Computing Gradient Vector and Jacobian Matrix in Arbitrarily Connected Neural Networks

Bogdan M. Wilamowski, *Fellow, IEEE*, Nicholas J. Cotton, Okyay Kaynak, *Fellow, IEEE*, and Günhan Dündar

***Abstract*—This paper describes a new algorithm with neuron-by-neuron computation methods for the gradient vector and the Jacobian matrix. The algorithm can handle networks with arbitrarily connected neurons. The training speed is comparable with the Levenberg–Marquardt algorithm, which is currently considered by many as the fastest algorithm for neural network training. More importantly, it is shown that the computation of the Jacobian, which is required for second-order algorithms, has a similar computation complexity as the computation of the gradient for first-order learning methods. This new algorithm is implemented in the newly developed software, Neural Network Trainer, which has unique capabilities of handling arbitrarily connected networks. These networks with connections across layers can be more efficient than commonly used multilayer perceptron networks.**

***Index Terms*—Learning, neural network.**

I. Introduction

RECENTLY, we have observed an increased interest in applications of neural networks in industrial electronics. In the February 2007 "Special Section on Neural Network Applications in Power Electronics and Motor Drives," 23 papers were published in the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS [1]. Neural networks are useful in many applications such as control of nonlinear systems [2], [3], harmonic detection and compensation [4], [5], load forecasting [6], [7], motor drive [8]–[12], prediction of nonlinear load harmonics [13], maximum power tracking in photovoltaic systems [14], fault detection [15], sensorless control [16]–[19], and diagnosis [20]–[23]. However, most of the research did not exploit the full power of neural networks. In most other cases, standard multilayer perceptron (MLP) networks were utilized [1]–[7], [9]–[13], [15]–[18], [20]–[23], and often [3], [16], only the simplest possible single-layer neural networks, known as ADALINE, were used.

In some cases of industrial electronics applications, fuzzy neural networks (FNNs) were used [8], [14], [20]. FNNs require signal-by-signal multiplications which are not a common feature of biological neurons. They also require an excessive number of neurons in order to follow the large number of fuzzy rules required by FNNs. Other implemented approaches are radial basis function (RBF) networks and principal component analysis (PCA) [23].

PCA uses Hebbian neural networks which have comparably simple architectures as ADALINE networks which naturally limit their ability to process nonlinear signals effectively.

On the other hand, RBF networks handle nonlinear problems well, and are very easy to train, but RBF networks need a hidden neuron for every training pattern. If training patterns are grouped into clusters, then the number of hidden neurons can be reduced to the number of clusters, but even in this case, the number of neurons used in RBF networks turns out to be very large.

Fully connected network (FCN) topologies are relatively powerful, and usually, a fewer number of neurons have to be used than in the case of MLP networks. It will be shown in Section II that FCNs are even more powerful than MLP networks and that they use a smaller number of neurons to fulfill the same task.

One may notice in the literature that, for almost all cases, very simple algorithms, such as least mean square or error back propagation (EBP), are used to train neural networks. These algorithms converge very slowly in comparison to second-order methods, which converge significantly faster. One reason why second-order algorithms are seldom used is their complexity which requires computation of not only gradients but also Jacobian or Hessian matrices.

Various methods of neural network training have already been developed, ranging from the evolutionary computation search through gradient-based methods. The best known method is EBP [24], but this method is characterized by very poor convergence. Several improvements for EBP were developed such as the quickprop algorithm, resilient EBP, back percolation, and delta-bar-delta, but much better results can be obtained using second-order methods [25]–[27] such as Newton or Levenberg–Marquardt (LM) [25]. In the latter one, not only the gradient but also the Jacobian matrix must be found.

This paper presents a new neuron-by-neuron (NBN) method of computing the Jacobian matrix [28]. It is shown that the computation of the Jacobian matrix can be as simple as the computation of the gradient in the EBP algorithm; however, more memory is required for the Jacobian. In the case of a network with the number of training patterns n_p and the number of network outputs n_o, the Jacobian is $n_p \times n_o$ which is of larger dimensions than the gradient and therefore requires more

Manuscript received July 18, 2008; revised July 18, 2008. First published August 19, 2008; current version published October 1, 2008. This work was supported in part by the National Science Foundation international grant, U.S.–Turkey Cooperative Research: Silicon implementation of computational intelligence for mechatronics, under Award NSF OISE 0352771.

B. M. Wilamowski and N. J. Cotton are with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849-5201 USA (e-mail: wilambm@auburn.edu).

O. Kaynak and G. Dündar are with the Department of Electrical and Electronics Engineering, Bogazici University, Istanbul 34342, Turkey.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2008.2003319

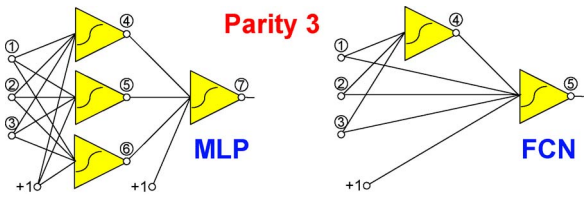

Fig. 1. Network architectures to solve the parity-3 problem with three-layer MLP networks and with three-layer FCNs.

