2010 24th IEEE International Conference on Advanced Information Networking and Applications

Neural Network Trainer through Computer Networks

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

***Abstract* - This paper introduces a neural network training tool through computer networks. The following algorithms, such as neuron by neuron (NBN) [1][2], error back propagation (EBP), Levenberg Marquardt (LM) and its improved versions are implemented in two different computing methods, traditional forward-backward computation and newly developed forward-only computation. The training tool can handle not only conventional multilayer perceptron (MLP) networks, but also arbitrarily connected neuron (ACN) networks. There are several benefits that make this network training tool desirable. Network training process can be used remotely through any network connection or any operating system can be used to access the network training tool, making the application operating system independent. Also much less installation time and configuration time is required because the training tool locates on one central machine. Many users can access at the same time. Users can train and see the training results directly through networks. And the most important thing is the software producers can protect their intellectual property when network browsers are used as user interface.**

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

I. INTRODUCTION

Communication through computer networks has become a popular and efficient means of computing and simulation. Most companies and research institutions use networks to some extent on a regular basis [3][4]. Computer networks provide the ability to access all kinds of information which are available from all around the world, and intranet networks provide connectivity for a smaller, more isolated domain like a company or a school. Several tools of computer network programming are available today including Java, CGI (Common Gateway Interface)...CGI scripts utilize PERL, PHP, or other scripting languages.

Several neural network trainer tools are available on the market. One of the freeware available tools is “Stuttgart Neural Network Simulator” is based on widely C platform and is distributed in both executable and source code version [5]. However the installation of this tool requires certain knowledge of compiling and setting up the application. Also, it is based on XGUI that is not freeware and still single type architecture- Unix architecture [6].

The commercial neural network software called “Neurodesigner” is a product of CyberSoft [7]. Even though aiming to attract users by Java based windows interface, but this tool lacks some fundamental algorithms, like Levenberg Marquardt algorithm. MATLAB Neural Network Tool box contains basic training algorithms, such as EBP algorithm and LM algorithm, but they are limited for traditional multilayer perceptron networks, which are inefficient [8]. The commercial price also places a hurdle on educational use of such a tool.

This paper will describe the power of network programming technology used to develop the neural network training tool which can be trained remotely through any computer network. Several network programming tools which are available today, include in Java, CGI, JavaScript, ActiveX, HTML, PERL, PHP [9] while HTML, JavaScript, PHP are used to develop in this particular application. The trainer has a user-friendly GUI (Graphical User Interface) and features include password protection and user accounts, local or remote files for training, training result hyperlinks and Error curve in the form of image.

The PHP network programming language supports for dynamic web-pages. Its communication with web browser is accomplished through network connection. PHP has capability to process, extract data and receive request to execute the external execution files in the background. When the execution files finish, it will send the results back to the requesting clients. Moreover, HTML and JavaScript can be incorporated into PHP scripts very efficiently. This feature makes PHP more powerful for the web programming. HTML and JavaScript provide the front-end graphical user interface that allows users to input data, upload data files or click a button to start training.

The training process can be described as Fig. 1: when clients upload all the input files and send a request to the server, the request will be processed and executed by PHP scripts on the server. After training, the server will send the results back to the local machine that made the request.

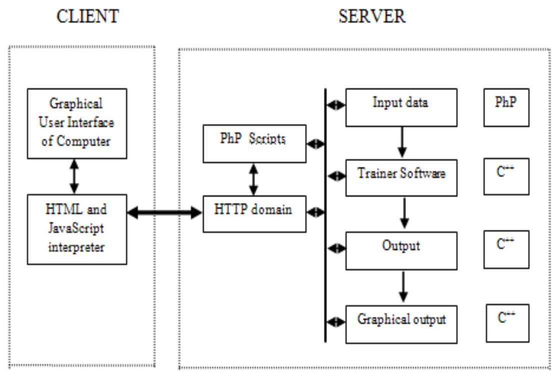


Fig.1. Data Flow in Neural Network Trainer

II. OVERVIEW OF NEURAL NETWORK

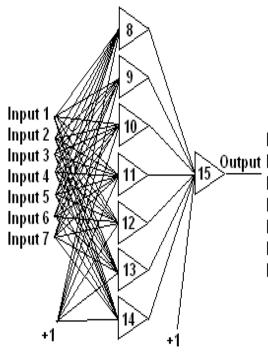
Artificial neural networks (ANN) are used in many industrial applications, such as nonlinear control [10][11], system analysis and diagnosis [12][13], VLSI design [14][15] and data classification [16][17]. It is easy to demonstrate that they can surplus human capabilities to recognize the complex patterns [18][19].

