Efficient and Reliable Training of Neural Networks

Hao Yu†, Bogdan M. Wilamowski, *Fellow*, *IEEE* †
†Electrical and Computer Engineering, Auburn University, Alabama, US
hzy0004@auburn.edu wilam@ieee.org

***Abstract* — This paper introduces a neural network training tool, NBN 2.0, which is developed based on neuron by neuron computing method [1][2]. Error backpropagation (EBP) algorithm, Levenberg Marquardt (LM) algorithm and its improved versions are implemented in two different computing methods, traditional forward-backward computation and newly developed forward-only computation. The software can handle not only conventional multilayer perceptron (MLP) networks, but also arbitrarily connected neuron (ACN) networks. Several examples are presented to explain how to use this tool for neural network training. The software is developed based on Visual Studio platform using C++ language and it is available for everyone on the website.**

***Keywords* — neural networks, training tool**

I. BACKGROUND

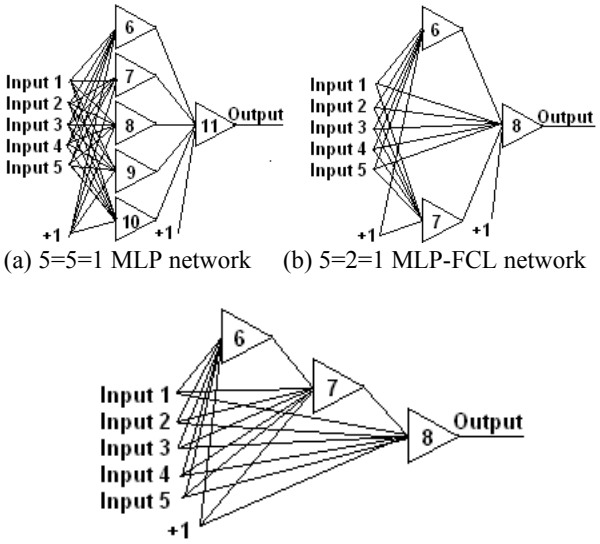
ARTIFICIAL neural networks (ANN) are used in many industrial applications, such as nonlinear control [3][4], system analysis and diagnosis [5][6], VLSI design [7][8] and data classification [9]. It is easy to demonstrate that they can surpass human capabilities to recognize complex patterns [10][11].

Error backpropagation (EBP) algorithm [12] is the most popular training method, however, it is not an efficient one because of its slow convergence and low training stability. Based on EBP algorithm, lots of improvements are developed for better training [13][14], and some of them, such as momentum [15], Quickprop [16] and Resilient EBP [17] work well.

Although complex computation for Jacobian/Hessian matrix is necessary during training process, Levenberg Marquardt (LM) algorithm [18] is considered as one of the most efficient algorithms for small and median sized patterns training. Also, there are many good improved algorithms [19] based on LM algorithm for better training.

Most of neural network training software which uses LM algorithm (e.g. MATLAB Neural Network Toolbox) is not able to train neural networks with arbitrary connections between neurons. This deficiency was overcome by the NBN algorithm [1].

Like EBP algorithm, multilayer perceptron (MLP) networks are broadly accepted in practical applications because they could be realized easily by programming. However, MLP networks are not efficient for neural topology design. Some arbitrarily connected neuron networks, such as MLP with full connections among layers (MLP-FCL) and fully connected neuron (FCN) networks often cost less neurons than MLP networks to solve the same problems. For example, for parity-5 problem, it needs at least 6 neurons for the standard MLP network to get a solution (Fig. 1(a)), while, for MLP-FCL and FCN networks, only 3 neurons are required (Fig. 1(b) and (c)). For efficient training, MLP-FCL and FCN networks are wiser choices, but they also require more challenging computation.



(a) 5=5=1 MLP network (b) 5=2=1 MLP-FCL network

(c) 3 neurons in FCN network

Fig. 1. The least neurons required in different neural network structures for parity-5 problem.

In this paper, the software NBN 2.0 is introduced as a powerful training tool. It contains EBP algorithm, LM algorithm, neuron by neuron (NBN) algorithm [1][2] and a new algorithm. Besides conventional forward-backward computation, a newly developed forward-only [20] (without error backpropagation process) computation is also implemented in the software. Based on the neuron by neuron computing scheme, this tool can handle arbitrarily connected neuron (FCN) networks.