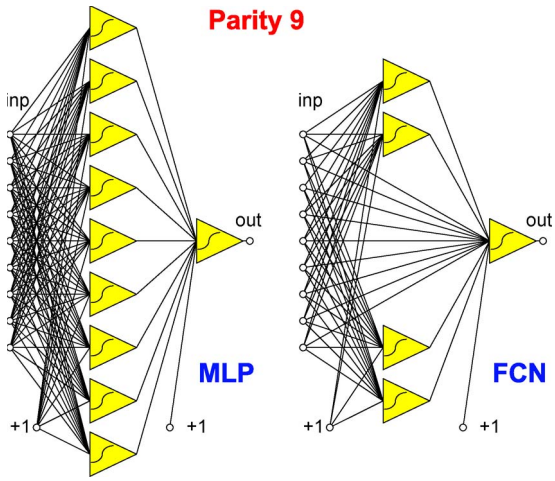

Fig. 2. Network architectures to solve the parity-9 problem with three-layer MLP networks and with three-layer FCNs.

memory. In this sense, the NBN algorithm has the same limitations as the well-known LM algorithm. For example, in the case of 10 000 patterns and neural networks with 25 weights and 3 outputs, the Jacobian $\mathbf{J}$ will have 30 000 rows and 25 columns, all together having 750 000 elements. However, the matrix inversion must be done only for quasi-Hessian $\mathbf{J} \times \mathbf{J}^{\mathrm{T}}$ of 25×25 size.

The structuring of this paper is as follows. In the following section, a discussion is presented on the advantages of networks with arbitrarily connected neurons (ACNs). The NBN algorithm is presented in Section III. Section IV describes the new software implementing the NBN algorithm which can handle ACN networks. Section V presents various experimental results.

II. Advantages of Networks With ACNs

Comparing FCNs with MLP networks, one may conclude that the latter ones require about twice as many neurons to perform a similar task. For example, Figs. 1 and 2 show the minimum architectures required to solve parity-3 and parity-9 problems. It is relatively simple to design neural networks to solve parity-n problems [29]. However, to find a solution by training is much more difficult. For three-layer MLP networks to solve parity-n problems, the required number of neurons is [29]

$$N_{\mathrm{MLP}} = N + 1 \tag{1}$$

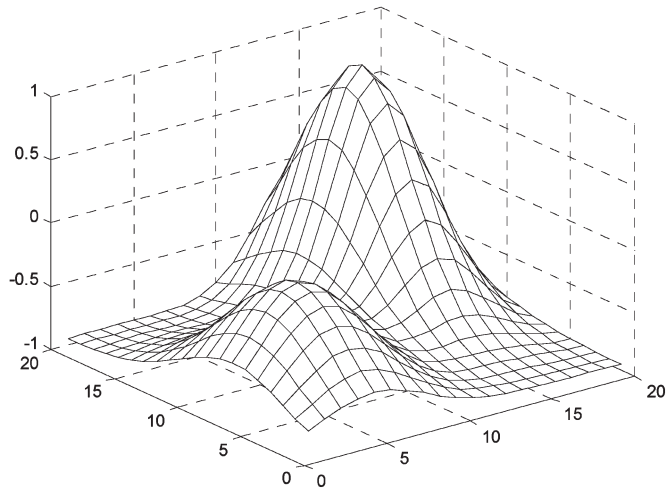

Fig. 3. Desired nonlinear control surface that the neural networks use to train to.

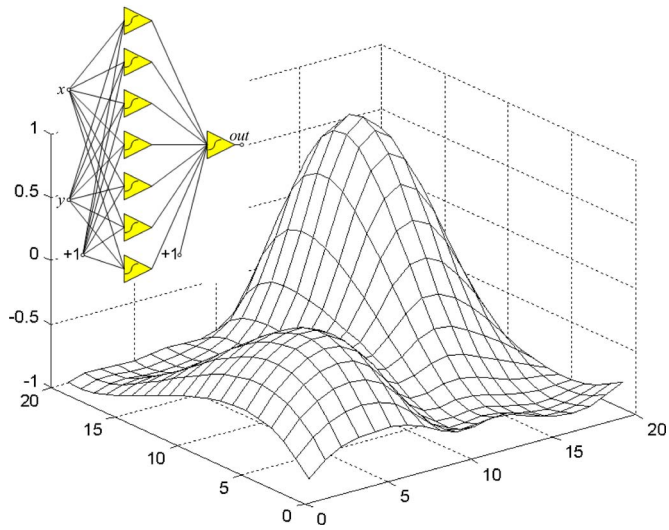

Fig. 4. Resulting control surface obtained with the MLP architecture with seven neurons in one hidden layer. The total mse is 0.00234.

while for three-layer FCNs, the minimum number of neurons is

$$N_{\mathrm{FCN}} = \begin{cases} \frac{N-1}{2}, & \text{for odd-number parity problems} \\ \frac{N}{2}, & \text{for even-number parity problems.} \end{cases} \tag{2}$$

Another example of where an ACN network outperforms a traditionally connected network is in a nonlinear control system. In Fig. 3, a desired highly nonlinear control surface for two variables is shown.