Error back propagation (EBP) algorithm [20][21] is the most popular training method, however, it is not an efficient one because of its slow convergence and its low training stability. Based on EBP algorithm, lots of improvements are developed for better training [22][23], and some of them, such momentum [24], Quick-prop and Resilient EBP [25], work well.

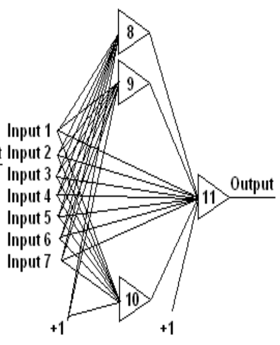
Although complex computation for Jacobian/Hessian matrix is necessary during training process, Levenberg Marquardt (LM) algorithm [26] is considered as one of the most efficient algorithms for small and median sized training patterns. Also there are many good improved algorithms [27] based on LM algorithm 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].

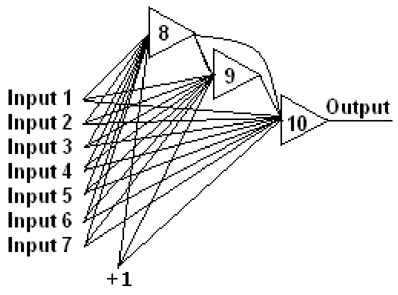
Like EBP algorithm, multilayer perceptron (MLP) networks are broadly accepted in practical applications because they could be realized easily by programming. However, MLP networks perform much less efficient training than other structures, such as MLP with full connections among layers (Bridged MLP) and fully connected cascade (FCC) networks, to deal with the same problems. For example, for parity-7 problem, it needs at least 8 neurons for the standard MLP network to get a solution (Fig. 2(a)), while, for BMLP and FCC networks, only 4 and 3 neurons are required respectively (Fig. 2(b) and (c)). For efficient training, BMLP and FCC networks are better choices, but they also require more challenging computation [8].



(a) 7=7=1 MLP network



(b) 7=3=1 BMLP network



(c) 3 neurons in FCC network

Fig.2. The least neurons required in different neural network structures for parity-7

In this paper, the neural network trainer 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 improved NBN algorithm. Besides the conventional forward-backward computation, a newly developed forward-only (without error back propagation process) computation is also implemented in the software. Based on the neuron by neuron computing scheme, this tool can handle arbitrarily connected cascade (FCC) networks.

In the section IV of this paper, the detailed information of NBN is described, and the section V presents a practical example how to use the NBN 2.0.

III. OVERVIEW OF NETWORK PROGRAMMING

Several network programming tools are available such as Java, CGI, ActiveX, Java-script, VBScript, HTML, and PERL. During software development, it is important to justify which part of the software should run on the client machine and which part should run on the server. CGI is quite different from writing Java applets. Applets are transferred though a network when requested and execution is performed entirely on the client machine that made request. In CGI much less information has to be passed to the server and the server executes instructions based on the given information and sends the results back to the local machine that make the request. In case of neural network trainer it only makes sense to use CGI for training process because it is impossible to send the trainer through the computer network every time it was requested, and this would be extremely slow. Therefore, it is important to develop methods which take advantage for networks. This would require solving several issues, such as:

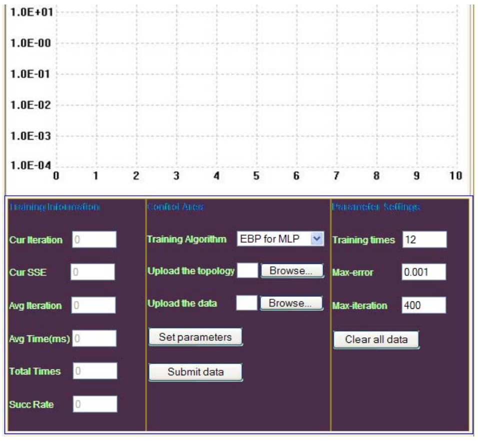
- Minimization of the amount of data which must be sent by network
- Selection of programming tools used for various tasks
- Development of user-friendly interface
- Security, privacy
- Portability of software used on servers and clients
- Task partitioning between the server and client

This training tool currently incorporates CGI, PHP, HTML and Java-Script. CGI programming allows dynamic web-page generation in a web browser. Communication between the CGI program and the web browser is accomplished through a network connection between the web browser and a computer running an HTTP server. A CGI program is executed on a server when it receives a request to process information from a web browser. The server then decides if the request should be granted. If the authorization is secured, the server executes the CGI program and returns the results to the web browser that requested.

PHP is a scripting language like PERL. In fact, its syntax resembles PERL. The main difference lays in the set of standard built-in libraries that support generation of HTML code, processing data from and to the web server, and handling cookies. The same functionality can be accessed in PERL by inclusion of one or more libraries. PHP can be used either as a classical CGI scripting language or as an implementation of ASP technology [28]. Since certain frequently used functionality is built in directly into the language, it is more efficient to use. In general, any specialized tool will be somewhat more efficient for one particular task it was designed for, instead of other powerful but general purpose tools.

IV. NEURAL NETWORK TRAINER

The NBN 2.0 [29][30] is developed based on Visual Studio 6.0 using C++ language hosting on server and communicating with clients through PHP scripts. Its main interface is shown in Fig. 3. In the following part of this section, a detailed instruction of the training tool is presented.



Cur Iteration: 0
Cur SSE: 0
Avg Iteration: 0
Avg Time(ms): 0
Total Times: 0
Succ Rate: 0
Training Algorithm: EBP for MLP
Upload the topology
Upload the data
Training times: 12
Max-error: 0.001
Max-iteration: 400

Fig.3. The user interface of NBN 2.0

A. *Training Algorithms*

Up to now, the training tool is developed with six different types of training algorithms.

- Error Back Propagation (EBP)

This is an improved EBP algorithm [31], with momentum and multiple learning constants. The momentum is introduced to adjust weight vector using the information from the previous iteration. Therefore, the update rule of the improved EBP algorithm can be described as

$$\Delta \boldsymbol{w}_k = -\{\alpha_1, \alpha_2 ... \alpha_3\}(1-\eta)\boldsymbol{g}_k + \eta \Delta \boldsymbol{w}_{k\text{-}1} \qquad (1)$$

where: k is the index of iterations; $\boldsymbol{g}$ is the gradient vector; $\Delta\boldsymbol{w}$ is the weight change; η is the momentum; $\{\alpha 1, \alpha 2 ... \alpha 3\}$ are multiple learning constants. For every iteration, the most optimal learning constant will be picked out by comparison of training errors.

- Levenberg Marquardt (LM)

Original LM algorithm with update rule [32]:

$$\Delta \boldsymbol{w}_k = -(\boldsymbol{H}_k + \mu \boldsymbol{I})^{-1} \boldsymbol{g}_k \qquad (2)$$

where: $\boldsymbol{H}$ is the Hessian matrix; $\boldsymbol{I}$ is the identity matrix and μ is the combination coefficient. The original LM algorithm is only used for MLP networks.

- Neuron By Neuron (NBN)

This is the improved LM algorithm, with advantages: (1) ability of training arbitrarily connected neural networks [1][2]; (2) without Jacobian matrix computation and storage [33].

- Error Back Propagation, forward-only (EBP, forward only)

This is the EBP algorithm with different computation routings from traditional backpropagation computation.