In section II of this paper, the detailed information of NBN 2.0 is described, and section III presents several practical applications with the NBN 2.0.

978-1-4244-3960-7/09/ $25.00 2009 IEEE

II. DESCRIPTION OF NBN 2.0

The NBN 2.0 is developed based on Visual Studio 6.0 using C++ language. Its main interface is shown in Fig. 2.In the following part of this section, detailed instructions about the software are presented.

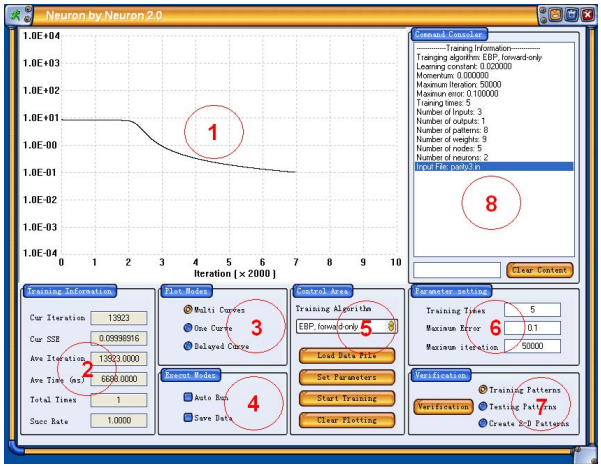


Fig. 2. The user interface of NBN 2.0.

A. Description of files

The software is made up of 6 types of files, including executing files, parameter file (unique), topology files, training pattern files, training result files and training verification files.

- Executing files

Executing files contain three files: files "FauxS-TOON.ssk" and "skinppwtl.dll" for interface design; file "NBN 2.0 exe" for running the software. Also, other files, such as user instruction and accessory tools (Matlab code "PlotFor2D.m" for 2-D plotting), are included.

- Parameter file

This file is named "Parameters.dat" and it is necessary for running the software. It contains initial data of important parameters shown in Table 1.

There are two ways to set those parameters: (1) Edit the parameter file manually, according to the descriptions of parameters in Table 1; (2) All those parameters can be edited in the user interface, and they will be saved in the parameter file automatically once training is executed, as the initial values for next time of running the software.

- Topology files

Topology files are named "*.in", and they are mainly used to construct the neural network topologies for training. Topology files consist of four parts: topology design, weight initialization (optional), neuron type instruction and training data specification.

The topology design is aimed to create neural structures. The general command is "n [*b*] [*type*] [a_1 a_2 ... a_n]", which means inputs/neurons indexed with a_1, a_2 ..., a_n are connected to neuron *b* with a specified neural type (bipolar, unipolar or linear). Fig. 3 presents the topology commands for the neural networks shown in Fig. 1.

The weight initialization part is used to specify initial weights for training and this part is optional. If there is no weight initialization in the topology file, the software will generate initial weights randomly (from -1 to 1) before training. The general command is "w [w_{bias}] [w_1 w_2 ... w_n]", corresponding to the topology design. Fig. 4 shows the example of weight initialization for parity-3 problem with 2 neurons in FCN network.

TABLE 1: PARAMETERS FOR TRAINING

Parameters	*Descriptions*
algorithm	Index of algorithms in the combo box
alpha	Learning constant for EBP
scale	Parameter for LM/NBN
mu	Parameter for LM/NBN
max mu	Parameter for LM/NBN (fixed)
min mu	Parameter for LM/NBN (fixed)
max error	Maximum error
ITE_FOR_EBP	Maximum iteration for EBP
ITE_FOR_LM	Maximum iteration for LM/NBN
ITE_FOR_PO	Maximum iteration for the new Alg.
momentum	Momentum for EBP
po alpha	Parameter for the improved NBN
po beta	Parameter for the improved NBN
po gama	Parameter for the improved NBN
training times	Training times for automatic running

```
//6 neurons in 5=5=1 MLP network
n  6   mbip  1  2  3  4  5
n  7   mbip  1  2  3  4  5
n  8   mbip  1  2  3  4  5
n  9   mbip  1  2  3  4  5
n  10  mbip  1  2  3  4  5
n  11  mbip  6  7  8  9  10
```

(a) 5=5=1 MLP network of Fig. 1(a)

```
//3 neurons in 5=2=1 MLP-FCL network
n  6  mbip  1  2  3  4  5
n  7  mbip  1  2  3  4  5
n  8  mbip  1  2  3  4  5  6  7
```

