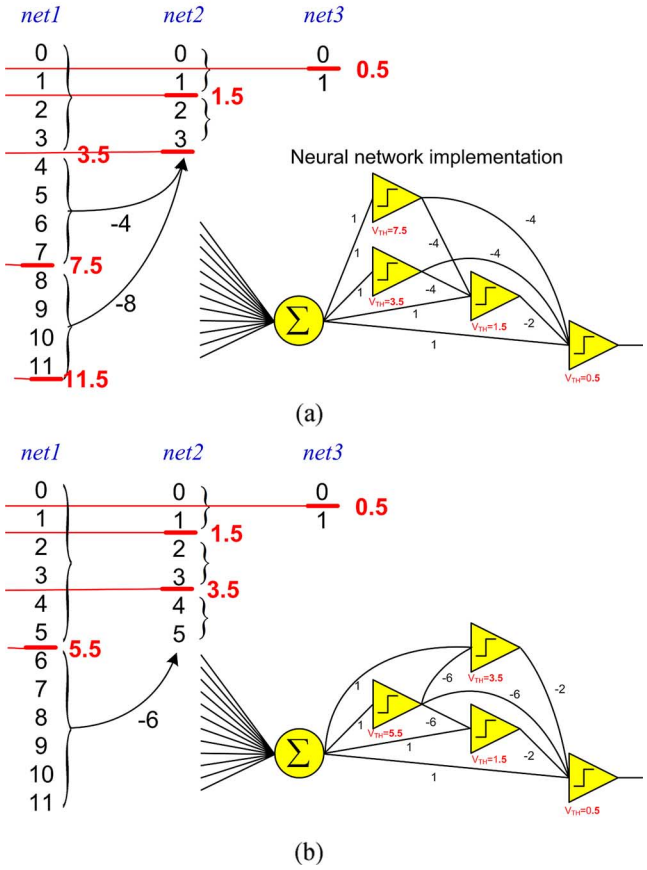

Fig. 14. BMLP with two hidden layers for solving the Parity-11 problem. (a) Two neurons in the first hidden layer and one neuron in the second hidden layer. (b) Two neurons in the second hidden layer and one neuron in the first hidden layer.

for solving Parity-11 problem. From (3) and Fig. 14, one may notice that the power of the neural network is the same regardless of whether it is the first or the second hidden layer that uses more neurons.

As shown in [56], (3) can be generalized for k hidden layers

$$N = 2(n_1 + 1)(n_2 + 1)\dots(n_{k-1} + 1)(n_k + 1) - 1 \quad (4)$$

where N represents the parity number; n_1 is the number of neurons in the first hidden layer; n_2 is the number of neurons in the second hidden layer; and so on; until n_k which is the number of neurons in the last hidden layer.

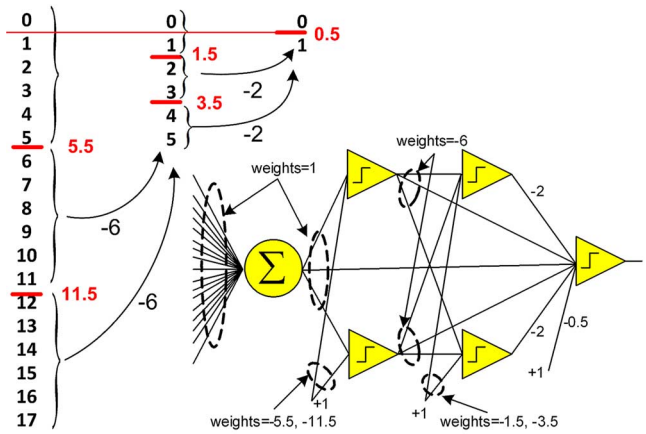

Fig. 15. BMLP with two hidden layers for solving the Parity-17 problem.

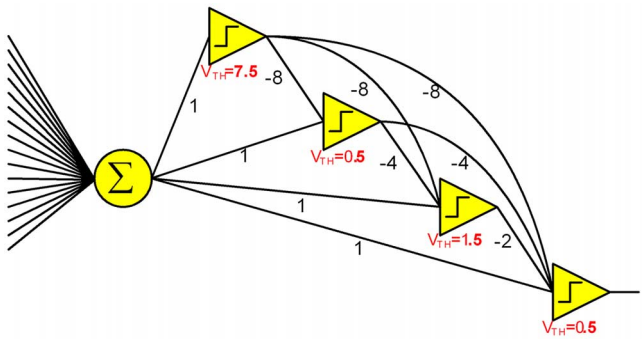

Fig. 16. FCC architecture for solving Parity-15 problem.

By introducing additional hidden layers in the BMLP network, the power of the network is significantly increased. Larger Parity-N problems can be solved with fewer neurons, but additional cross-layer connections are introduced.

Fig. 15 shows a BMLP network in the $17 = 2 = 2 = 1$ configuration. The addition of 1 additional neuron in one of the hidden layers enables this network to solve the Parity-17 problem. Fig. 15 (right) shows the required network structure and Fig. 15 (left) shows the concept of operation for the network.

D. FCC—Fully Connected Cascade Architecture

Use of an FCC network can further reduce the number of neurons in the network. One may notice that in the FCC architecture, all neurons in the network are directly connected. When some of these connections are removed the network will be reduced to the BMLP architecture.

The largest Parity-N problem which can be solved with n neurons in the FCC architecture is [55]

$$N = 2^n - 1 \quad (5)$$

where n is the total number of neurons in the network.

Fig. 16 shows the unipolar implementation of a FCC network for the Parity-15 problem.

One of the difficulties in implementing the FCC network is the large number of layers.

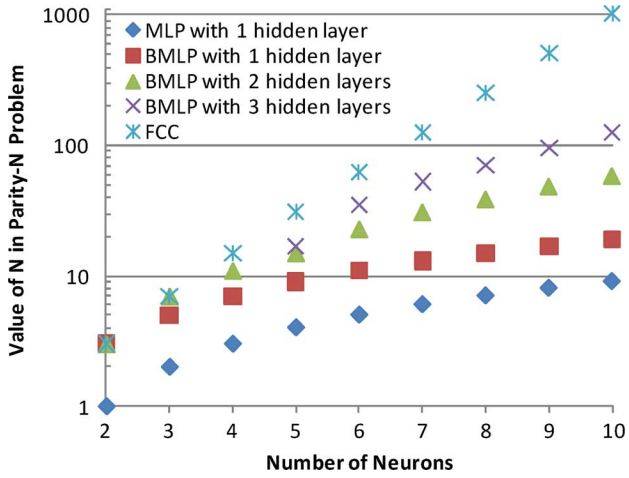

Fig. 17. Efficiency comparison of various neural network architectures.