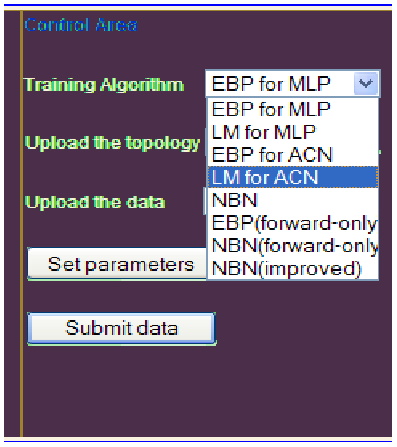
- Neuron By Neuron, forward-only (NBN, forward only)

This is the NBN algorithm with forward-only computation process.

- Neuron By Neuron, improved (NBN-improved)

This is a newly developing algorithm derived from EBP algorithm and Newton algorithm. It's designed to do only one matrix inversion for each iteration during training. With this property, it's supposed to compute faster than LM algorithm. At the same time, by inheriting gradient searching ability from EBP algorithm, the proposed algorithm also can perform stable training.

The drop-down box is used to select training algorithms as Fig.4.



Training Algorithm: EBP for MLP

Fig.4. Training Algorithms

B. *File Instruction*

The neural network interface will receive the request from users with uploading files and input parameters to generate the data files and then send the command to the training software locating on server. There are two files required for training process: topology file and training data file and the users have to use "Browse" buttons as in Fig.5 to upload two these files. These two files have to follow certain syntax so that the training tool can make sense in the correct way.

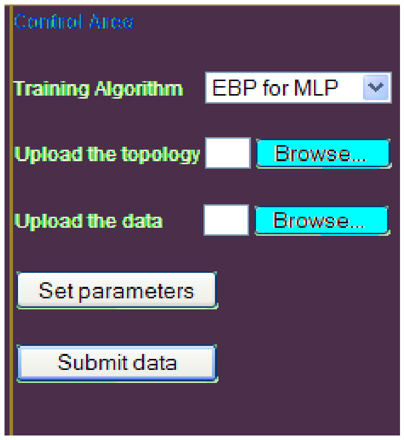
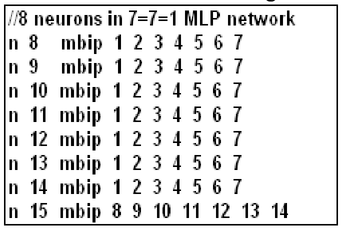


Fig.5. Uploading Files

- Topology file

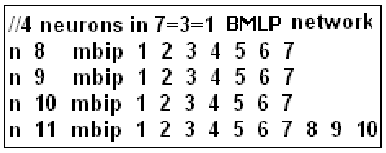
The topology files are named "*.in", and they are mainly used to construct neural network topologies for training. The topology files consist of four parts: topology design, weight initialization (optional), neuron type instruction and training data specification.

The topology design is aimed to create the 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. 6 presents the topology file for the neural networks shown in Fig. 2.



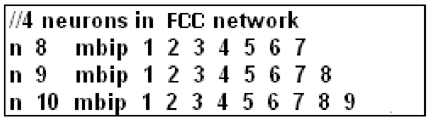
```
//8 neurons in 7=7=1 MLP network
n 8  mbip 1 2 3 4 5 6 7
n 9  mbip 1 2 3 4 5 6 7
n 10 mbip 1 2 3 4 5 6 7
n 11 mbip 1 2 3 4 5 6 7
n 12 mbip 1 2 3 4 5 6 7
n 13 mbip 1 2 3 4 5 6 7
n 14 mbip 1 2 3 4 5 6 7
n 15 mbip 8 9 10 11 12 13 14
```

(a) 7=7=1 MLP network



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

(b) 7=3=1 MLP-FCL network

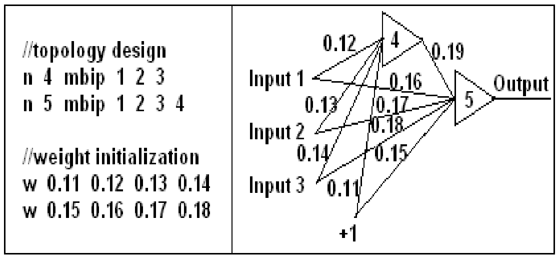


```
//4 neurons in FCC network
n 8  mbip 1 2 3 4 5 6 7
n 9  mbip 1 2 3 4 5 6 7 8
n 10 mbip 1 2 3 4 5 6 7 8 9
```

(c) 3 neurons FCC network

Fig.6. Topology design for networks shown in Fig. 2; all neurons are bipolar neurons.