(b) 5=2=1 MLP-FCL network of Fig. 1(b)

```
//3 neurons in FCN network
n  6  mbip  1  2  3  4  5
n  7  mbip  1  2  3  4  5  6
n  8  mbip  1  2  3  4  5  6  7
```

(c) 3 neurons FCN network of Fig. 1(c)

Fig. 3. Topology design for networks shown in Fig. 1; all neurons are bipolar neurons.

```
//topology design
n 4 mbip 1 2 3
n 5 mbip 1 2 3 4

//weight initialization
w 0.11 0.12 0.13 0.14
w 0.15 0.16 0.17 0.18
```

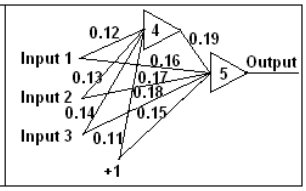


Fig. 4. Weight initialization for parity-3 problem with 2 neurons in FCN network.

In the neuron type instruction part, three different types of neurons are defined. They are bipolar ("mbip"), unipolar (mu) and linear ("mlin"). Bipolar neurons have positive or negative outputs, while unipolar neurons only have positive outputs. The outputs of both bipolar and unipolar neurons

are no more than 1. If the desired outputs are larger than 1, linear neurons are considered to be the output neurons. The general command is ".model [*mbip/mu/mlin*] fun=[*bip/uni/lin*], gain=[*value*], der=[*value*]". Table 2 presents the three types of neurons used in the software.

TABLE 2: THREE TYPES OF NEURONS IN THE SOFTWARE

Neuron Types	***Activation Functions***
bipolar	$f_b(net) = \tanh(gain \times net) + der \times net$
unipolar	$f_u(net) = \dfrac{1}{1+e^{-gain \times net}} + der \times net$
linear	$f_l(net) = gain \times net$

From Table 2, it can be seen that "gain" and "der" are parameters of activation functions. Parameter "der" is introduced to adjust the slope of activation function (for unipolar and bipolar), which is a trick we used in the software to avoid training process entering the saturation region, where slope is approching to zero.

The training data specification part is used to set the name of training pattern file, in order to get correct training data. The general command is "datafile=[*file name*]".

- Training pattern files

The training pattern files include input patterns and related desired outputs. In a training pattern file, the number of rows is equal to the number of patterns, while the number of columns is equal to the sum of the number of inputs and the number of outputs. However, only with the data in training pattern file, one can't tell the number of inputs and the number of outputs, so the neural topology should be considered together in order to decide those two parameters (Fig. 5). The training pattern files are specified in the topology files as mentioned above, and it should have the same route as related topology files.

training data	*topology*	*explanation*
-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	//2 inputs and 2 outputs n 3 mbip 1 2 n 4 mbip 1 2	In command "n 3 mbip 1 2", the index of the first neuron is 3, so the number of inputs is 2, and the number of outputs is 2
	//3 inputs and 1 output n 4 mbip 1 2 3 n 5 mbip 1 2 3 4	In command "n 4 mbip 1 2 3", the index of the first neuron is 4, so the number of inputs is 3, and the number of outputs is 1

Fig. 5. Get the number of inputs and the number of outputs from the data file and topology.

As described in Fig. 5, the number of inputs is obtained from the first command line of topology design and it is equal to the index of the first neuron minus 1. After that, the number of outputs is calculated by the number of columns in training pattern files minus the number of inputs.

- Training result files

Training result files are used to store the training information and results. Once the "save data" function is enabled in the software, important information for current training, such as training algorithm, training pattern file, topology, parameters, initial weights, result weights and training results will be saved after the training is finished. The name of the training result file is generated automatically according to the starting time and the format is "*date_time*_result.txt".

- Training verification files

Training verification files are generated by the software when the verification function is triggered. The result weights from the current training will be verified, by computing the actual outputs of related patterns. The name of training verification file is also created by the system time when the verification starts and it is "*date_time*_verification.txt".

B. Interface instruction

As shown in Fig. 2, the user interface consists of 6 areas: (1) Plotting area; (2) Training information area; (3) Plot modes setting area; (4) Execute modes setting area; (5) Control area; (6) Parameter setting area; (7) Verification area; (8) Command consoler area.