With three-layer eight-neuron MLP networks shown in Fig. 4, it was not possible to reach the desired surface. However, with the five-neuron FCN architecture shown in Fig. 5, it was possible to find a satisfactory solution, with the control surface obtained being very close to the required surface of Fig. 5. One may notice that the FCN topology with five neurons produces significantly smaller error than the eight-neuron MLP topology with one hidden layer. The mean square error (mse) is defined as

$$\mathrm{mse} = \frac{1}{n_p n_o} \sum_{i=1}^{n_p} \sum_{i=1}^{n_o} e_{ij}^2 \tag{3}$$

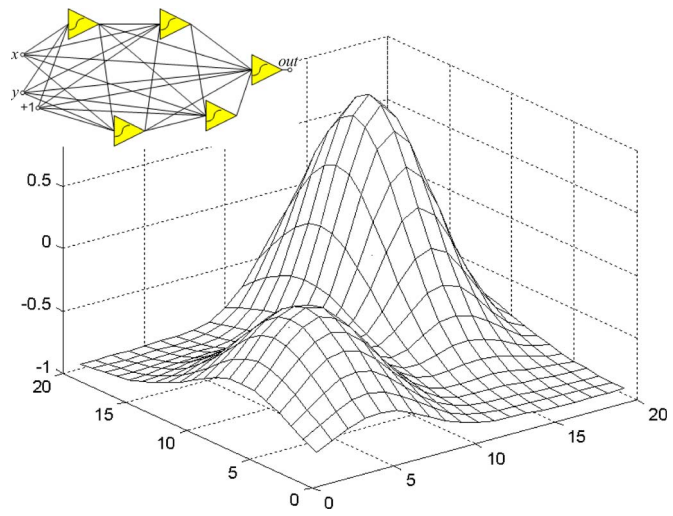

Fig. 5. Resulting control surface obtained with the FCN architecture with four hidden neurons. The total mse is 0.00014.

where

$$e_{ij} = \text{out}_{ij} - \text{dout}_{ij} \tag{4}$$

with dout being the desired output, out being the actual output, n_p being the number of patterns, and n_o being the number of outputs.

Comparing three-layer networks, as shown in Figs. 1 and 2, one may also conclude that FCNs are more transparent than MLP networks. With connections across layers in ACN networks, there are fewer neurons (nonlinear elements) on the signal paths, and as a result, learning algorithms converge faster. Unfortunately, most of the neural network learning software, such as the popular MATLAB Neural Network Toolbox, is developed for MLP networks and are not able to handle FCNs or ACNs. It is also much easier to write computer software for regular architectures, organized layer by layer, in comparison to neural networks with ACN. Both FCN and MLP networks are, of course, a subset of ACN networks.

III. Calculation of Gradient and Jacobian

The EBP algorithm requires only the computation of the error gradient. Second-order algorithms, such as the LM or Davidon–Fletcher–Powel [24], require the computation of the Jacobian. EBP follows the concept of the steepest descent optimization algorithm where the global error is reduced by following the steepest descent path (moving in the opposite direction to the gradient $\mathbf{g}$.) The weight updating rule is

$$\Delta \mathbf{w} = -\alpha \mathbf{g} \tag{5}$$

where α is the experimentally selected "learning constant" and $\mathbf{g}$ is the gradient vector. For the LM algorithm, the weight updating rule is

$$\Delta \mathbf{w} = -(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \mu \mathbf{I})^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{e} \tag{6}$$

where $\mathbf{I}$ is the identity matrix, $\mathbf{e}$ is the error vector with elements given by (4), $\mathbf{J}$ is the Jacobian matrix, and μ is a learning parameter [23], [26]. If the Jacobian $\mathbf{J}$ is known, then the gradient $\mathbf{g}$ can be found as

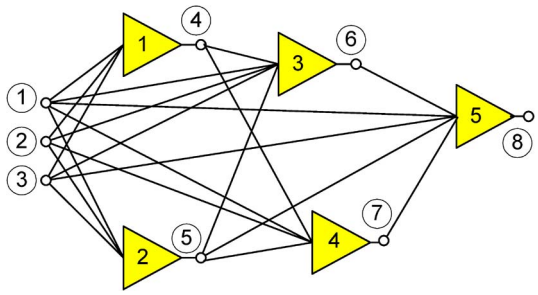

Fig. 6. Example of an ACN network. The network has five neurons numbered from 1 to 5 and eight nodes, three of which are input nodes from 1 to 3 and five of which are neuron nodes from 4 to 8.

$$\mathbf{g} = 2\mathbf{J}^{\mathrm{T}}\mathbf{e}. \tag{7}$$

Therefore, the updates on the weights $\Delta \mathbf{w}$ can be found in both EBP and LM algorithms [(5) and (6)] if the error vector $\mathbf{e}$ and the Jacobian matrix $\mathbf{J}$ are evaluated.