The weight initialization part is used to specify the initial weights for training and this part is optional. If there is no weight initialization in the topology file, the trainer will generate the 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. 7 shows the example of weight initialization for the parity-3 problem with 2 neurons in FCN network. "//" sign in the topology file is understood as the comment by the trainer.



```
//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.7. 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, the linear neurons are considered to be the output neurons. The general command is ".model [mbip/mu/mlin] fun=[bip/uni/lin], gain=[value]". "gain" is the parameter of related activation functions (Fig. 8).

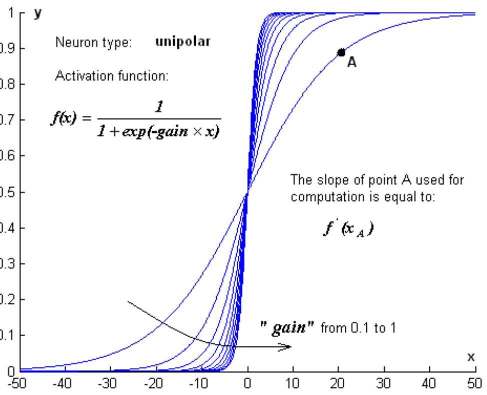
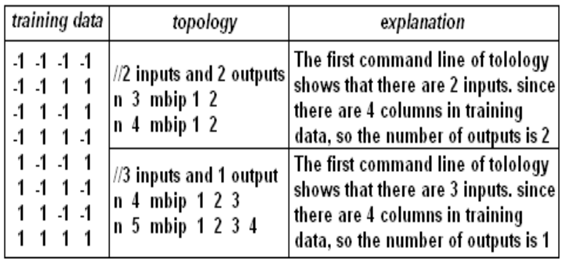


Fig.8. Activation functions of unipolar neuron with different "gain" values; for bipolar and linear neurons, "gain" plays the same roles as in unipolar neurons

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 file

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 the 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. 9). The training pattern files are specified in the topology files as mentioned above, and it should have the same route as the related topology files.



training data	topology	explanation

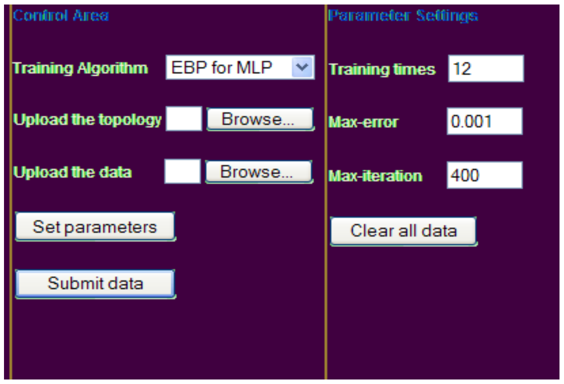
-1 -1 -1 -1
-1 -1 1 1
-1 1 -1 1
-1 1 1 -1
	//2 inputs and 2 outputs
	n 3 mbip 1 2
	n 4 mbip 1 2
		The first command line of tolology shows that there are 2 inputs. since there are 4 columns in training data, so the number of outputs is 2

1 -1 -1 1
1 -1 1 -1
1 1 -1 -1
1 1 1 1
	//3 inputs and 1 output
	n 4 mbip 1 2 3
	n 5 mbip 1 2 3 4
		The first command line of tolology shows that there are 3 inputs. since there are 4 columns in training data, so the number of outputs is 1

Fig.9. Get the number of inputs and the number of outputs from the data file and topology

C. Parameter Settings

All parameters can be edited in the user interface Fig. 10 with the blanks written in HTML form and Java-Script. There are three main parameters: training times, max-error, max-iteration have to be filled in. These values must be the number. If users do not put these parameters in the proper forms, the warning message will pop up. If these parameters are left with the blanks, the trainer will automatically assign their values by zeros. Besides these parameters, there are other parameters with default values which will dynamically pop up for inputting data of "combination coefficient", "scale constant", "momentum constant", "learning constant", "alpha constant", "beta constant", "gamma constant" which depends on the selected algorithm when users click the "Set parameters" button.



Training Algorithm: EBP for MLP
Upload the topology
Upload the data
Training times: 12
Max-error: 0.001