- Plotting area

This area is used to depict the sum squared error (SSE) during training. The log scaled vertical axis presents SSE values from 0.0001 to 10000, while the horizontal axis, which is linearly scaled automatically with the coefficient at the bottom of plotting area ("×[*value*]" in Fig. 2), shows the number of iterations cost for training.

- Training information area

Instantaneous training data are presented in this area, including SSE and cost iterations for current training, average iteration and time spent in solving the same problems, and the success rate for multiple times training.

- Plot modes setting area

Three plot modes are available in this software, multi curves, one curve and delayed curve. In multi curves mode, all the training curves will be plotted together and updated instantaneously. In one curve mode, only current training is plotted, while other curves will be erased. The delayed curve mode is only used in automatic training for multiple times; during the training process, there is no plotting, while all the curves will be presented together after the whole training process is finished. This function is designed for training process which needs huge iterations and costs time for plotting.

- Execute modes setting area

This area is used to control training mode, either training one time or automatic training for several times, either saving the training results or not.

If it is set to run automatically, the train will not stop unless it reaches the required training times or the "Stop to Train" button is clicked. The default training times is 100 and it can be changed through command consoler.

If it is set to save training data, all the important information, such as algorithm type, topology, training parameters, initial weights, result weights and training results (SSE and cost iteration) will be saved in training result files.

- Control area

The combo box is used to select training algorithms. There are 6 choices: (1) "EBP"; (2) "LM"; (3) "NBN"; (4) "EBP, forward-only"; (5) "NBN, forward-only"; (6) "NBN, improved".

Button "Load Data File" is used to choose a topology file for training; button "Set Parameters" is used to set training parameters for the selected training algorithm; button "Start To Train/Stop To Train" helps control the training process; button "Clear Plotting" is used to erase the current plotting in the plotting area.

- Parameter setting area

This area is used to set training related parameters, such as training times for automatic training, maximum error for convergent judgment and maximum iteration. All the settings will be saved in the parameter file once training is executed, and they will be loaded as the initial values for the next time using the software.

- Verification area

This area is used for training results verification, by calculating the actual outputs for each pattern with the result weights. The verifying patterns can be training patterns, testing patterns or user created patterns for 2-D situation. Training patterns are from the data file for current training (Fig. 6), while testing patterns are loaded from data files having the same format as the training data file. For the case of 2-D inputs, the patterns are generated from selected range of each input with a certain number of points (see two-spiral problem verification).

The verification results can be easily uploaded by Matlab, MS Excel, Origin, or other software for analysis. For 2-D input patterns, the verification data can be plotted in Matlab by the accessory tool included in the software, named as "PlotFor2D.m".

-1.00	-1.00	-1.00	-1.0000	-0.8671	-0.1329
-1.00	-1.00	1.00	1.0000	0.8962	0.1038
-1.00	1.00	-1.00	1.0000	0.8962	0.1038
-1.00	1.00	1.00	-1.0000	-0.8962	-0.1038
1.00	-1.00	-1.00	1.0000	0.8962	0.1038
1.00	-1.00	1.00	-1.0000	-0.8962	-0.1038
1.00	1.00	-1.00	-1.0000	-0.8962	-0.1038
1.00	1.00	1.00	1.0000	0.8671	0.1329

Fig. 6. A list of verification results of parity-3 problem by training patterns; the first 3 columns are inputs; the fourth column is the desired output; the fifth column is the actual output; the sixth column is the error from desired output minus related actual output.

The verification can be also for networks with multiple outputs, and the results are presented in the format: "*input columns desired output one actual output one error one desired output two actual output two error two ....*".

Verification results will be stored in training verification files as introduced above and they will appear as pop-up windows automatically.

- Command consoler area

The list box is used to show the important information or hints for users' operations. It is also a command consoler. Most of the operations can be achieved by related commands. Table 3 presents the available commands and their functions.