In the Jacobian matrix, each row corresponds to the pth input pattern and the oth network output; therefore, the number of rows of the Jacobian is equal to the product $n_o \times n_p$, where n_p is number of training patterns and n_o is the number of outputs. The number of columns is equal to the number of weights n_w's in the neural network. Every neuron has the number of elements in this row, which is equal to the number of inputs plus one. For the pth pattern, oth output, and nth neuron with K inputs, the fragment of the Jacobian matrix has the form

$$\cdots \quad \frac{\partial e_{po}}{\partial w_{n0}} \quad \frac{\partial e_{po}}{\partial w_{n1}} \quad \frac{\partial e_{po}}{\partial w_{n2}} \quad \frac{\partial e_{po}}{\partial w_{n3}} \quad \cdots \quad \frac{\partial e_{po}}{\partial w_{nK}} \quad \cdots \tag{8}$$

where the weight with index 0 is the biasing weight and e_{po} is the error on the oth network output.

In this paper, a new NBN method for calculating the gradients and the Jacobians for arbitrarily connected feedforward neural networks is presented. The rest of the computations for weight updates follow the LM algorithm. In order to explain the computation algorithm, consider an ACN network with one output, as shown in Fig. 6.

The row elements of the Jacobian matrix for a given pattern are being computed in the following three steps:

1) forward computations;
2) backward computations;
3) calculation of Jacobian elements.

A. Forward Computation

Forward and backward calculations are done using NBN calculations. In the forward calculation, the neurons connected to the network inputs are first processed so that their outputs can be used as inputs to the subsequent neurons. The following neurons are then processed as their input values become available. In other words, the selected computing sequence

has to follow the concept of feedforward networks and the signal propagation. If a signal reaches the inputs of several neurons at the same time, then these neurons can be processed in any sequence. For the network in Fig. 6, there are only four possible ways in which neurons can be processed in the forward direction: 12345, 21345, 12435, or 21435. When the forward pass is concluded, the following two temporary vectors are stored: the first vector $\mathbf{o}$ with the values of the signals on the neuron outputs and the second vector $\mathbf{s}$ with the values of the slopes of the neuron activation functions, which are signal dependent.

B. Backward Computation

The sequence of the backward computation is opposite to the forward computation sequence. The process starts with the last neuron and continues toward the input. In the case of the network of Fig. 6, the following are the possible sequences (backward signal paths): 54321, 54312, 53421, or 53412. To demonstrate the case, let us select the 54321 sequence. The attenuation vector $(\mathbf{a})$ represents signal attenuation from a network output to the outputs of all other neurons. The size of this vector is equal to the number of neurons.

The process starts with the values of one assigned to the last element of the $\mathbf{a}$ vector and zero to the remaining output neurons. During backward processing for each neuron, the value of the delta of this neuron is multiplied by the slope of the neuron activation function (the element of the $\mathbf{s}$ vector calculated during forward computation) and then multiplied by neuron input weights. The results are added to the other elements of the $\mathbf{a}$ vector neurons which are not yet processed. The second step in the example updates only the elements of the $\mathbf{a}$ vector that are associated with neurons 3 and 4 because only these neurons are directly connected to the inputs of neuron 5. In the next step, neuron 4 is processed, and the elements of the $\mathbf{a}$ vector associated with neurons 1 and 2 are updated. Next, neuron 3 is processed, and again, the elements of the $\mathbf{a}$ vector that correspond to neurons 1 and 2 are updated. There is no reason to continue the process beyond this point because there are no other neurons connected to the inputs of neurons 1 or 2. Results of the backward processing elements of the $\mathbf{a}$ vector are then obtained.

One may notice that backward computation is done only for a limited number of neurons. For example, in the case of the four topologies shown in Figs. 1 and 2, only one output neuron is processed.

The size of the $\mathbf{a}$ vector is equal to the number of neurons, while the number of Jacobian elements in one row is much larger and is equal to the number of weights in the network. In order to obtain all row elements of the Jacobian for the p pattern and oth output, a very simple formula can be used to obtain the element of the Jacobian matrix associated with the input k of neuron n

$$\frac{\partial e_{po}}{\partial w_{nk}} = d(n)_{po} \cdot s(n)_p \cdot \text{node}(k)_{po} \tag{9}$$

where $d(n)$ is the element of the $\mathbf{a}$ vector and $s(n)$ is the slope calculated during forward computation, with both of them being associated with neuron n. node(k) is the value on the kth input of this neuron.

```
n4 mbip 1 2 3
n5 mbip 1 2 3
n6 mbip 1 2 3
n7 mbip 4 5 6
.model mbip fun=bip, gain=0.3
datafile=parity3.dat
```

```
n4 mbip 1 2 3
n5 mbip 1 2 3  4
.model mbip fun=bip, gain=0.3
datafile=parity3.dat
```

Fig. 7. Input files with network topologies of Fig. 1 (for the parity-3 problem).

