MODELING AND VERIFICATION OF DIGITAL LOGIC CIRCUIT USING NEURAL NETWORKS

Dr. Reza Raeisi

Indiana State University, Terre Haute, Indiana; Email: reza@indstate.edu

1. INTRODUCTION

Artificial Neural Networks are electronic models based on the neural structure of a human brain (DACS Store, Artificial Neural Networks Technology). Neural Networks have plethora of applications such as in process control, financial forecasting, voice recognition, pattern recognition, medical diagnosis and many other areas (Principe, et al; 2000)

The problem of interest to us consists of first, using artificial neural networks for modeling digital logic circuits based on truth table, logic diagram or Boolean logic expression and second to contribute specific method and understanding of hardware aspects of neural networks for modeling and verification of digital logic circuits.

Neural networks are analogous to digital logic circuits because they produce outputs to specific inputs. But unlike digital logic, a neural network can be trained and is dynamic. Its output values can change corresponding to its inputs every time it is trained.

In order to test the digital logic, a neural network model is built with specific parameters that could be trained and tested. This neural network model has the ability to change dynamically its logic behavior through user inputs.

2. METHODOLOGY

For the purpose of testing digital logic circuits, Java Object Oriented Neural Engine (JOONE) with back-propagation algorithm is used. In neural network software packages, a neuron is called a processing element. There are three layers in a neural network, namely, input, hidden and output layer. The back-propagation algorithm is the most widely used algorithm. It begins by feeding input values forward through the network. It calculates the output error and propagates it backward through the network in order to find errors for neurons contained in hidden layers. This process is repeated until the neural network output falls within the minimum tolerance value.

An example of a back propagation to test full adder data is as shown in Figure 1. The required data Input A, Input B, Input C is fed to the neural network. The neural network needs to learn this data by experience. It gives the Sum and the Carry error which is then compared with the desired values. It adjusts the weights that connect different nodes in the layers to give the desired output. This process takes place until the error is within the minimum tolerance value. The same concept is used to model wide variety of combinational logic circuits like multipliers, multiplexers and demultiplexers for different input values.