Max-iteration: 400

Fig.10. Input Parameters

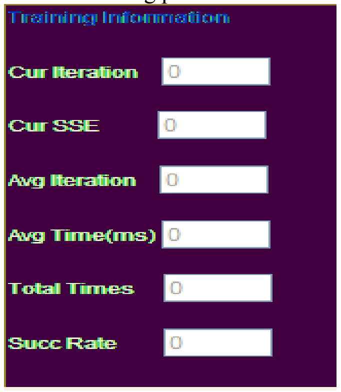
"Clear all data" button is used to clear all old data set and start again with a new set.

TABLE 1: PARAMETERS FOR TRAINING

Parameters	***Descriptions***
alpha	Learning constant for EBP
momentum	Momentum for EBP
scale	Parameter for LM/NBN
mu	Parameter for LM/NBN
po alpha	Parameter for NBN improved
po beta	Parameter for NBN improved
po gama	Parameter for NBN improved
max error	Maximum error
max iteration	Maximum iteration for training
Training times	The number of training trials

D. Training Information

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. These parameters have default values "0s", are disabled and only updated after each training process.



Cur Iteration: 0
Cur SSE: 0
Avg Iteration: 0
Avg Time(ms): 0
Total Times: 0
Succ Rate: 0

Fig.11. Training Information

E. Plot 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. 3), shows the number of iterations cost for training. The plot is provided in bmp format, widely recognized by most of popular web browsers. All plots are dynamically generated.

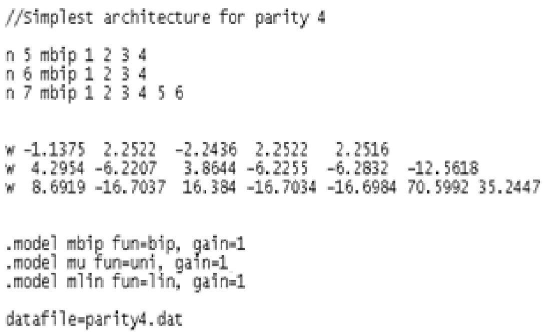
The "Submit data" button allows the training tool to check all inputs. This is very necessary because the network trainer can't handle all types of formats of files and input parameters which can affect the training success. All submitted data is saved into the database and the trainer will automatically retrieve these data from the database and initialize them as the next data set.

When all the requirements are fulfilled and set up properly the button "Start training" will appear. Otherwise, it will not and the training tool will send the error warnings back to the clients. The training process can be long or short depending on the convergence of the selected algorithm, the complication of network structure, the input parameters and the speed of network. If the training process is successful the training result will be generated. The training result file is used to store training information and the results such as training algorithm, training pattern file, topology, parameters, initial weights, result weights and training results will be saved after the training process finishes. The name of the training result file is generated automatically according to the name of the training file and hyperlinked right above the Graphical Users Interface. If the training process gets errors the "Error" hyperlink will appear instead of the "Training Result" hyperlink.

V. EXAMPLE

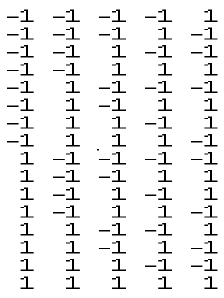
This example will present all steps of training process with parity-4 problem. Two input files include parity4.in, parity4.dat, selected algorithm is NBN with parameter training times= 500, max-error= 0.001, max-iteration= 500, combination coefficient= 0.01, scale constant=10.

- Input files