C. Calculation of Jacobian Elements

The process is repeated for every pattern, and if a neural network has several outputs, it is also repeated for every output. The process of gradient computation in the ACN network is exactly the same, but instead of storing values in the Jacobian matrix, they are being summed into one element of the gradient vector

$$g(n,k) = \sum_{p=1}^{P} \sum_{o=1}^{O} \frac{\partial e_{po}}{\partial w_{nk}} e_{po}. \tag{10}$$

If the Jacobian is already computed, then the gradient can also be calculated using (5). The latter approach, with Jacobian calculation, has similar computation complexity, but it requires much more memory to store the Jacobian matrix.

IV. Software Implementation

The NBN computation algorithm is implemented in the newly developed software—Neural Network Trainer (NNT). Several modifications of second-order algorithms have been experimented with the use of the NBN computation scheme. One difficulty faced has been to find a way to describe the network topology for ACN, and this is solved by following the element-by-element concept implemented in the SPICE program. This way, the computation method is made fully compatible with the input-file structure because the software also processes NBN using a restricted sequence as specified in the input file.

In order to train a neural network, the following two files have to be prepared: one with the network topology and the other with the training patterns. If it is desired to experiment with several architectures, only the topology file has to be modified. Fig. 7 shows the input for the first neural network shown in Fig. 1 (parity-3 problem with three neurons). One may notice that the neuron number is the same as the number of the node connected to the neuron output. In this example, numbers 1, 2, and 3 are reserved for three input nodes. The model line defines the bipolar activation function with a gain of 0.3. The last line shows the name of the data file for the parity-3 problem with binary values in Fig. 8.

In the data file for all patterns, first, the input values are listed, followed by the output values. Since the first neuron listed in the input file has number 4 (see Fig. 7), the first three values in each row of Fig. 8 must be the input values, and the remaining value(s) is/are the output value(s). Moreover, the

```
-1 -1 -1  -1
-1 -1  1   1
-1  1 -1   1
-1  1  1  -1
 1 -1 -1   1
 1 -1  1  -1
 1  1 -1  -1
 1  1  1   1
```

Fig. 8. Data file for the parity-3 problem.

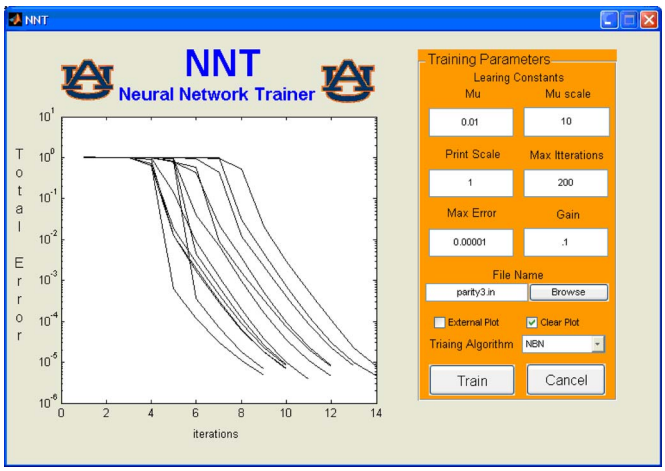

Fig. 9. User interface of the NNT, where the NBN algorithm was implemented.

software is able to automatically distinguish between inputs and outputs within the data set (without any special characters) by the specified architecture in the input file.

The software is developed in the MATLAB environment, but not an element of the Neural Network Toolbox has been used. The user interface is shown in Fig. 9. In order to train the neural network, two input files (Figs. 7 and 8) have to be prepared first. The algorithm and the training parameters can then be selected from the user interface shown in Fig. 9. The software can be downloaded from http://www.eng.auburn.edu/~wilambm/nnt/.

V. Experimental Results

Several simulations were run to test the architectures and algorithms. Figs. 10 and 11 show a comparison of the EBP and NBN training methods for the parity-3 problem with the MLP architecture shown in Fig. 1. In the experiment, EBP was given preference treatment, allowing EBP to use 2000 times more iterations and 1000 times larger final error (10^{-4} instead of 10^{-7}). In other words, EBP had an advantage of $2000 \times 1000 = 2\,000\,000$ times. The NBN method converged to an error of less than 10^{-7} for every time in less than 30 iterations. EBP required about 6000 iterations for an accuracy of 10^{-4} (i.e., 1000 times worse). In our simulation studies, we have found that EBP had difficulties to converge for any parity problems higher than parity 3.