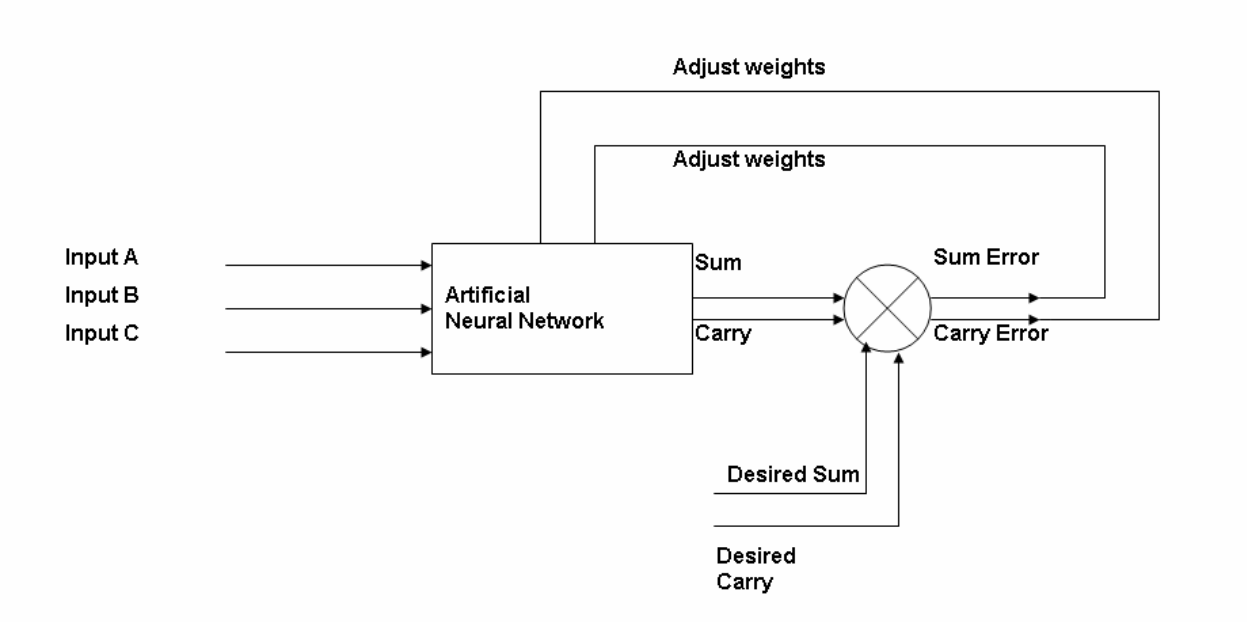

Figure 1: An example of a back-propagation network to model a full adder data.

Joone is a neural net framework written in Java. It's composed by a core engine, a GUI editor and a distributed training environment. Neural network that tests digital logic is built using the GUI editor. The GUI editor is a Graphical User Interface to visually create, modify and test the neural net. The features of the GUI editor are:

- Creation of any architecture, according to the nature of the core engine
- Neural net save and restore from a file system
- Setting of all parameters for each component of the neural network
- Checking for the correctness of the neural network
- Use of pre-processing and control plug-ins
- A complete macro editor to define and control the scripting code
- A control centre to train the neural network
- A charting component to visualize the output values of the neural network
- A flexible environment based on XML files to add new layers, synapses, etc. to the editor's palette
- Exporting of the embedded neural network to distribute and run it on external applications

See (Marrone. P, Java Object Oriented Neural Engine, The GUI editor).

This program generates a truth table for a particular logic circuit in order to train its network. The user can enter the data and try to train and test the network until the optimum values for the network are reached. The flowchart for the training and testing the neural network that uses a back-propagation algorithm is as shown in the Figure 2.