TABLE III
NEURONS AND WEIGHTS REQUIRED BY VARIOUS ARCHITECTURES

Architecture	Parity-3		Parity-7		Parity-15		Parity-31		Parity-63	
	# neurons	# weights	# neurons	# weights	# neurons	# weights	# neurons	# weights	# neurons	# weights
MLP 1 hidden layer	4	16	8	64	15	256	32	1024	64	4096
BMLP 1 hidden layer	3	14	5	44	9	152	17	560	33	2144
BMLP 2 hidden layers	N/A		3	27	5	88	7	239	11	739
BMLP 3 hidden layers	N/A		N/A		4	70	6	203	8	530
FCC	2	9	3	27	4	70	5	170	6	399

E. Comparison of NN Efficiencies

The three architectures shown so far are capable of solving the Parity-N problem; however, each of these architectures has a different level of efficiency. Fig. 17 compares architecture efficiency.

The MLP architecture is the least efficient of the presented architectures and requires many neurons, typically, $N + 1$, to solve the Parity-N problem.

The BMLP architecture with one hidden layer improves upon the MLP by significantly reducing the number of neurons required for solving the same Parity-N problem. If we implement a BMLP with two or three hidden layers, the number of required neurons for solving the same Parity-N problem is reduced even further.

The FCC architecture provides additional improvement in efficiency over the BMLP, requiring even fewer neurons than the BMLP for the same Parity-N problem.

To better understand the efficiency improvement in each of the presented architectures, we will use the MLP as our basis for comparison. Table III shows a summary of the number of neurons and weights required by different architectures [55].

TABLE IV
EXPERIMENTAL RESULTS FOR MLP ARCHITECTURE WITH ONE HIDDEN LAYER (IN EACH BOX THE FOLLOWING VALUES ARE LISTED: SUCCESS RATE, # ITERATIONS, TIME)

# Total Neurons	NBN MLP1 Problem									
	Parity-3	Parity-4	Parity-5	Parity-6	Parity-7	Parity-8	Parity-9	Parity-10	Parity-11	Parity-12
3	51.8% 231 0.0315	32.3% 560 0.0868	0.0% – –	0.0% – –	0.0% – –	0.0% – –	0.0% – –	0.0% – –	0.0% – –	0.0% – –
4	83.6% 94 0.0297	76.0% 234 0.0900	29.1% 933 0.3358	1.2% 1791 1.8396	0.0% – –	0.0% – –	0.0% – –	0.0% – –	0.0% – –	0.0% – –
5	94.2% 50 0.0334	91.7% 113 0.0937	71.3% 459 0.3111	29.5% 1405 1.6158	0.4% 2449 3.5357	0.0% – –	0.0% – –	0.0% – –	0.0% – –	0.0% – –
6	97.7% 35 0.0382	97.0% 75 0.1092	88.1% 246 0.2660	68.3% 840 1.2941	23.8% 2074 3.2713	0.4% 3196 4.8969	0.0% – –	0.0% – –	0.0% – –	0.0% – –
7	99.1% 30 0.0451	98.8% 70 0.1384	92.5% 181 0.2458	85.6% 547 1.1234	57.8% 1472 2.7656	7.5% 3050 4.7934	0.7% 4042 11.0413	0.0% – –	0.0% – –	0.0% – –
8	99.7% 29 0.0525	99.6% 71 0.1665	93.1% 170 0.2582	93.3% 384 0.9746	75.4% 1101 2.4584	23.2% 2712 4.6904	4.1% 3979 10.8658	0.0% – –	0.0% – –	0.0% – –
9	99.9% 28 0.0586	99.9% 73 0.1937	93.8% 166 0.2975	96.5% 309 0.9406	86.4% 845 2.2983	40.5% 2344 4.8953	10.3% 3851 10.4410	1.2% 4977 18.5973	0.2% 6049 64.9035	0.0% – –
10	100.0% 28 0.0665	100.0% 76 0.2265	96.0% 158 0.3889	98.5% 282 1.0288	91.1% 682 2.0845	57.5% 1956 5.0498	19.9% 3658 10.7275	4.1% 4930 18.6700	0.8% 6041 66.6671	0.2% 7198 95.1553
11	100.0% 28 0.0734	100.0% 77 0.2493	97.1% 164 0.4921	99.5% 280 1.1577	92.3% 580 1.8562	69.7% 1600 4.6973	30.0% 3456 11.3925	8.9% 4831 19.4236	1.0% 6033 75.1835	0.2% 7198 96.4368
12	100.0% 28 0.0780	100.0% 74 0.2560	96.3% 181 0.5698	100.0% 295 1.3236	93.7% 474 1.6388	76.8% 1319 4.0918	37.9% 3289 11.6512	15.3% 4681 20.7855	0.5% 6032 88.6892	0.0% – –

Analyzing Fig. 17 and Table III, one may notice that there is a significant difference in capabilities of neural network architectures. For example, with ten neurons, using popular MLP architecture with one hidden layer, the largest parity problem which can be solved is the Parity-9. However, if the same ten neurons are connected in FCC topology, a problem as large as Parity-1023 can be solved.

IV. EXPERIMENTAL COMPARISONS OF VARIOUS NEURAL NETWORK ARCHITECTURES

As it was shown in Section II, the NBN algorithm is superior to other algorithms. Therefore, the NBN algorithm was used in the following experimental comparison of various neural network architectures.

The results for MLP architecture with one hidden layer are shown in Table IV. Similar results were generated for MLP with 2 and 3 hidden layers and for BMLP with 1, 2, and 3 hidden layers. Also, the experiment was done with FCC architectures. In all experiments, parity problems from 4 to 12 were conducted and the total number of neurons varied from 3 to 12. For the given number of neurons, the quantity of neurons in hidden layers was distributed as equally as possible.

Simulations were run using the same NBN training algorithm, but varying only in the chosen architecture, as presented in Section II. Per all network interconnections, initial nonzero weights were randomly generated, and training trials were run and averaged 100 times up to a maximum number of algorithm iterations per each time. Parity and total number of neurons were varied per each experiment, and results were tabulated.

From experimental results, several conclusions could be drawn.

1) Number of required iterations increases significantly with increasing parity number.

