A new C++ implemented feed forward neural network simulator

József Sütő
Faculty of Informatics
University of Debrecen
Debrecen, Hungary
sutojozsef90@gmail.com

Stefan Oniga
Faculty of Informatics
University of Debrecen
Debrecen, Hungary
oniga.istvan@inf.unideb.hu

***Abstract*—This paper presents the implementation of a simulator application for feed forward neural networks which was made in Qt application framework. The paper demonstrates the object oriented design and the performance of the software. The main topics cover the class organization and some test results where the Matlab neural network toolbox was used as reference.**

Keywords— Neural network; Function approximation, C++; Qt; Object oriented design

I. INTRODUCTION

Nowadays, artificial neural network (ANN) is an outstanding research area. Many researchers work on ANN algorithms and architecture development which require lots of tests. Unfortunately, the number of available free of charge simulator or tester software is rather limited. The best known application is the Matlab neural network toolbox, but it is rather costly. On the other hand, the toolbox can be a very good reference to test similar applications.

Our simulator application provides a good possibility to test feed forward ANN in education and research. The advantages of the application is that it is easy to use and every useful parameters of training algorithms are adjustable. Moreover, the program supports some well applicable error analysis options. In many cases, the parameters play an important role in training process. For example, the learning rate influences the oscillation of performance or stretching log-sigmoid transfer function influences the steepness of a neuron output.

The application supports training algorithms for single and multi-layer ANN. For single layer network the Hebbian and the Widrow-Hoff (LMS), while for multi-layer networks some modified backpropagation algorithms were implemented.

II. USAGE OF THE APPLICATION

The software was made in Qt. Qt is a cross-platform application framework that based on C++. It is well applicable for developing software with graphical user interface (GUI). In the programming environment the basic GUI elements are given, thus the developer can form the GUI easily. The simulator application provides a simple GUI where the users can set the ANN properties. Fig. 1 shows the "main" GUI of the application.

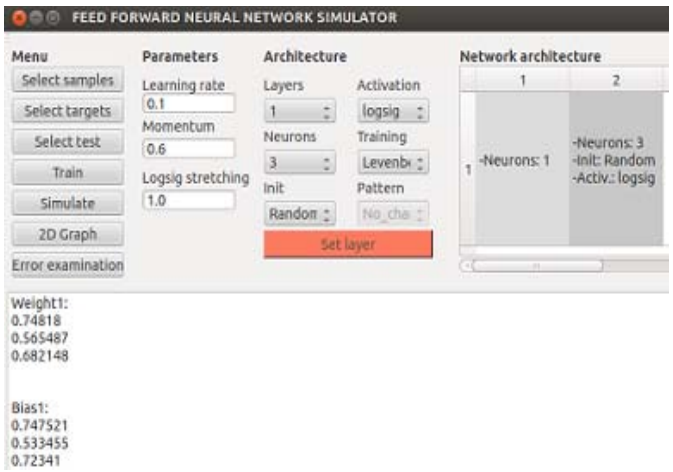

Fig. 1. The main GUI of the application

On the first figure in the architecture area, combo boxes contain properties of layers while the network architecture table shows the adjusted properties. In the bottom text browser the user will see initial values and information about the training and the simulation algorithms operation. The software reads sample patterns, test patterns and target values from file and similarly writes the output (error and result) into file.

The training process contains an iteration limit (1000). Training will stop if the iteration counter reaches the limit or the iteration error does not decreases significantly.

The parameters area on fig. 1 refers to the constant multipliers of the implemented training algorithms. For example, α denotes the *learning rate* while γ denotes the *momentum* in weight calculation of LMS and momentum based backpropagation algorithm.

$$\boldsymbol{W}(t+1) = \boldsymbol{W}(t) + 2\alpha \boldsymbol{e}(t)\boldsymbol{p}^{\boldsymbol{T}}(t) \quad (1)$$

$$\Delta \boldsymbol{W}^{\boldsymbol{k}}(t+1) = \gamma \boldsymbol{W}^{\boldsymbol{k}}(t) - (1-\gamma)\alpha \boldsymbol{s}^{\boldsymbol{k}}(t)\left(\boldsymbol{o}^{\boldsymbol{k-1}}\right)^{T} \quad (2)$$