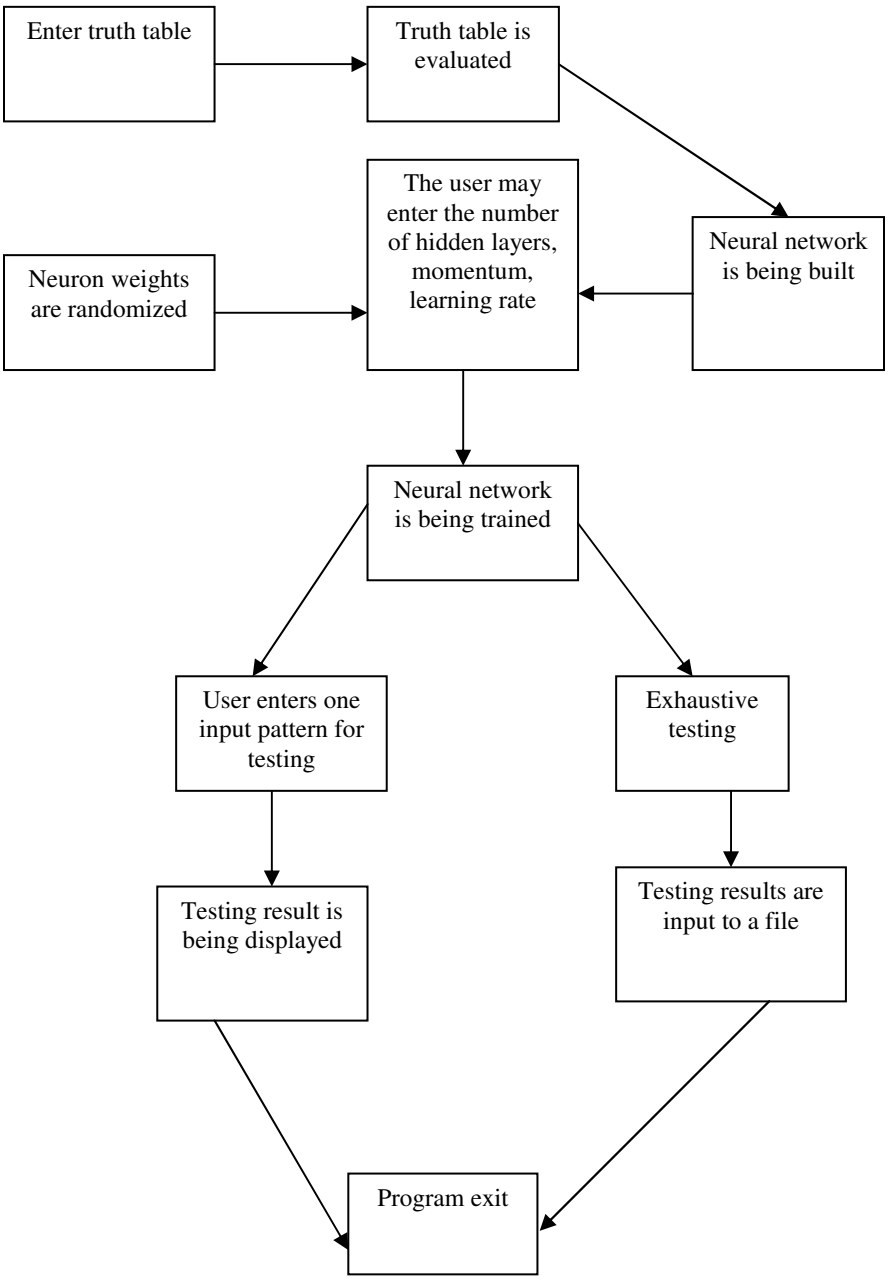

Figure 2: Flowchart for training and testing of neural network using back-propagation algorithm.

3. TRAINING AND TESTING

For training and testing different digital logical functions like AND, XOR, full adder and other combinational logic circuits, a neural network model is built that employs a back-propagation algorithm. The neural network dynamically adjusts itself depending upon the size of the network. The number of input variables determines the number of input layer neurons. The number of hidden layers is predicted based on the performance of the network and there is no particular criterion for choosing the number of hidden layers. Again, the number of output variables determines the number of output layer neurons. The input and the expected output values are entered in a text editor. The other parameters that also affect the performance are the momentum, learning rate, and epoch. Epoch is the number of times the text file is processed. Momentum determines the speed at which the network runs and the learning rate is the rate at which the neural network learns all the input and the expected output values. After entering all the values of the variables, the network is first trained until it reaches its minimum acceptable RMS error (less than 0.01). If the network has an RMS error less than 0.01, it is then tested for an epoch of 1. The results are displayed in the text editor file specified in the JOONE software. This process is repeated with different network parameters. A logic 1 is assigned to any output value between 0.5 and 1. A logic 0 is assigned to any output value between 0 and 0.5.

Figure 3 shows the input and expected outputs entry through a notepad file for a full adder circuit.

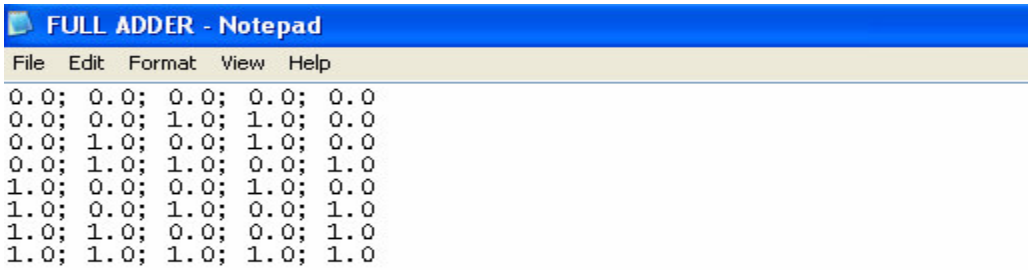

Figure 3: Full Adder input entry in the notepad file.

After the inputs and the expected outputs are entered in the notepad, a full adder circuit is built using the GUI editor that is available in JOONE software. This is as shown in Figure 4. It consists of an input layer with three inputs, an output layer with two outputs for sum and carry and a hidden layer which is chosen through trial and error method. The training set and the desired data are entered through the file input layers. The teacher layer helps the neural network to learn in order to reach the expected value of outputs.

The method of building these blocks are explained in the help menu provided at the top of the neural editor.