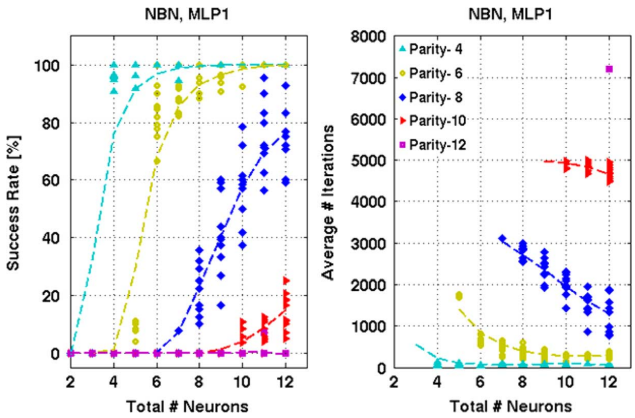

Fig. 18. Results for for MLP architecture with one hidden layer.

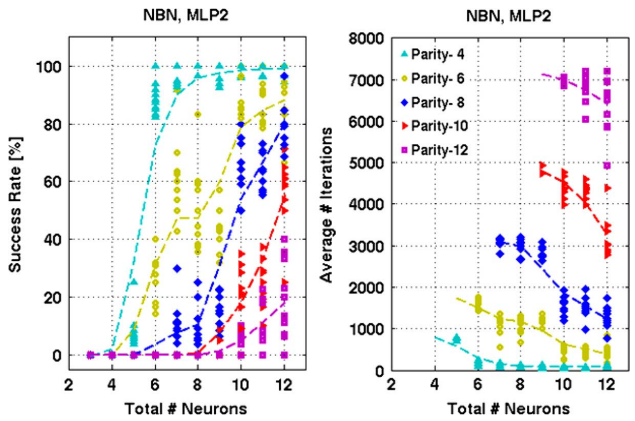

Fig. 19. Results for for MLP architecture with two hidden layers.

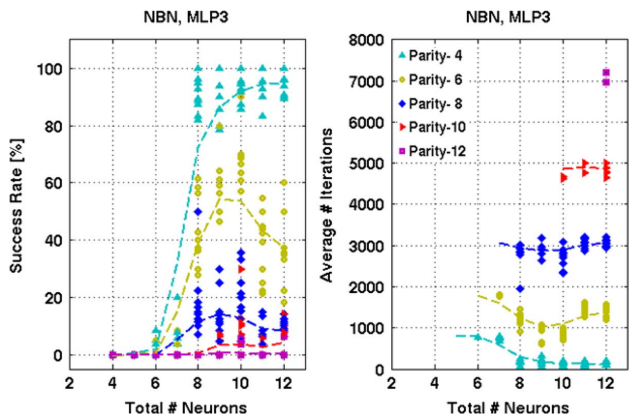

Fig. 20. Results for MLP architecture with three hidden layers.

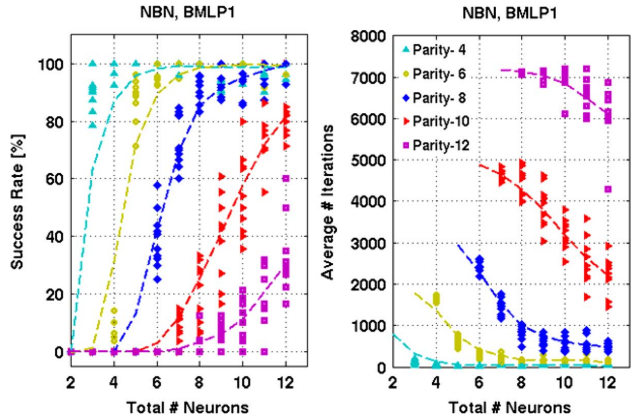

Fig. 21. BMLP networks with one hidden layer.

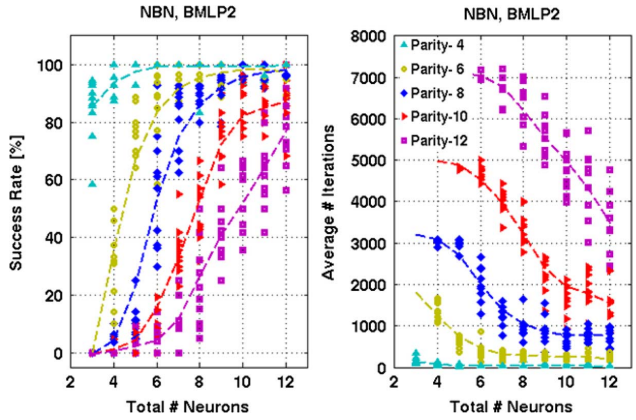

Fig. 22. BMLP with two hidden layers.

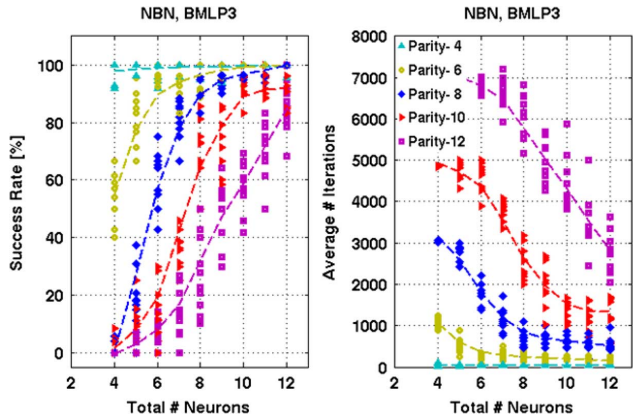

Fig. 23. BMLP with three hidden layers.

2) Number of required iterations decreases with increasing number of neurons used in the network.
3) Computing time increases with number of neurons.
4) Success rate decreases with increasing parity number.
5) Success rate increases with number of neurons used.

Items (1)–(4) seem to be logical. The plots illustrating success rates for parities 4, 6, 8, 10, and 12 for all investigated architectures are shown in Figs. 18–24.

V. Generalization Issues

In order to sustain neural network generalization abilities, the network should have as few neurons and weights as possible [42], [43], [47]. The problem is similar to curve fitting using polynomial interpolation, as illustrated in Fig. 25. Notice that if the order of the polynomial is too low (first to fifth orders) there is a poor approximation everywhere. When order is too high (eighth to tenth orders) there is a perfect fit at the given data

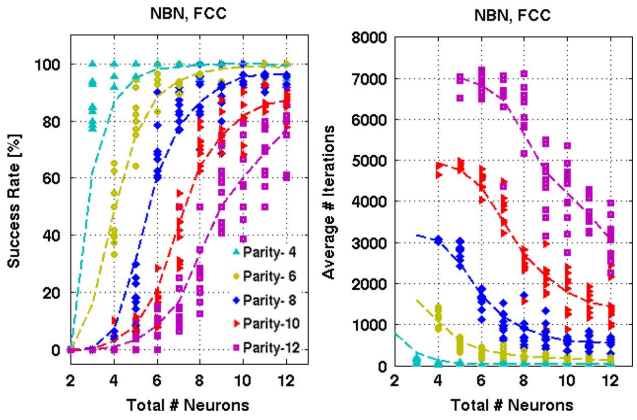

Fig. 24. Results for FCC architecture.