TABLE 3: AVAILABLE COMMANDS AND RELATED FUNCTIONS

Commands	*Functions*
help	list all the available commands and instructions
clr	clear the content of the list box
cc	clear all the curves in plotting area
sus	suspend training and save current status
res	resume training with the status at suspending point
tra	start/stop training
sav	data saving control
aut	automatic training control
pm=	select plotting mode, e.g. pm=2 (one curve mode)
load	load input file
para	set training parameters for selected algorithm, e.g. para
info	show current training setting
iw	show current iw value (used for debug)
topo	show current topology
alpha? / alpha=	get/set learning constant, e.g. alpha=1
mom? / mom=	get/set momentum, e.g. mom=0.001
tt? / tt=	get/set training times for automatic training, e.g. tt?
me? / me=	get/set maximum error, e.g. me=0.01
mi? / mi=	get/set maximum iteration, e.g. mi=100
th? / th=	get/set parameter for the "NBN, improved" algorithm

C. Implemented algorithms

As introduced above, there are 6 available algorithms in the software for training. The following part is going to introduce the characteristics and limitations for each algorithm.

EBP: This is EBP algorithm with traditional forward-backward computation; for EBP algorithm, it may work a little bit faster than forward-only computation. Now it is only used for standard MLP networks. EBP algorithm converges slowly, but it can be used for huge patterns training.

LM: This is LM algorithm with traditional forward-backward computation; for LM (and NBN) algorithm, the improved forward-only computation performs faster training than forward-backward computation for networks with multiple outputs. Now it is also only used for standard MLP networks. LM (and NBN) algorithm converges much faster than EBP algorithm for small and media sized patterns training. For huge patterns (huge Jacobian matrix, we have solved this problem) and huge networks (huge Hessian matrix), it may work slower than EBP algorithm.

NBN: This is NBN algorithm with forward-backward computation. NBN algorithm is developed based on LM

algorithm, but it can handle arbitrarily connected neuron (ACN) networks, also, the convergence is improved [1][2].

EBP, forward-only: This is EBP algorithm with forward-only computation. It can work on arbitrarily connected neuron networks.

NBN, forward-only: This is NBN algorithm with forward-only computation. It can handle arbitrarily connected neuron networks and, as mentioned above, it works faster than "NBN" algorithm, especially for networks with multiple outputs.

NBN, improved: This is a newly developed second order algorithm, implemented with forward-only computation, so it can handle arbitrarily connected neuron networks. In this algorithm, Hessian matrix is inverted only one time per iteration, so this algorithm is supposed to compute faster than LM (and NBN) algorithm which may have several times Hessian matrix inversion per iteration. The train ability (convergence) is also improved in this algorithm. Furthermore, a local minima detector is implemented in this algorithm. When the detector diagnoses that the training is trapped in local minima, all the weights will be regenerated randomly for further training.

III. PRACTICAL APPLICATIONS

Let us use the two-spiral problem as an example to illustrate how to use the software for neural network training. Other practical applications are also presented followed.

A. Two-spiral problem

Two-spiral problem is considered as an efficient evaluation of both training algorithms and neural structures [21]. This problem is aimed to separate two groups of twisted points, as shown in Fig. 7.

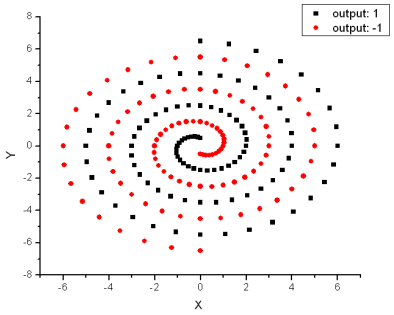


Fig. 7. Two-spiral problem; it's aimed to separate red points from black points.

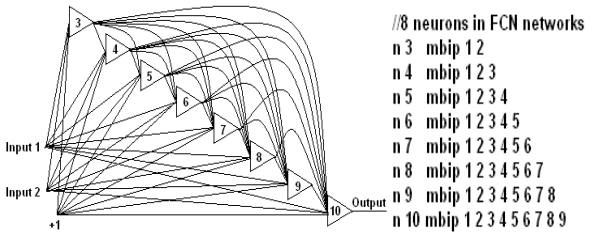
The first step is to generate the training pattern file. Two-spiral data can be obtained by the code shown in Fig. 8.

```
double x,y,angle,radius;
for( int i = 0; i <= 96; i++ )
{
    angle = i*3.1415926/16.0;
    radius = 6.5*(104-i)/104.0;
    x = radius*sin(angle);
    y = radius*cos(angle);
    cout <<x <<"  "<<y <<"  "<< 1 <<endl;
    cout <<-x <<"  "<<-y <<"  "<< -1 <<endl;
}
```