In the above equations the bold notations are matrices and vectors. The **W** denotes the weight matrix, ***e*** is the error vector and **p** is the input pattern in the *t.* iteration. In multi-layer networks *k* indicates the layer, **o** is the output vector of a given layer and ***s*** is the sensitivity of the error [1]. Obviously, the γ and α were used similarly in bias calculation. In addition, the *sigmoid stretching* controls the steepness of the sigmoid transfer function.

$$F(i^k) = \frac{1}{1 + e^{-\lambda i^k}} \quad (3)$$

Where i^k is the summarized input of a neuron and λ is equal with the *sigmoid stretching* parameter. Therefore the derivative can be written as:

$$\lambda\left(1 - \frac{1}{1 + e^{-\lambda i^k}}\right)\left(\frac{1}{1 + e^{-\lambda i^k}}\right) = \lambda(1 - o^k)o^k \quad (4)$$

In the equation above (4) o^k denotes the neuron output. Moreover, the application includes another two GUIs where the user can visualize different types of 2D and 3D graphs. One of them is responsible for 2D graphs while the other is responsible for 3D graphs. The 2D plots are made with QCustomPlot. QCustomPlot is a well applicable C++ widget for plotting. It supports some useful possibilities, for instance, axis scaling due to mouse scroll. Currently four plotting mode are supported (line plot (x, y), line plot (x), impulse and bar plot). Plotting algorithms require data file(s) which contain the values of the graph. The name of the graph or the function depends on the selected file name. Fig. 2 illustrates an example how QCustomPlot draws the error function.

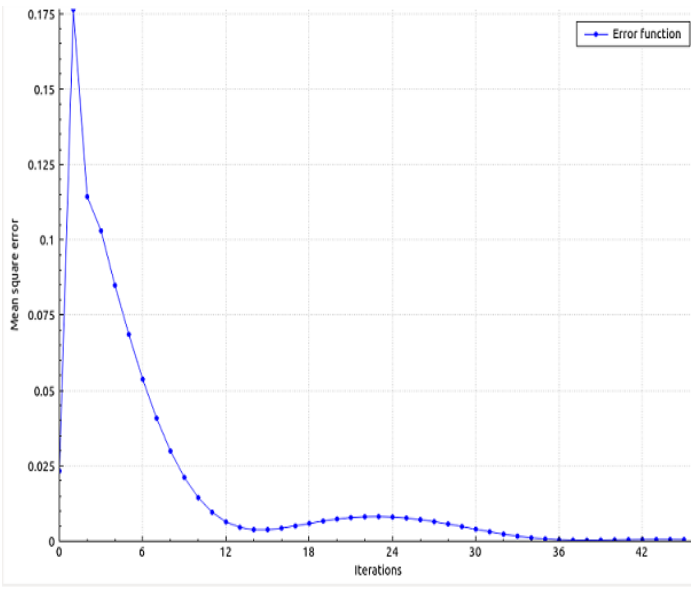

Fig. 2. Error function after training

Under Linux, the programmer can call shell script from Qt which contains one or more embedded GLE (Graphics Layout Engine) scripts. GLE is a graphics scripting language that is available for free of charge. The program uses the GLE to display error-tester graphs. The user can examine the change of the output error according to the alteration of one or two parameters. The error-tester method modifies the selected parameters in little steps over the entire adjusted range and stores the measured error into matrix or vector (depends on how many parameters change). For instance, fig. 3 shows an error surface which derives from two weight modification. On fig. 3 there is an arrow which indicates the minimum point of the surface.

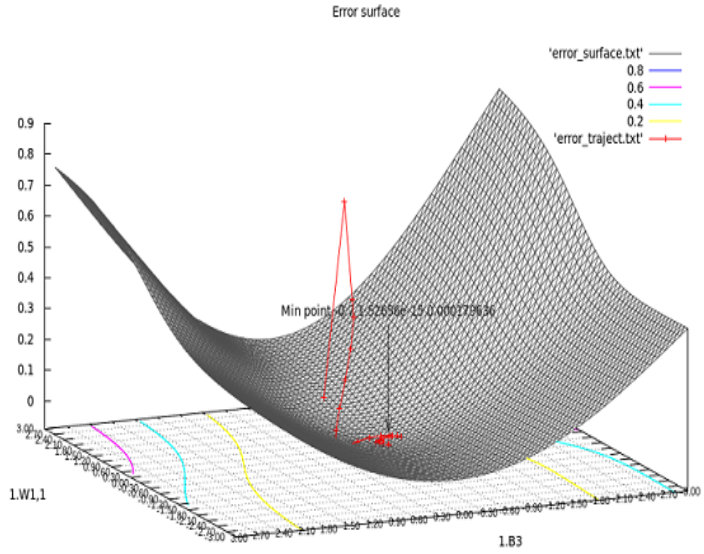

Fig. 3. Error surface

In addition, the red trajectory refers to the error function. Thanks to the error-tester option the user can check the convergence of the error trajectory. Obviously, the training algorithm works properly if the error trajectory converges to the minimum point. Another useful graph is the contour surface. The contour surface shows the convergence of the error trajectory in another approach. The contour surface can be seen on fig. 4.