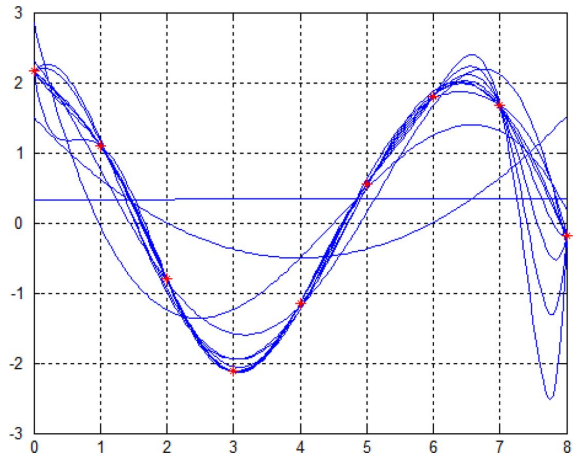

Fig. 25. Approximation of measured points by polynomials with different orders starting with first through tenth order.

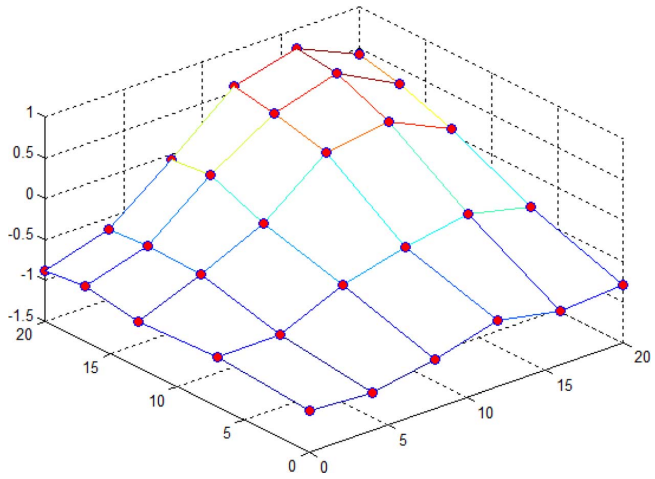

Fig. 26. Control surface: determined by dataset used to build fuzzy controller with $6 \times 5 = 30$ points.

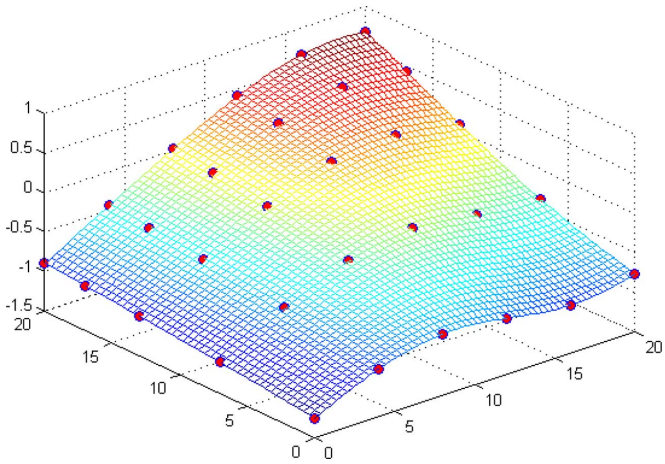

Fig. 27. Interpolation result obtained with two neurons in FCC network, training sum square error is 0.6712.

points, but the interpolation between points is rather poor. Optimal results are obtained by sixth or seventh order polynomials.

In the case of neural networks, we observe similar results; with excessive numbers of neurons, it is easy to train the network to very small output error with a finite set of training data, but this may lead to very poor and frustrating results when this trained neural network is used to process new input patterns.

As it can be seen from the experimental data of Sections II and IV, the success rates for all neural network architectures increase with increased number of neurons. Indeed, such larger networks can be trained faster and converge to smaller errors, but this "success" is very misleading. Such networks with excessive numbers of neurons are most likely losing their generalization abilities. It means that they will respond very poorly for new patterns never used in the training. 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.

To illustrate the problem with neural networks, let us try to find neural network architecture to replace a fuzzy controller. Fig. 26 shows the required control surface and the defuzzification rules for the TSK (Tagagi, Sugeno, Ken) [57], [58] fuzzy controller [5], [29].

Let us select the FCC neural network architecture and apply the NBN algorithm for neural network design. In order to find solutions, FCC networks consisting of a different number of neurons are tried. The interpolation results of FCC networks consisting of two to five neurons are presented as Figs. 27–30, respectively.

As shown in Fig. 27, the FCC network with two neurons is not powerful enough to match the training data to small errors (0.6712). As the number of neurons increased, the training errors get smaller (0.1119 for three neurons and 0.0366 for four neurons) and results become more acceptable, as shown in Figs. 28 and 29. One may notice that the best results were obtained for the four neuron architecture (Fig. 29). However, when five neurons were used we were able to reduce the training error (0.0089), but the neural network begins to lose its generalization ability (Fig. 30).

The conclusion is that for the best generalization abilities, neural networks should have as few neurons as possible. 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.

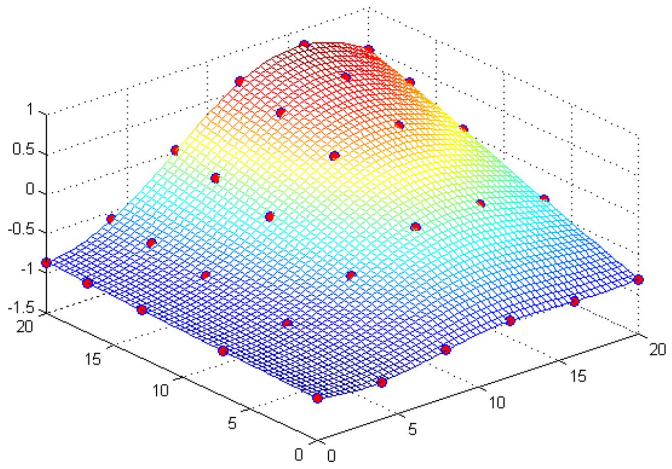

Fig. 28. Interpolation result obtained with three neurons in FCC network, training sum square error is 0.1118.