The results of more complex experiments are shown in Tables I and II. Two types of neural networks, MLP and FCN

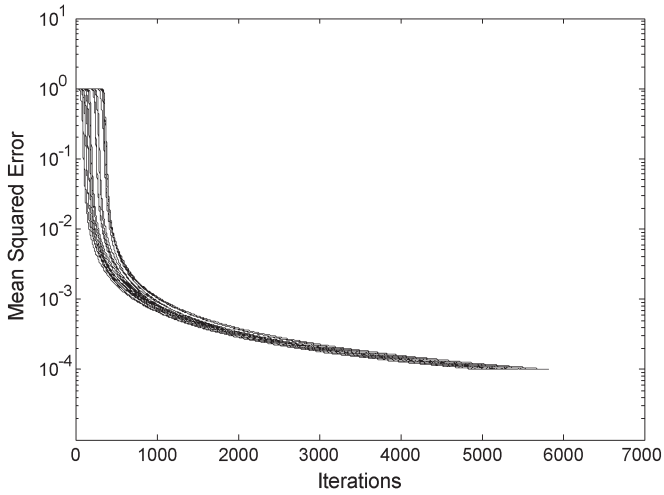

Fig. 10. MSE as a function of number of iterations for EBP training of the FCN of Fig. 1 for a parity-3 problem; 20 curves correspond to 20 training processes with randomly selected initial weights.

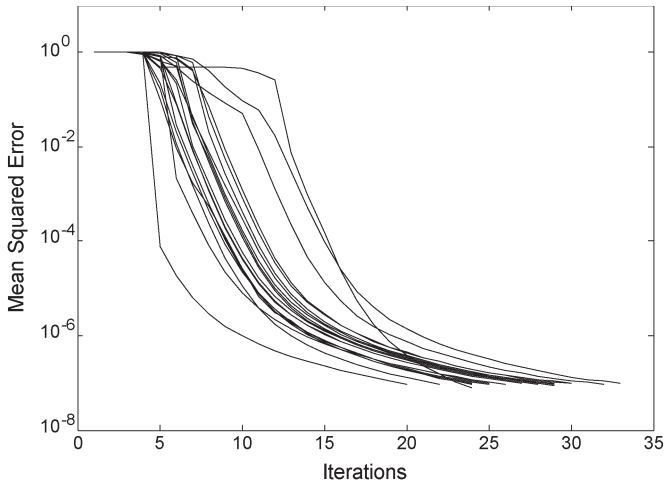

Fig. 11. MSE as a function of number of iterations for NBN training of the FCN of Fig. 1 for a parity-3 problem; 20 curves correspond to 20 training processes with randomly selected initial weights.

TABLE I
Parity-3 Success Rate (SR) and Average Number of Iterations (ANI) for the NBN Algorithm

Hidden Neurons		1	2	3	4	5
MLP	SR	0%	80%	98%	100%	100%
	ANI	NA	19	10	9	8
FCN	SR	100%	100%	100%	100%	100%
	ANI	7.8	7.2	7.0	6.8	6.7

(see Figs. 1 and 2), were trained for parity-3 and parity-5 problems. For all cases, two types of architectures were used, namely, MLP and FCN. For each time, the neural network was trained 1000 times with randomly started initial weights. The tables show that as more neurons are used, the success rate increases, and the number of iterations required for convergence decreases. This is true for all tested cases. It is possible to train a network to solve the parity-5 case with only three neurons in the hidden layer, but it will successfully train only about

TABLE II
PARITY-5 SUCCESS RATE (SR) AND AVERAGE NUMBER OF ITERATIONS (ANI) USING THE NBN ALGORITHM

Hidden Neurons		2	3	4	5	6	7
MLP	SR	0%	4.9%	42%	74%	89%	96%
	ANI	NA	45	57.5	46.7	34.3	28.5
FCN	SR	24%	57%	69%	86%	95%	95%
	ANI	24	21	19	18	17.7	16.8

5% of the time. However, if more neurons are added, this will drastically increase. For example, with seven hidden neurons, the convergence rate is 96% of the cases. The same is true with FCN, but this type of network is more likely to converge even with fewer neurons and converge in fewer iterations.

NBN converged for all parity problems tested with different success rates starting from 100% for parity 3 to about 5% for parity 11. No tests were attempted beyond parity 11.

VI. CONCLUSION

In this paper, a novel NBN algorithm for the computation of the gradient vector and the Jacobian matrix was presented. With the proposed NBN computation scheme, the Jacobian can be calculated almost as easily as the gradient. This way, second-order learning algorithms can easily be implemented. It is also shown (Figs. 10 and 11) that second-order algorithms require about 1000 times fewer iterations to find an acceptable solution. In second-order algorithms, a square matrix with the size that is equal to the number of weights has to be inverted at every iteration. This significantly slows second-order algorithms for large networks. One of the main advantages of the proposed NBN algorithm, in comparison to the LM algorithm, is that it can handle ACN networks, which, as shown in Section II, are more efficient than MLP networks. Also, from experimental results, one may conclude that ACN and FCN are easier to train than commonly used MLP networks. This paper includes a short description of the newly developed software tool, NNT, on which the proposed NBN algorithm is implemented.

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] C. Kwan and F. L. Lewis, "Robust backstepping control of nonlinear systems using neural networks," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 30, no. 6, pp. 753–766, Nov. 2000.