```
//Simplest architecture for parity 4

n 5 mbip 1 2 3 4
n 6 mbip 1 2 3 4
n 7 mbip 1 2 3 4 5 6


w -1.1375  2.2522  -2.2436  2.2522   2.2516
w  4.2954 -6.2207   3.8644 -6.2255  -6.2832  -12.5618
w  8.6919 -16.7037  16.384 -16.7034 -16.6984 70.5992 35.2447


.model mbip fun=bip, gain=1
.model mu fun=uni, gain=1
.model mlin fun=lin, gain=1

datafile=parity4.dat
```

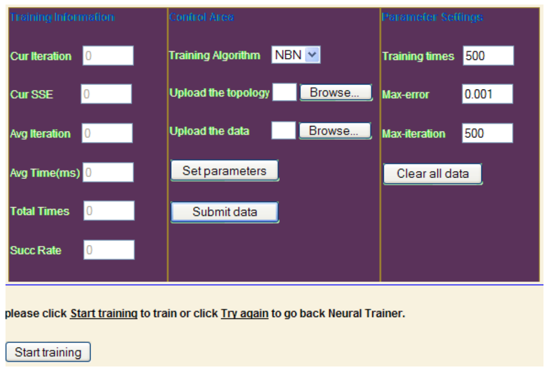
Fig.12. Parity4.in (topology)



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

Fig.13. Parity4.dat (training data)

- Input parameters



Training Information

Cur Iteration: 0
Cur SSE: 0
Avg Iteration: 0
Avg Time(ms): 0
Total Times: 0
Succ Rate: 0

Control Area

Training Algorithm: NBN
Upload the topology
Upload the data

Parameter Settings

Training times: 500
Max-error: 0.001
Max-iteration: 500

please click Start training to train or click Try again to go back Neural Trainer.

Fig.14. Input Parameters

- Output plot

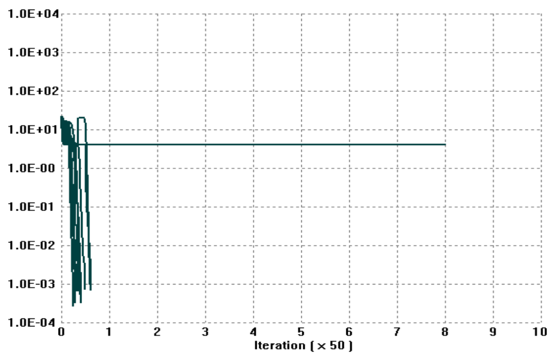
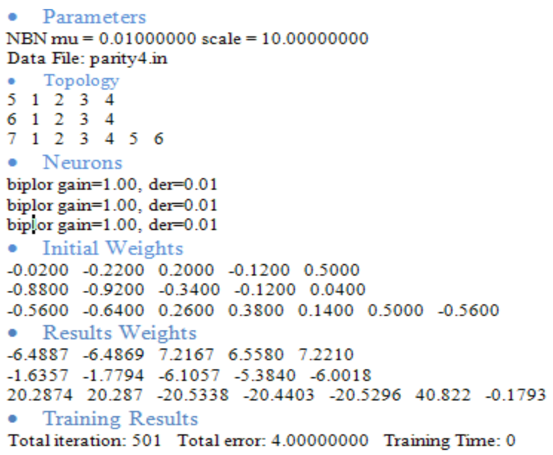


Fig.15. Output Plot

- Training result



- Parameters

NBN mu = 0.01000000 scale = 10.00000000
Data File: parity4.in

- Topology

```
5  1  2  3  4
6  1  2  3  4
7  1  2  3  4  5  6
```

- Neurons

biplor gain=1.00, der=0.01
biplor gain=1.00, der=0.01
biplor gain=1.00, der=0.01

- Initial Weights