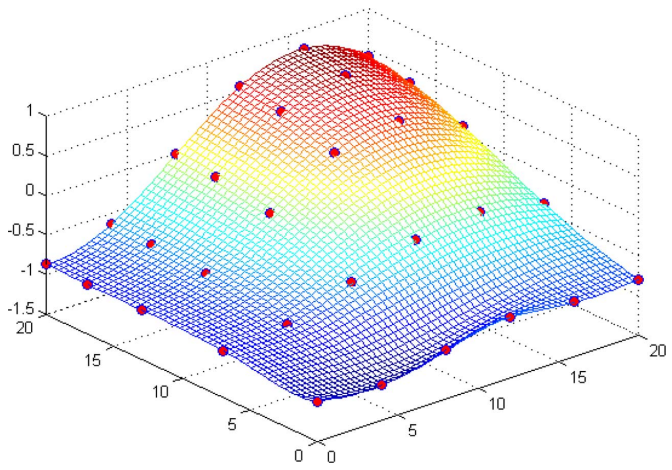

Fig. 29. Interpolation result obtained with four neurons in FCC network, training sum square error is 0.0366.

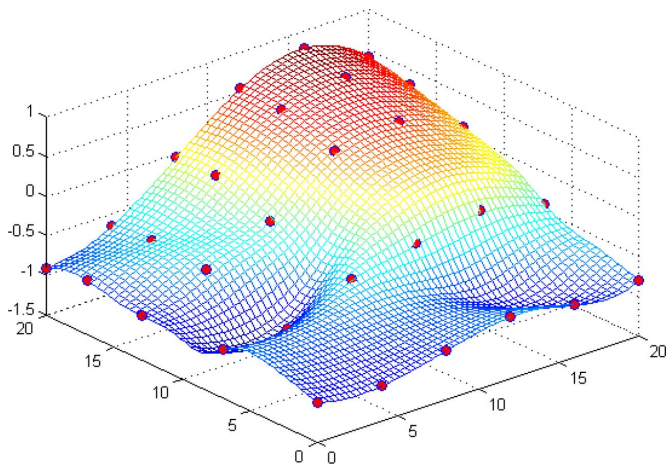

Fig. 30. Interpolation result obtained with five neurons in FCC network, training sum square error is 0.0089.

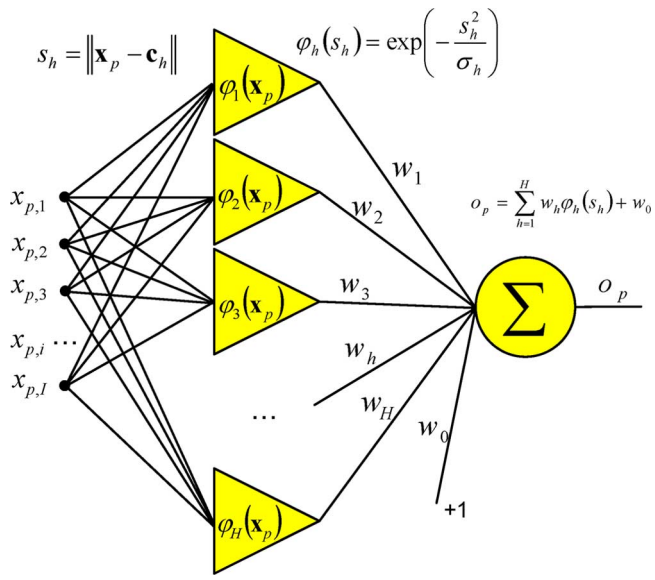

Fig. 31. Architecture of radial basis function networks.

VI. Other Networks

A. Radial Basis Function (RBF) Networks

RBF networks have architecture similar to MLP networks with one hidden layer. The architecture of RBF networks (Fig. 31) is fixed and consists of three layers: 1) input layer with pattern data; 2) hidden layer with RBF units for nonlinear mapping using Gaussian functions; and 3) output layer (summator) which performs linear separation. Number of nonlinear hidden units is usually significantly larger that number of network inputs, so input space is transformed into higher dimensional space, where patterns become linearly separable [61]. Recently, RBF networks became attractive in practical applications [51] because of simpler design process and improved generalization ability compared to neural networks [62].

Unlike the randomly generated initial parameters (weights) in neural networks, it is important in RBF networks to find proper initial network parameters such as locations of centers of hidden units [63], [64]. Otherwise, multiple RBF hidden units may be trapped in the same center and the training may never converge.

Also, because of its fixed one hidden layer architecture, RBF networks usually require many more units than neural networks to solve the same problem. Let us use the two-spiral problem (Fig. 6) as an example to compare the performance of RBF networks and neural networks.

Experimental results show that in order to reach the same training sum square error level (0.001), RBF networks require at least 30 hidden units; while neural networks can solve the problem with only seven hidden units using FCC architectures. However, the generalization abilities of RBF networks are better than traditional neural networks (Fig. 32). The error correction (ErrCor) algorithm allows for automatic finding of best locations of centers of hidden units, while an improved second-order (ISO) algorithm presented significantly improve the training process of RBF networks [40]. One should keep in mind that also RBF networks need to be designed as compact as possible to secure its generalization abilities.

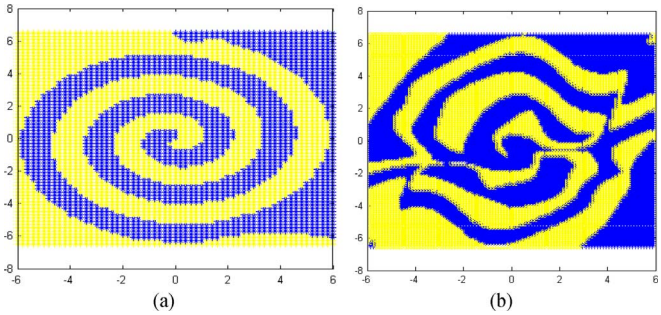

Fig. 32. Generalization results of two-spiral problem. (a) RBF networks with 30 hidden units. (b) Neural networks with seven hidden units.