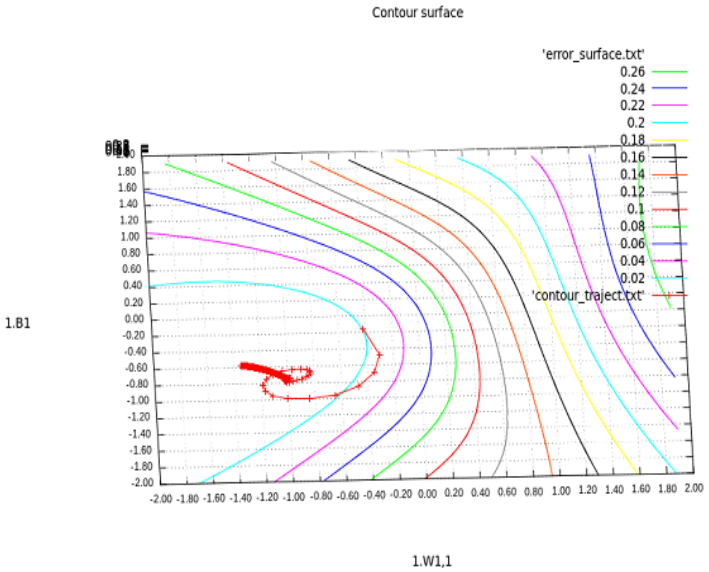

Fig. 4. Contour surface

III. CLASS ORGANIZATION

The application builds up from classes. While c++ is an object oriented programming language, the developer can represent complex structure as an object. Since the training algorithms are based on matrix operations, thus a matrix class that includes every necessary methods is indispensable. In c/c++ the developer should pay attention to memory management when the program calculates lots of matrix operations. If the developer would like to represent matrix dynamically then a pointer array is necessary. Every instance of the matrix class allocate new memory fields. Usually the function which performs matrix operation returns with a new matrix that contains the result. In that case, if the program will not free temporary matrices (unused memory) the program will run out of operative memory and the program crashes (or the whole system). Another undesired situation when the CPU is unexploited because the memory is loaded and the data exchange between operative memory and caches is time-consuming. It is recommended to create matrix pointers which will point to the newly created matrix and free those after usage.

In the application every layer is an object, therefore, they independently manageable. The network input was considered as a separate layer ("inputlayer" object) with own data types and methods. Hidden layers are the same objects where the last adjusted will be the ANN output. In this case, the most data types of the class are matrices because the algorithms are based on matrix operations.

Simpler algorithms such as Hebbian or LMS are located in a header file. In this case, new class creation is not necessary. Actually, the simulator program supports the following training algorithms:

- Hebbian
- Widrow-Hoff (LMS)
- Standard backpropagation
- Momentum based backpropagation
- Variable learning rate backpropagation
- Levenberg-Marquardt

IV. TEST RESULTS

The performance of the program was tested by three function approximation tests. Function approximation is one of the main application area of neural networks. In the first test the program approximated a simple exponential function. The exponential function is the following:

$$F_1(x) = 0.8 \exp(x) \quad (5)$$

Where *x* denotes the interval between -2 and 3. The program output was compared with the Matlab neural network fitting tool output. In both cases, 1-2-1 (1 input neuron, 2 hidden neurons and 1 output neuron) architecture was used. The two outputs are nearly equal to the original. Fig. 5 illustrates the test result of the first function approximation.