```
-0.0200  -0.2200  0.2000  -0.1200  0.5000
-0.8800  -0.9200  -0.3400  -0.1200  0.0400
-0.5600  -0.6400  0.2600  0.3800  0.1400  0.5000  -0.5600
```

- Results Weights

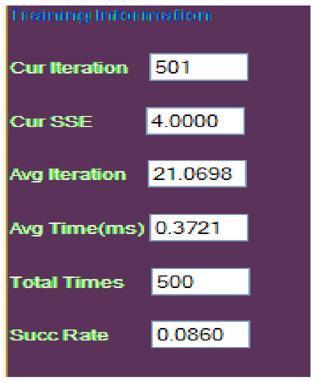
```
-6.4887  -6.4869  7.2167  6.5580  7.2210
-1.6357  -1.7794  -6.1057  -5.3840  -6.0018
20.2874  20.287  -20.5338  -20.4403  -20.5296  40.822  -0.1793
```

- Training Results

Total iteration: 501   Total error: 4.00000000   Training Time: 0

Fig.16. Training Results

- Training Information



Training Information

Cur Iteration: 501
Cur SSE: 4.0000
Avg Iteration: 21.0698
Avg Time(ms): 0.3721
Total Times: 500
Succ Rate: 0.0860

Fig.17. Training Information

VI. Conclusion

In this paper, the training tool NBN 2.0 through computer networks is introduced for neural network training. This trainer 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 example presented in the paper, one can get familiar with this useful tool for neural network training. The NBN 2.0 is available at http://131.204.128.91/NNTrainer/index.php, and it will be updated from time to time.

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] B. M. Wilamowski, John Regnier, Aleksander Malimowski, "SIP-Spice Intranet Package", *IEEE International Conference on Industrial Electronics,* pp.192-195, June 7-10 1998.

[4] B. M. Wilamowski, A. Malinowski, and J. Regnier, "SPICE based Circuit Analysis using Web Pages", *ASEE 2000 Annual Conference, St. Louis, MO*, CD-ROM session 2520, June 18 to Aug 2, 2000.

[5] Stuttgart Neural Network Simulator, URL from May 2002: http://www-ra.informatik.uni-tuebingen.de/SNNS/.

[6] M. Manic, B. M. Wilamowski, and A. Malinowski, "Internet based Neural Network Online Simulation Tool", *Proc. of the 28th Annual Conference of the IEEE Industrial Electronics Society*, pp. 2870-2874*,* Sevilla, Spain, Nov 5-8, 2002.

[7] Neurodesigner- comprehensive neural network software, URL from May 2002: http://www.neurodesigner.com/

[8] B. M. Wilamowski, Neural Network Architectures and Learning Algorithms, IEEE Industrial Electronics Magazine, vol. 3, no 4, pp. 56-63, 2009.

[9] A. Malinowski, B. M. Wilamowski, "Internet Technology as a Tool for Solving Engineering Problems ", *The 27th Annual Conference of the IEEE Industrial Electronics Society* (tutorial) pp. 1622-1630, Denver CO, Nov 29-Dec 2, 2001.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] B. Vigdor, B. Lerner, "Accurate and Fast Off and Online Fuzzy ARTMAP-Based Image Classification With Application to Genetic Abnormality Diagnosis," *IEEE Trans. on Neural Networks,* vol. 17, no. 5, pp. 1288-1300, May 2006.

[17] 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.

[18] 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.

[19] 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.

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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

[28] Aleksander Malinowski, Bogdan Wilamowski, "Internet Technology as a Tool for Solving Engineering Problems", *The 27th Annual Conference of the IEEE Industrial Electronics Society*, pp 1622-1630, November 29- December 2, 2001.

[29] Hao Yu and B. M. Wilamowski, "C++ Implementation of Neural Networks Trainer", *13-th International Conference on Intelligent Engineering Systems*, INES-09, Barbados, April 16-18, 2009.

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

[31] Robert Hecht-Nielsen, "Theory of the Back Propagation Neural Network". *Proc. 1989 IEEE IJCNN*, 1593-1605, IEEE Press, New York, 1989.

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

[33] Wilamowski M. Bogdan, Hao Yu, "Improved computation for Levenberg Marquardt Training", *IEEE Trans. on Neural Networks* (accepted).