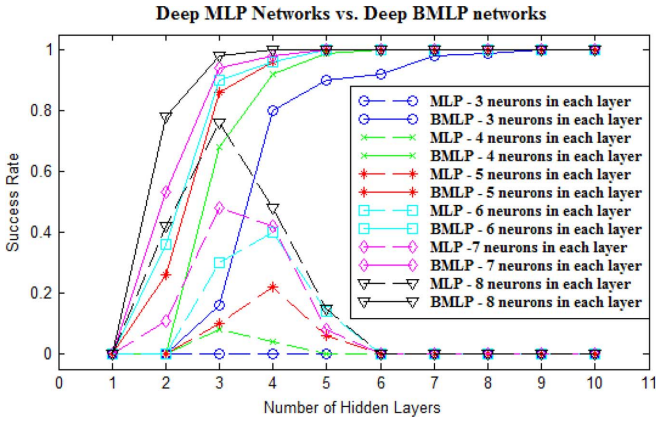

Fig. 33. Success rates comparison for training the two-spiral patterns, using different number of hidden layers (from 1 to 10). For each network, all the hidden layers consist of the same number of neurons (from 3 to 8).

B. Deep Neural Networks

Recently, there is an emerging interest in deep neural networks [67], [68] which consist of multiple hidden layers connected in MLP architecture. These deep architectures are attractive because they provide significantly larger computing power than shallow neural networks. However, it is very difficult to train deep neural networks.

We are suggesting another solution. By introducing connections across layers into deep MLP networks (deep BMLP topologies) the networks become more transparent for the learning process.

As another example, let us train the deep MLP networks and deep BMLP networks using the two-spiral problem of Fig. 6. Each hidden layer consists of seven neurons. Fig. 33 presents the training success rates using different numbers of hidden layers from 1 to 10. For a given network, all hidden layers have the same number of neurons (from 3 to 8). One may notice that traditional deep neural network architectures (MLP) cannot be efficiently trained if network become too deep. With connections across layers (BMLP) the depth of the networks actually makes training easier.

Also, using Parity-N problems and comparing results for networks with three hidden layers (Figs. 20 and 23), one may notice that much larger success rate and smaller numbers of iterations are required for BMLP (Fig. 23) in comparison to MLP architectures (Fig. 20).

One may notice that the connections across layers not only improve the training process of deep architectures, but also significantly increase the power of networks (see Fig. 17 and Table III).

VII. Conclusion

This paper focused on the discussion about offline neural network design. First, the importance of the proper learning algorithm was emphasized. With advanced learning algorithms, we can train closer to optimal architectures which cannot be trained with simple algorithms. The traditional LM algorithm adopted in the popular MATLAB NN Tool Box [59] can handle only MLP topology without connections across layers, and these topologies are far from optimal. The training tool [60] developed based on the NBN algorithm can handle arbitrarily connected neural networks. The training tool and computation engine of the NBN algorithm can be easily and freely downloaded from the website: http://www.eng.auburn.edu/~wilambm/nnt/. The offline design of neural networks requires that all the datasets are known before the training process. In the applications where networks need to be adjusted dynamically using time varying datasets, the offline design process is no longer suitable; instead, online training algorithms should be applied [65], [66].

Then, it was shown that neural network architectures vary in complexity and efficiency. The power of neural networks increases significantly with increased number of connections across layers. The most powerful are FCC architectures with the largest number of connections across layers per number of neurons (see Fig. 17 and Table III). However, with increased depth more nonlinear transformations are made and this may require higher accuracy of computing and larger sensitivity to noise.

A reasonable compromise is to use BMLP where large power of the network can be obtained even if number of layers is significantly limited. The BMLP network is much easier to train than traditional deep neural network architectures (MLP).

Since we still have to use the trial and error approach to find number of layers and number of neuron in each hidden layer, the major problem is how to reduce the number of trials. With FCC architectures, there is a single topology precisely defined by the number of neurons used. Since the power of the FCC network increases dramatically with the number of neurons, usually not many trials are required. We have to always remember to select the architecture with acceptable training error and the smallest number of neurons. Further optimization of the FCC topology is possible by pruning some connections between neurons so that the FCC network is converted to a BMLP network with reduced depth.

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] T. Orlowska-Kowalska, M. Dybkowski, and K. Szabat, "Adaptive sliding-mode neuro-fuzzy control of the two-mass induction motor drive without mechanical sensors," *IEEE Trans. Ind. Electron.*, vol. 57, no. 2, pp. 553–564, Feb. 2010.

[3] M. Pucci and M. Cirrincione, "Neural MPPT control of wind generators with induction machines without speed sensors," *IEEE Trans. Ind. Electron.*, vol. 58, no. 1, pp. 37–47, Jan. 2011.

[4] V. N. Ghate and S. V. Dudul, "Cascade neural-network-based fault classifier for three-phase induction motor," *IEEE Trans. Ind. Electron.*, vol. 58, no. 5, pp. 1555–1563, May 2011.

[5] C. Xia, C. Guo, and T. Shi, "A neural-network-identifier and fuzzy-controller-based algorithm for dynamic decoupling control of permanent-magnet spherical motor," *IEEE Trans. Ind. Electron.*, vol. 57, no. 8, pp. 2868–2878, Aug. 2010.

[6] F. F. M. El-Sousy, "Hybri-based wavelet-neural-network tracking control for permanent-magnet synchronous motor servo drives," *IEEE Trans. Ind. Electron.*, vol. 57, no. 9, pp. 3157–3166, Sep. 2010.

[7] F. F. M. El-Sousy, "Hybrid $H\infty$ based wavelet-neural-network tracking control for permanent-magnet synchronous motor servo drives," *IEEE Trans. Ind. Electron.*, vol. 57, no. 9, pp. 3157–3166, Sep. 2010.

[8] Q. N. Le and J. W. Jeon, "Neural-network-based low-speed-damping controller for stepper motor with an FPGA," *IEEE Trans. Ind. Electron.*, vol. 57, no. 9, pp. 3167–3180, Sep. 2010.

[9] C.-H. Lu, "Wavelet fuzzy neural networks for identification and predictive control of dynamic systems," *IEEE Trans. Ind. Electron.*, vol. 58, no. 7, pp. 3046–3058, Jul. 2011.

[10] R. H. Abiyev and O. Kaynak, "Type 2 fuzzy neural structure for identification and control of time-varying plants," *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, pp. 4147–4159, Dec. 2010.

[11] M. A. S. K. Khan and M. A. Rahman, "Implementation of a wavelet-based MRPID controller for benchmark thermal system," *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, pp. 4160–4169, Dec. 2010.

[12] A. Bhattacharya and C. Chakraborty, "A shunt active power filter with enhanced performance using ANN-based predictive and adaptive controllers," *IEEE Trans. Ind. Electron.*, vol. 58, no. 2, pp. 421–428, Feb. 2011.

[13] J. Lin and R.-J. Lian, "Intelligent control of active suspension systems," *IEEE Trans. Ind. Electron.*, vol. 58, no. 2, pp. 618–628, Feb. 2011.