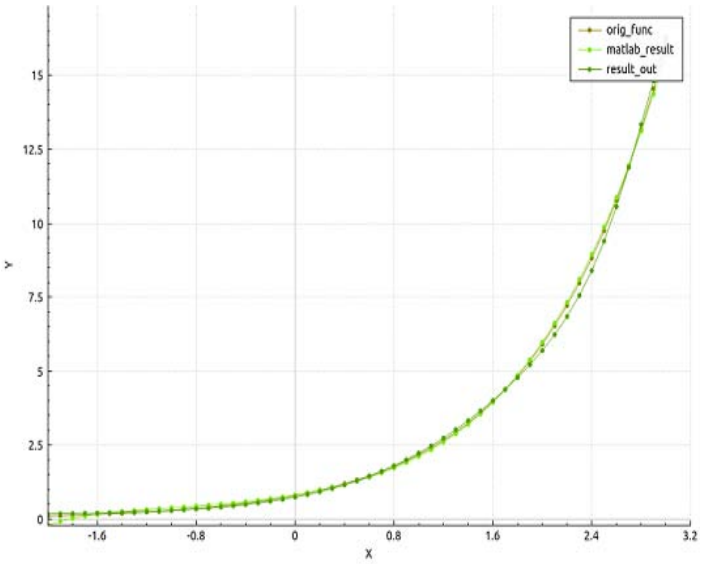

Fig. 5. The result of the first test

On the figure above, the three curves are the original function, the Matlab and the program result.

In the second test, the program approximated a slightly more complex sinusoidal function. The following equation describes the function:

$$F_2(x) = 1 + \sin(pi * x) \quad (6)$$

The program output was again compared with Matlab. Now, the network architecture is 1-4-1. The two results are almost identical. Fig. 6 shows the second function approximation test.

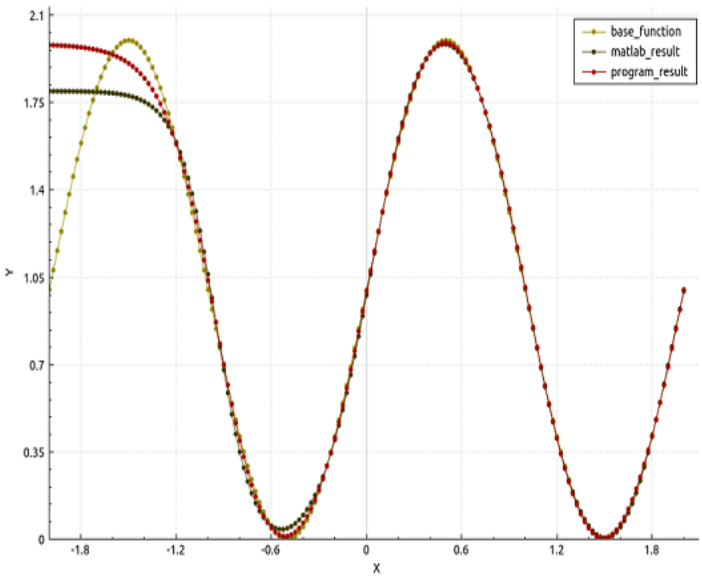

Fig. 6. The result of the second test

Another indicator is the adjusted parameters of the network. If the parameters after the training are equal, the two training algorithms work similarly. For example, in the

previous test the weight matrices of the second layer in Matlab and in the simulator are the following:

$$W_2^M = \begin{bmatrix} -1.5242 \\ 1.7996 \\ -1.5715 \\ -1.1648 \end{bmatrix} \quad (7)$$

$$W_2^S = \begin{bmatrix} -1.7401 \\ 1.5232 \\ -1.3538 \\ -1.1020 \end{bmatrix} \quad (8)$$

W_2^M refers to the Matlab weight while W_2^S is the weight of the simulator. The difference between the two matrices is negligible. It is another evidence for the correct operation of the simulator.

In the last test a sawtooth wave was approximated with neural network. In order to the network can approximate the sawtooth wave it should contain more neurons in the hidden layer. The domain of the sawtooth is between -2 and 2. The following equation (9) describes the sawtooth:

$$F_3(x) = \begin{cases} 2 - |x|, & x < 0 \\ 0, & x = 0 \\ x, & x > 0 \end{cases} \quad (9)$$

Now, the applied architecture of the network is 1-20-1. The result of the last test can be seen on fig. 7.