[3] H. Miyamoto, K. Kawato, T. Setoyama, and R. Suzuki, "Feedback-error-learning neural network for trajectory control of a robotic manipulator," *Neural Netw.*, vol. 1, no. 3, pp. 251–265, 1988.

[4] H. C. Lin, "Intelligent neural network-based fast power system harmonic detection," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 43–52, Feb. 2007.

[5] B. Singh, V. Verma, and J. Solanki, "Neural network-based selective compensation of current quality problems in distribution system," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 53–60, Feb. 2007.

[6] M.-S. Kang, C.-S. Chen, Y.-L. Ke, C.-H. Lin, and C.-W. Huang, "Load profile synthesis and wind-power-generation prediction for an isolated power system," *IEEE Trans. Ind. Appl.*, vol. 43, no. 6, pp. 1459–1464, Nov. 2007.

[7] T. Saksornchai, W.-J. Lee, K. Methaprayoon, J. R. Liao, and R. J. Ross, "Improve the unit commitment scheduling by using the neural-network-based short-term load forecasting," *IEEE Trans. Ind. Appl.*, vol. 41, no. 1, pp. 169–179, Jan. 2005.

[8] R.-J. Wai and C.-C. Chu, "Robust petri fuzzy-neural-network control for linear induction motor drive," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 177–189, Feb. 2007.

[9] T. Pajchrowski and K. Zawirski, "Application of artificial neural network to robust speed control of servodrive," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 200–207, Feb. 2007.

[10] J. O. P. Pinto, B. K. Bose, and L. E. B. da Silva, "A stator-flux-oriented vector-controlled induction motor drive with space-vector PWM and flux-vector synthesis by neural networks," *IEEE Trans. Ind. Appl.*, vol. 37, no. 5, pp. 1308–1318, Sep. 2001.

[11] X.-L. Wei, J. Wang, and Z.-X. Yang, "Robust smooth-trajectory control of nonlinear servo systems based on neural networks," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 208–217, Feb. 2007.

[12] A. Rubaai, M. J. Castro-Sitiriche, M. Garuba, III, and L. Burge, "Implementation of artificial neural network-based tracking controller for high-performance stepper motor drives," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 218–227, Feb. 2007.

[13] J. Mazumdar, R. G. Harley, F. C. Lambert, and G. K. Venayagamoorthy, "Neural network based method for predicting nonlinear load harmonics," *IEEE Trans. Power Electron.*, vol. 22, no. 3, pp. 1036–1045, May 2007.

[14] B. Wilamowski and X. Li, "Fuzzy system based maximum power tracking for PV system," in *Proc. 28th Annu. Conf. IEEE Ind. Electron. Soc.*, Seville, Spain, Nov. 5–8, 2002, pp. 1990–1994.

[15] R. M. Tallam, T. G. Habetler, and R. G. Harley, "Stator winding turn-fault detection for closed-loop induction motor drives," *IEEE Trans. Ind. Appl.*, vol. 39, no. 3, pp. 720–724, May 2003.

[16] M. Cirrincione, M. Pucci, G. Cirrincione, and G.-A. Capolino, "A new TLS-based MRAS speed estimation with adaptive integration for high-performance induction machine drives," *IEEE Trans. Ind. Appl.*, vol. 40, no. 4, pp. 1116–1137, Aug. 2004.

[17] H. S. Ooi and T. C. Green, "Simulation of neural networks to sensorless control of switched reluctance motor," in *Proc. IEEE Power Electron. Variable Speed Drives Conf.*, Sep. 1998, pp. 281–286.

[18] C. A. Hudson, N. S. Lobo, and R. Krishnan, "Sensorless control of single switch-based switched reluctance motor drive using neural network," *IEEE Trans. Ind. Electron.*, vol. 55, no. 1, pp. 321–329, Jan. 2008.

[19] D. S. Reay, T. C. Green, and B. W. Williams, "Application of associative memory neural networks to the control of a switched reluctance motor," in *Proc. IEEE IECON*, Nov. 1993, vol. 1, pp. 200–206.

[20] M. S. Ballal, Z. J. Khan, H. M. Suryawanshi, and R. L. Sonolikar, "Adaptive neural fuzzy inference system for the detection of inter-turn insulation and bearing wear faults in induction motor," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 250–258, Feb. 2007.

[21] B. 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.

[22] S. Khomfoi and L. M. Tolbert, "Fault diagnostic system for a multilevel inverter using a neural network," *IEEE Trans. Power Electron.*, vol. 22, no. 3, pp. 1062–1069, May 2007.

[23] J. F. Martins, V. Ferno Pires, and A. J. Pires, "Unsupervised neural-network-based algorithm for an on-line diagnosis of three-phase induction motor stator fault," *IEEE Trans. Ind. Electron.*, vol. 54, no. 1, pp. 259–264, Feb. 2007.