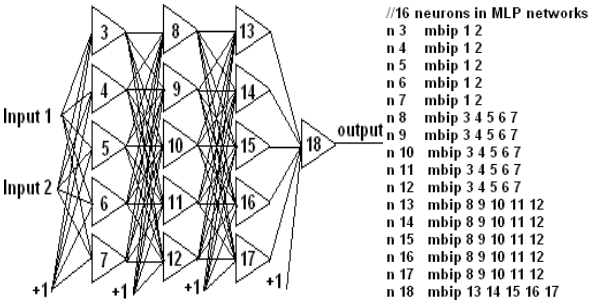
0.00000	6.50000	1.0
0.00000	-6.50000	-1.0
1.25589	6.31381	1.0
-1.25589	-6.31381	-1.0
2.43961	5.88973	1.0
-2.43961	-5.88973	-1.0
3.50704	5.24865	1.0
-3.50704	-5.24865	-1.0
4.41942	4.41942	1.0
-4.41942	-4.41942	-1.0

Fig. 8. Left: C++ code to generate two-spiral patterns; right: part of two-spiral data (10 in 194 patterns totally).

The second step is to create the topology file. For a better explanation, two different topologies are designed as: 8 neurons in FCN networks (Fig. 9(a)) and 16 neurons in 2=5=5=5=1 MLP networks (Fig. 9(b)). All neurons are bipolar because the desired outputs are 1 or -1.



(a) 8 neurons in FCN network



(b) 16 neurons in MLP network

Fig. 9. Different neural structures and related topologies design for two-spiral problem.

The third step is to train the two-spiral patterns with the designed topologies. "NBN, improved" algorithm is used for the topology in Fig. 9(a), while "NBN" is used for Fig. 9(b). After loading topology file and setting related parameters, the training can begin.

The last step is to do verification for the training results. Since two-spiral problem has 2 inputs, the "Created 2-D Patterns" can be used for verification. Fig. 10 presents the plotting in Matlab (using "PlotFor2D.m") with the verification results, by setting the ranges of X and Y both from -6.5 to 6.5, and 1300 points for each dimension.

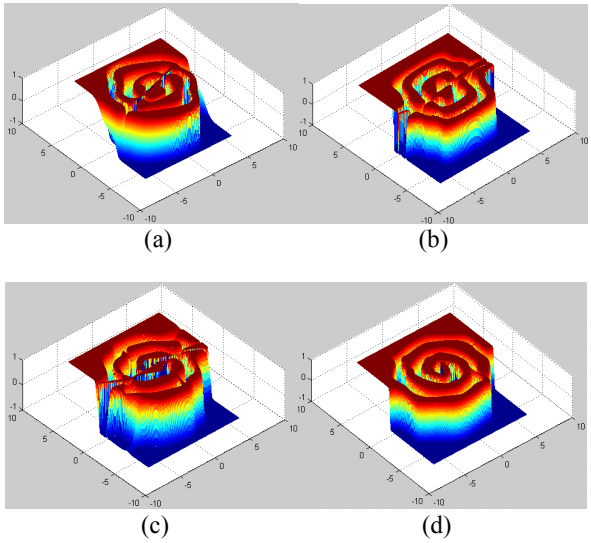
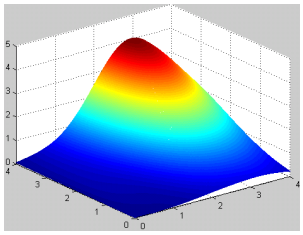


Fig. 10.Verification results of two-spiral problem using different training structures and algorithms: (a) 8 neurons in FCN network, "NBN, improved" algorithm; (b) 16 neurons in 2=5=5=5=1 MLP network, "NBN" algorithm; (c) 16 neurons in 2=5=5=5=1 MLP-FCL network, "EBP, forward-only" algorithm; (d) 15 neurons in FCN network, "NBN, forward-only" algorithm.

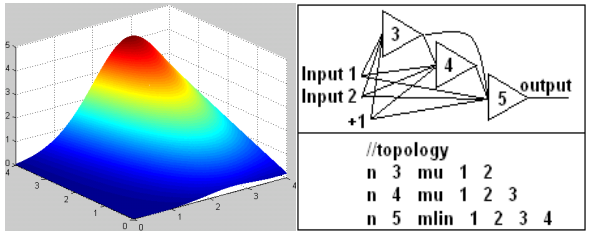
B. *Function approximation*

With the function described by (1), 25 points (x, y and z) are picked out from 0 to 4 as the training data saved in training pattern file. The purpose of the training is to set up a neural network with outputs approximating to the results (z values) of (1) if the inputs (x and y) are the same. A training result from the smallest topology for this problem is presented in Fig. 11.

$$z = 4\exp\left(-0.15(x-4)^2 - 0.5(y-3)^2\right) + 10^{-9} \quad (1)$$



(a) Desired surface, depicted by (1)



(b) Approximated surface, SSE = 0.24997

Fig. 11. Training results of the function approximation problem.