[14] Z. Lin, J. Wang, and D. Howe, "A learning feed-forward current controller for linear reciprocating vapor compressors," *IEEE Trans. Ind. Electron.*, vol. 58, no. 8, pp. 3383–3390, Aug. 2011.

[15] M. O. Efe, "Neural network assisted computationally simple PI_λ D_μ control of a quadrotor UAV," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 354–361, May 2011.

[16] T. Orlowska-Kowalska and M. Kaminski, "FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system," *IEEE Trans. Ind. Informat.*, vol. 7, no. 3, pp. 436–445, Aug. 2011.

[17] C.-F. Juang, Y.-C. Chang, and C.-M. Hsiao, "Evolving gaits of a hexapod robot by recurrent neural networks with symbiotic species-based particle swarm optimization," *IEEE Trans. Ind. Electron.*, vol. 58, no. 7, pp. 3110–3119, Jul. 2011.

[18] C.-C. Tsai, H.-C. Huang, and S.-C. Lin, "Adaptive neural network control of a self-balancing two-wheeled scooter," *IEEE Trans. Ind. Electron.*, vol. 57, no. 4, pp. 1420–1428, Apr. 2010.

[19] G. W. Chang, C.-I. Chen, and Y.-F. Teng, "Radial-basis-function-based neural network for harmonic detection," *IEEE Trans. Ind. Electron.*, vol. 57, no. 6, pp. 2171–2179, Jun. 2010.

[20] R.-J. Wai and C.-Y. Lin, "Active low-frequency ripple control for clean-energy power-conditioning mechanism," *IEEE Trans. Ind. Electron.*, vol. 57, no. 11, pp. 3780–3792, Nov. 2010.

[21] Yahyaoui, N. Fnaiech, and F. Fnaiech, "A suitable initialization procedure for speeding a neural network job-shop scheduling," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 1052–1060, Mar. 2011.

[22] V. Machado, A. Neto, and J. D. de Melo, "A neural network multiagent architecture applied to industrial networks for dynamic allocation of control strategies using standard function blocks," *IEEE Trans. Ind. Electron.*, vol. 57, no. 5, pp. 1823–1834, May 2010.

[23] M. Charkhgard and M. Farrokhi, "State-of-charge estimation for lithium-ion batteries using neural networks and EKF," *IEEE Trans. Ind. Electron.*, vol. 57, no. 12, Dec. 2010.

[24] M. Wilamowski and O. Kaynak, "Oil well diagnosis by sensing terminal characteristics of the induction motor," *IEEE Trans. Ind. Electron.*, vol. 47, no. 5, pp. 1100–1107, Oct. 2000.

[25] N. J. Cotton and B. M. Wilamowski, "Compensation of nonlinearities using neural networks implemented on inexpensive microcontrollers," *IEEE Trans. Ind. Electron.*, vol. 58, no. 3, pp. 733–740, Mar. 2011.

[26] A. Gomperts, A. Ukil, and F. Zurfluh, "Development and implementation of parameterized FPGA-based general purpose neural networks for online applications," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 78–89, Feb. 2011.

[27] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "FPGAs in industrial control applications," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 224–243, May 2011.

[28] A. Dinu, M. N. Cirstea, and S. E. Cirstea, "Direct neural-network hardware-implementation algorithm," *IEEE Trans. Ind. Electron.*, vol. 57, no. 5, pp. 1845–1848, May 2010.

[29] A. Malinowski and H. Yu, "Comparison of various embedded system technologies for industrial applications," *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 244–254, May 2011.

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

[31] P. J. Werbos, "Back-propagation: Past and future," in *Proc. Int. Conf. Neural Netw.*, San Diego, CA, 1988, pp. 343–354, 1.

[32] S. Ferrari and M. Jensenius, "A constrained optimization approach to preserving prior knowledge during incremental training," *IEEE Trans. Neural Netw.*, vol. 19, no. 6, pp. 996–1009, Jun. 2008.

[33] Q. Song, J. C. Spall, Y. C. Soh, and J. Ni, "Robust neural network tracking controller using simultaneous perturbation stochastic approximation," *IEEE Trans. Neural Netw.*, vol. 19, no. 5, pp. 817–835, May 2008.

[34] A. Slowik, "Application of an adaptive differential evolution algorithm with multiple trial vectors to artificial neural network training," *IEEE Trans. Ind. Electron.*, vol. 58, no. 8, pp. 3160–3167, Aug. 2011.

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

[36] A. Salvetti and B. M. Wilamowski, "Introducing stochastic processes within the backpropagation algorithm for improved convergence," in *Proc. Artif. Neural Netw. Eng.–ANNIE'94*.

[37] B. M. Wilamowski and L. Torvik, "Modification of gradient computation in the back-propagation algorithm," in *Artif. Neural Netw. Eng.–ANNIE'93*.

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

[39] S. E. Fahlman, An Empirical Study of Learning Speed in Backpropagation Networks 1988.

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

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

[42] B. M. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Ind. Electron. Mag.*, vol. 3, no. 4, pp. 56–63, Dec. 2009.

[43] B. M. Wilamowski, "Challenges in applications of computational intelligence in industrial electronics," in *Proc. Int. Symp. Ind. Electron.-ISIE'10*, Bari, Italy, Jul. 4–7, 2010, pp. 15–22.

[44] N. J. Cotton, B. M. Wilamowski, and G. Dundar, "A neural network implementation on an inexpensive eight bit microcontroller," in *Proc. 12th Int. Conf. Intell. Eng. Syst.–INES'08*, Miami, Florida, USA, Feb. 25–29, 2008, pp. 109–114.

[45] 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.

[46] B. M. Wilamowski and H. Yu, "Improved computation for Levenberg Marquardt training," *IEEE Trans. Neural Netw.*, vol. 21, no. 6, pp. 930–937, Jun. 2010.

[47] B. M. Wilamowski and H. Yu, "Neural network learning without backpropgation," *IEEE Trans. Neural Netw.*, vol. 21, no. 11, pp. 1793–1803, Nov. 2010.

[48] J. R. Alvarez-Sanchez, "Injecting knowledge into the solution of the two-spiral problem," *Neural Comput. Appl.*, vol. 8, pp. 265–272, Aug. 1999.

[49] S. Wan and L. E. Banta, "Parameter incremental learning algorithm for neural networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1424–1438, Jun. 200.