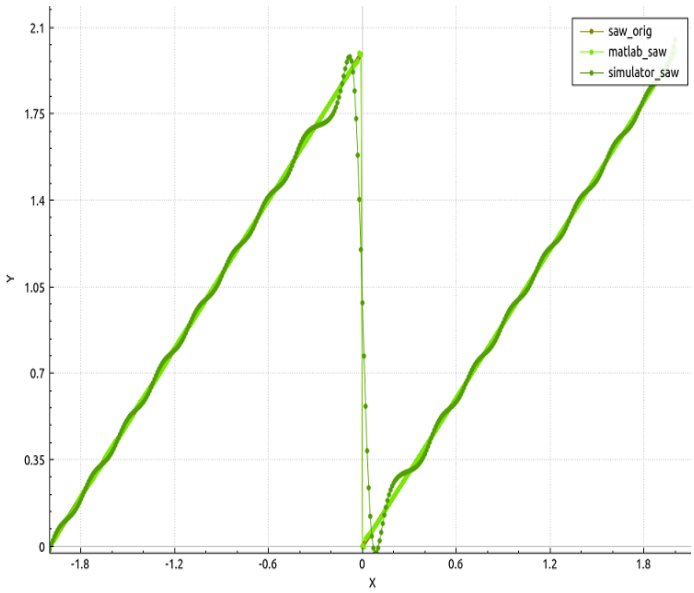

Fig. 7. The result of the last test

V. CONCLUSION

The test results show that the simulator works properly. Therefore, it well applicable in the education and in research works. Some functions and algorithms are specialized for educational purposes only. One of them is the standard backpropagation algorithm because the modified backpropagation algorithms give better solution. On the other hand, the Levenberg-Marquardt training algorithm can be useful in research.

One of the main property of the application is the flexibility. The source code is easily modifiable and the developer can upgrade the algorithms quickly.

The program was not tested completely. In the future, we want to test some additional indicators of the program such as the running time. In addition, we should improve the implemented algorithms in order to the program provides reliable solution in all cases.

ACKNOWLEDGMENT

This research was supported by the **European Union** and the **State of Hungary, co-financed by the European Social Fund** in the framework of TÁMOP 4.2.4. A/2-11-1-2012-0001 "National Excellence Program".

REFERENCES

[1] M. T. Hagan, H. B. Demuth, M. Beale, Neural network design, PWS Publishing Cmpany, 1996.

[2] M. H. Beale, M. T. Hagan, H. B. Demuth, Neural network toolbox user's guide, The MathWork Inc., 2013.

[3] A. Zell, N. Mache, R. Hübner, G. Mamier, M. Vogt, M. Schmalzl, K. Herrmann, "SNNS (Stuttgart neural network simulator)", Neural network Simulation Environments, vol. 254, pp. 165-186, 1994.

[4] B. Kröse, P. Smagt, An introduction to neural networks, University of Amsterdam, November 1996.

[5] A. Zell, N. Mache, T. Sommer and T. Korb, "Recent developments of the SNNS neural network simulator", SPIE Proceedings, Applications of Artificial Neural Networks II, vol. 1469, 1991.

[6] T. Makino, "A discrete – event neural network simulator for general neuron models", Neural Computing & Applications, vol. 11, pp. 210-223, June 2003.

[7] S. Haykin, Neural networks. A comprehensive foundation. Second edition, McMaster University, Hamilton, Ontario, Canada, 1999.

[8] M. T. Hagan, M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm", Neural Networks, IEEE Transactions, vol. 5, pp. 989-993, August 2002.

[9] P. Peretto, An introduction to the modeling of neural networks, Cambridge University Press, 1992.

[10] L. M. Hines, N. T. Carnevale, "The NEURON simulation environment", Neural Computation, vol. 9, no. 6, pp. 1179-1209, August 1997.

[11] D. Nguyen, B. Widrow, "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights", Neural Networks, 1990 IJCNN International Joint Conference, vol. 3, pp. 21-26, June 1990.

[12] J. Sütő, S. Oniga, "Testing artificial neural network for gesture recognition", Abstracts and Pre-Proceedings, 9 th international conference on applied mathematics, pp. 21, 2013.

[13] T. Masters, Practical Neural Network Recipes in C++, Academic Press, 1993.

[14] K. Hornik, "Approximation capabilities of multilayer feedforward networks", Neural Networks, vol. 4, pp. 251-257.

[15] S. Oniga, I. Orha, "Hardware implemented neural networks used for hand gestures recognition", Carpathian Journal of Electronic and Computer Engineering, vol. 4, no. 1, pp. 93-96, 2011.

[16] A. Tisan, A. Buchman, S. Oniga, C. Gavrincea, "A generic control block for feedforward neural network with on-chip delta rule learning algorithm", Proceedings of the 5th WSEAS Int. Conf. on DATA NETWORKS, COMMUNICATIONS & COMPUTERS, pp. 162-167.