With the training results above, one may notice that the trained neural network in Fig. 11(b) can perform a very similar computation which it is done by function (1). Therefore, in the case that there are not specified equations, but only data, a proper approximation can be made by neural networks trained with the given data.

C. *Parity-N problems*

Parity-N problems are aimed to associate n-bit binary input data with their parity bits. It is also considered to be one of the most difficult problems in neural network training, although it has been solved analytically [22]. Fig. 12 shows the verification of parity-2 problem (also called XOR problem). Experimental results for more complex parity-N problems are presented in Table 4, using fully connected neural networks with "NBN, improved" algorithm.

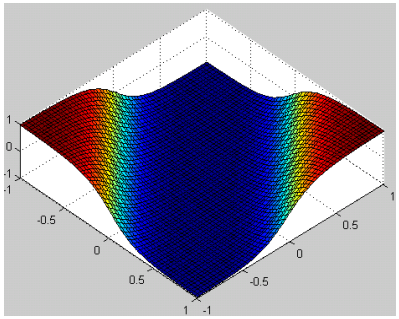


Fig. 12. Verification of the parity-two problem, with 2500 points from -1 to 1.

TABLE 4: TRAINING RESULTS OF PARITY-N PROBLEMS

Parity-N (neurons)	***4(3)***	***6(4)***	***8(5)***	***10(6)***	***12(8)***
Cost Iterations	10.5	23.6	41.4	43.2	78.3
Compute Time (ms)	166	373	758	1636	23943
Success Rate	1.00	1.00	1.00	1.00	1.00

Parameter settings: maximum error – 0.1; maximum iteration – 500; training times – 100; local minima detector – enabled with accuracy level 2.

D. *Error correction*

Error correction is an extension of parity-N problems for multiple parity bits. In Fig. 13, the left side is the input data, made up of signal bits and their parity bits, while the right side is the related corrected signal bits and parity bits as outputs, so number of inputs is equal to the number of outputs.

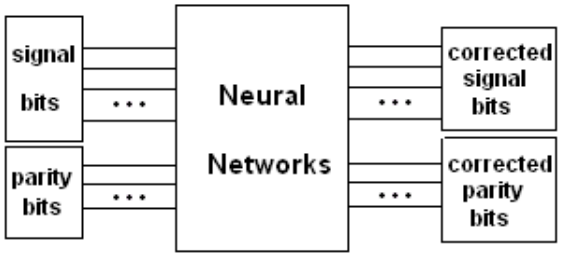


Fig. 13. Using neural networks to solve error correction problems; errors in input data can be corrected by well trained neural networks.

For the example, the problem consists of 4-bit signal with its 3-bit parity bits as inputs, 7 outputs and 128 patterns (16 correct patterns and 112 patterns with errors). Error patterns with one incorrect bit must be corrected. Using 29 neurons in fully connected neural network, with "NBN" algorithm, the training can converge in 84.2 averaged iterations and the success rate is nearly 36%.

With the trained neural network, all the patterns with one bit error are corrected successfully.

E. Image association

A simple example is presented to illustrate how to use NBN 2.0 for image association problems. Fig. 14 shows the digit images, each of which is made up of 56 pixels (8×7). The purpose is to associate images with related digits. The training patterns can be generated as the color of pixels, e.g. "1" is used for black, while "-1" for white. Therefore, in this problem, there are 56 inputs and 10 outputs. Using "NBN, forward-only" algorithm, with the topology 56=10 MLP network, the training is converged in 5 iterations, and the associations are all correct.

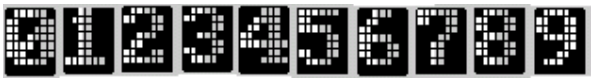


Fig. 14. Digit images with 8×7 pixels from 0 to 9.