[50] S. E. Fahlman and C. Lebiere, , D. S. Touretzky, Ed., "The cascade-correction learning architecture," in *Advances in Neural Information Processing Systems 2*. San Mateo, CA: Morgan Kaufmann.

[51] H. Yu, T. T. Xie, S. Paszczynski, and B. M. Wilamowski, "Advantages of radial basis function networks for dynamic system design," *IEEE Trans. Ind. Electron.*, vol. 58, no. 12, pp. 5438–5450, Dec. 2011.

[52] B. M. Wilamowski and L. Torvik, "Modification of gradient computation in the back-propagation algorithm," in *Proc. Artif. Neural Netw. Eng.–ANNIE'93*, St. Louis, MO, Nov. 14–17, 1993, pp. 175–180.

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

[54] S. Trenn, "Multilayer perceptrions: Approximation order and necessary number of hidden units," *IEEE Trans. Neural Netw.*, vol. 19, no. 5, pp. 836–844, May 2008.
[55] D. Hunter and B. M. Wilamowski, "Parallel multi-layer neural network architecture with improved efficiency," in *Proc. 4th Conf. Human Syst. Interactions–HSI'11*, Yokohama, Japan, May 19–21, 2011, pp. 299–304.
[56] B. M. Wilamowski, H. Yu, and K. Chung, "Parity-N problems as a vehicle to compare efficiency of neural network architectures," in *Industrial Electronics Handbook*, 2nd ed. Boca Raton, FL: CRC Press, 2011, vol. 5, Intelligent Systems, ch. 10, pp. 10-1–10-8.
[57] T. Sugeno and G. T. Kang, "Structure identification of fuzzy model," *Fuzzy Sets and Systems*, vol. 28, no. 1, pp. 15–33, 1988.
[58] T. T. Xie, H. Yu, and B. M. Wilamowski, "Replacing fuzzy systems with neural networks," in *Proc. IEEE Human Syst. Interaction Conf. –HSI'10*, Rzeszow, Poland, May 13–15, 2010, pp. 189–193.
[59] H. B. Demuth and M. Beale*, Neural Network Toolbox: For Use With MATLAB*. Natick, MA: Mathworks, 2000.
[60] H. Yu and B. M. Wilamowski, "Efficient and reliable training of neural networks," in *Proc. IEEE Human Syst. Interaction Conf.–HSI'09*, Catania, Italy, May 21–23, 2009, pp. 109–115.
[61] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electron. Comput.*, vol. EC-14, pp. 326–334.
[62] T. T. Xie, H. Yu, and B. M. Wilamowski, "Comparison of traditional neural networks and radial basis function networks," in *Proc. 20th IEEE Int. Symp. Ind. Electron.–ISIE'11*, Gdansk, Poland, Jun. 27–30, 2011, pp. 1194–1199.
[63] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Netw.*, vol. 2, no. 2, pp. 302–309, Mar. 1991.
[64] G. B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 57–67, Jan. 2005.
[65] M. Pucci and M. Cirrincione, "Neural MPPT control of wind generators with induction machines without speed sensors," *IEEE Trans. Ind. Electron.*, vol. 58, no. 1, pp. 37–47, Jan. 2011.
[66] Q. N. Le and J. W. Jeon, "Neural-network-based low-speed-damping controller for stepper motor with an FPGA," *IEEE Trans. Ind. Electron.*, vol. 57, no. 9, pp. 3167–3180, Sep. 2010.
[67] K. Chen and A. Salman, "Learning speaker-specific characteristics with a deep neural architecture," *IEEE Trans. Neural Netw.*, vol. 22, no. 11, pp. 1744–1756, 2011.
[68] I. Arel, D. C. Rose, and T. P. Karnowski, "Deep machine learning—A new frontier in artificial intelligence research," *IEEE Comput. Intell. Mag.*, vol. 5, no. 4, pp. 13–18, 2010.
[69] *Intelligent Engineering Systems Through Artificial Neural Networks*. New York: ASME, 1994, vol. 4, pp. 205–209.
[70] *Intelligent Engineering Systems Through Artificial Neural Networks*. New York: ASME, 1993, vol. 3, pp. 175–180, Also in.
[71] *Intelligent Engineering Systems Through Artificial Neural Networks*. New York: ASME, 1993, vol. 3, pp. 175–180.

David Hunter currently is in the Ph.D. program at Auburn University focusing on development of new neural network architectures and training methods. He received his M.S. in computer engineering in 2003 from the University of Idaho. Previously, Mr. Hunter worked for 15 years in the semiconductor industry with a focus in photolithography, metrology, and defects.

Hao Yu (S'10) received the Ph.D. degree in electrical and computer engineering from Auburn University, Auburn, AL, in 2011.

He was a Research Assistant with the Department of Electrical and Computer Engineering, Auburn University. He is currently working in Lattice Semiconductor. His research interests include machine learning, FPGA development, and EDA.

Dr. Yu serves as Reviewer for the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, and the IEEE TRANSACTIONS ON NEURAL NETWORKS.

Michael S. Pukish, III (S'07) is currently working towards the Ph.D. degree at Auburn University, Auburn, AL, studying neural network architectures and training methods and applied problems.

He has developed testing suites comprised of original PCB and FPGA designs for complex system-in-chip RFICs. Previously, he worked as a PCB Designer and analytical tools and application developer for various aerospace, radio astronomy and auditory technology companies and research groups.

Janusz Kolbusz received the M.Sc. degree in informatics and automatics engineering from the Rzeszow University of Technology, Rzeszow, Poland, in 1998, and the Ph.D. degree in telecommunications from the University of Technology and Life Sciences, Bydgoszcz, Poland, in 2010.

Since 1998, he has been working at Information Technology and Management, Rzeszow. His research interests focus on operations systems, computer networks, architectural systems development, quality of service and survivability of the next-generation optical transport networks.

Bogdan M. Wilamowski (SM'83–F'00) received the M.S. degree in computer engineering in 1966, the Ph.D. degree in neural computing in 1970, and the Dr.Habil. degree in integrated circuit design in 1977, all from Gdansk University of Technology, Gdansk, Poland.

He was with the Gdansk University of Technology; the University of Information Technology and Management, Rzeszow, Poland; Auburn University, Auburn, AL; University of Arizona, Tucson; University of Wyoming, Laramie; and the University of Idaho, Moscow. He is currently the Director of the Alabama Micro/Nano Science and Technology Center, Auburn University.

Dr. Wilamowski was the Vice President of the IEEE Computational Intelligence Society (2000–2004) and the President of the IEEE Industrial Electronics Society (2004–2005). He served as an Associate Editor in numerous journals. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS from 2007 to 2010