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

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

[26] S. Abid, A. Mouelhi, and F. Fnaiech, "Accelerating the multilayer perceptron learning with the Davidon Fletcher Powell algorithm," in *Proc. IJCNN*, Vancouver, BC, Canada, Jul. 16–21, 2006, pp. 3389–3394.

[27] B. M. Wilamowski, "Neural network architectures and learning," in *Proc. ICIT*, Maribor, Slovenia, Dec. 10–12, 2003, pp. TU1–TU12.

[28] B. Wilamowski, N. Cotton, O. Kaynak, and G. Dundar, "Method of computing gradient vector and Jacobean matrix in arbitrarily connected neural networks," in *Proc. IEEE ISIE*, Vigo, Spain, Jun. 4–7, 2007, pp. 3298–3303.

[29] B. M. Wilamowski and D. Hunter, "Solving parity-n problems with feedforward neural network," in *Proc. IJCNN*, Portland, OR, Jul. 20–23, 2003, pp. 2546–2551.

Bogdan M. Wilamowski (M'82–SM'83–F'00) received the M.S. degree in computer engineering, the Ph.D. degree in neural computing, and the Dr. Habil. degree in integrated circuit design in 1966, 1970, and 1977, respectively.

He received the title of Full Professor from the President of Poland in 1987. He was the Director of the Institute of Electronics (1979–1981) and the Chair of the Solid State Electronics Department (1987–1989), Technical University of Gdansk, Gdansk, Poland. He was/has been a Professor with the Gdansk University of Technology, Gdansk (1987–1989), the University of Wyoming, Laramie (1989–2000), the University of Idaho, Moscow (2000–2003), and Auburn University, Auburn, AL (2003–present), where he is currently the Director of the Alabama Micro/Nano Science and Technology Center and a Professor with the Department of Electrical and Computer Engineering. He was also with the Research Institute of Electronic Communication, Tohoku University, Sendai, Japan (1968–1970), and the Semiconductor Research Institute, Sendai (1975–1976), Auburn University (1981–1982 and 1995–1996), and the University of Arizona, Tucson (1982–1984). He is the author of four textbooks and about 300 refereed publications and is the holder of 28 patents. He was the Major Professor for over 130 graduate students. His main areas of interest include computational intelligence and soft computing, computer-aided design development, solid-state electronics, mixed- and analog-signal processing, and network programming.

Dr. Wilamowski was the President of the IEEE Industrial Electronics Society (2004–2005). He was an Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS, the IEEE TRANSACTIONS ON EDUCATION, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the *Journal of Intelligent and Fuzzy Systems*, the *Journal of Computing*, the *International Journal of Circuit Systems*, and the *IES Newsletter*. Currently, he is the Editor-in-Chief of the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.

Nicholas J. Cotton received the M.S. degree in electrical engineering from Auburn University, Auburn, AL, where he is currently working toward the Ph.D. degree in electrical engineering.

He teaches undergraduate courses and is a Research Assistant with the Department of Electrical and Computer Engineering, Auburn University. He has also worked as an Electrical Engineer with Dynetics Inc., Huntsville, AL. His main interests include computational intelligence, neural networks, embedded systems, and cooperative robotics.

Dr. Cotton is a Reviewer for the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS.

Okyay Kaynak (M'80–SM'90–F'03) received the B.Sc. (with first-class honors) and Ph.D. degrees in electronic and electrical engineering from the University of Birmingham, Birmingham, U.K., in 1969 and 1972, respectively.

From 1972 to 1979, he held various positions within industry. Since 1979, he has been with the Department of Electrical and Electronics Engineering, Bogazici University, Istanbul, Turkey, where he is currently a Full Professor, holding the UNESCO Chair on Mechatronics. He has held long-term (for nearly or more than a year) Visiting Professor/Scholar positions at various institutions in Japan, Germany, the U.S., and Singapore. His current research interests include intelligent control and mechatronics. He has authored three books and edited five and authored/coauthored more than 200 papers that have appeared in various journals and conference proceedings.

Dr. Kaynak was the President of the IEEE Industrial Electronics Society during 2002–2003. He is active in international organizations and has served on many committees of the IEEE.

Günhan Dündar was born in Istanbul, Turkey, in 1969. He received the B.S. and M.S. degrees in electrical engineering from Bogazici University, Istanbul, Turkey, in 1989 and 1991, respectively, and the Ph.D. degree in electrical engineering from Rensselaer Polytechnic Institute, Troy, NY, in 1993.

In 1994, he lectured at Bogazici University, teaching courses on electronics, electronics laboratory, IC design, electronic design automation, and semiconductor devices. From August 1994 to November 1995, he was with the Turkish Navy and taught courses on electronics, electronics laboratory, and signals and systems at the Turkish Naval Academy, Istanbul. Since 1995, he has been with Bogazici University, where he is currently a Professor and the Chairman of the Department of Electrical and Electronics Engineering. Between 2002 and 2003, he was with the École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, on sabbatical leave. His research interests include analog integrated-circuit design, computer-aided design for analog design, and soft-computing circuits.