IV. CONCLUSION

In this paper, the software NBN 2.0 is introduced for neural network training. This software contains both first order and second order training algorithms, which are implemented by traditional forward-backward computation and a newly developed forward-only computation respectively. It can handle not only MLP networks, but also ACN networks well. With the detailed instructions and several examples presented in the paper, one can get familiar with this useful tool for neural network training. The NBN 2.0 is available at:

http://www.eng.auburn.edu/users/wilambm/nnt/

And also, all the data of the examples presented in this paper are included in the software package.

REFERENCES

[1] B. M. Wilamowski, N. Cotton, J. Hewlett, O. Kaynak, "Neural network trainer with second order learning algorithms". *Proc. International Conference on Intelligent Engineering Systems*, June 29 2007-July 1 2007, pp. 127-132.

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

[3] J. A. Farrell, M. M. Polycarpou, "Adaptive Approximation Based Control: Unifying Neural, Fuzzy and Traditional Adaptive Approximation Approaches, " *IEEE Trans. on Neural Networks,* vol. 19, no. 4, pp. 731-732, April 2008.

[4] G. Colin, Y. Chamaillard, G. Bloch, G. Corde, "Neural Control of Fast Nonlinear Systems—Application to a Turbocharged SI Engine With VCT," *IEEE Trans. on Neural Networks,* vol. 18, no. 4, pp. 1101-1114, April 2007.

[5] S. Khomfoi, 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.

[6] J. F. Martins, V. Ferno Pires, 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.

[7] K. Cameron, A. Murray, "Minimizing the Effect of Process Mismatch in a Neuromorphic System Using Spike-Timing-Dependent Adaptation," *IEEE Trans. on Neural Networks,* vol. 19, no. 5, pp. 899-913, May 2008.

[8] G. Indiveri, E. Chicca, R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *IEEE Trans. on Neural Networks,* vol. 17, no. 1, pp. 211-221, Jan 2006.

[9] M. Kyperountas, A. Tefas, I. Pitas, "Weighted Piecewise LDA for Solving the Small Sample Size Problem in Face Verification," *IEEE Trans. on Neural Networks,* vol. 18, no. 2, pp. 506-519, Feb 2007.

[10] Jafarzadegan, M. , Mirzaei, H. , "A new ensemble based classifier using feature transformation for hand recoognition", *Human System Interactions, 2008 Conference on*, pp. 749-754, May, 2008.

[11] Mroczek, T. , Paja, W. , Piatek, L. , Wrzesie, M. ,"Classification and synthesis of medical images in the domain of melanocytic skin lesions", *Human System Interactions, 2008 Conference on*, pp. 705-709, May, 2008.

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

[13] Yinyin Liu, J.A. Starzyk, Zhen Zhu, "Optimized Approximation Algorithm in Neural Networks Without Overfitting," *IEEE Trans. on Neural Networks,* vol. 19, no. 6, pp. 983-995, June 2008.

[14] S. Ferrari, M. Jensenius, "A Constrained Optimization Approach to Preserving Prior Knowledge During Incremental Training," *IEEE Trans. on Neural Networks*, vol. 19, no. 6, pp. 996-1009, June 2008.

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

[16] Fok Hing Chi Tivive, A. Bouzerdoum, "Efficient training algorithms for a class of shunting inhibitory convolutional neural networks," *IEEE Trans. on Neural Networks*, vol. 16, no. 3, pp. 541-556, March 2005.

[17] M. Riedmiller, H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm". *Proc. International Conference on Neural Networks*, San Francisco, CA, 1993, pp. 586-591.

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

[19] A. Toledo, M. Pinzolas, J.J. Ibarrola, G. Lera, "Improvement of the neighborhood based Levenberg-Marquardt algorithm by local adaptation of the learning coefficient," *IEEE Trans. on Neural Networks,* vol. 16, no. 4, pp. 988-992, April 2005.

[20] Bogdan M. Wilamowski, Hao Yu, "Neural Network Learning without Backpropagation". (unpublished)

[21] J. R Alvarez-Sanchez, "Injecting knowledge into the solution of the two-spiral problem". *Neural Compute and Applications*, Vol. 8, pp. 265-272, 1999.

[22] Wilamowski, B.M. Hunter, D. Malinowski, A., "Solving parity-N problems with feedforward neural networks". *Proc. 2003 IEEE IJCNN,* 2546-2551, IEEE Press, 2003.