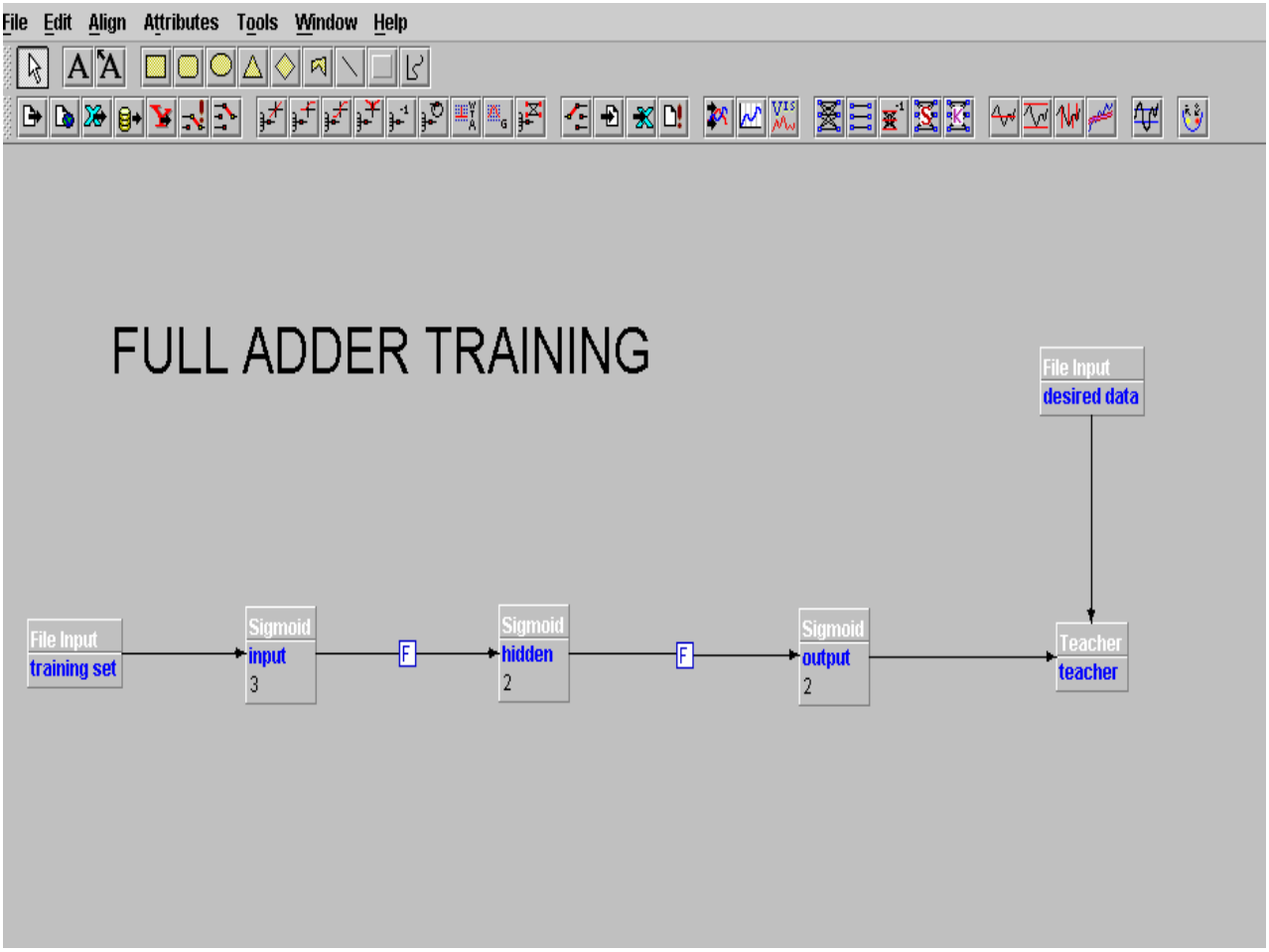

Figure 4: Full adder training using Joone neural net editor.

After the neural network is built using the standard GUI editor, the desired parameters are entered for the training to take place. An example is as shown in the Figure 5.

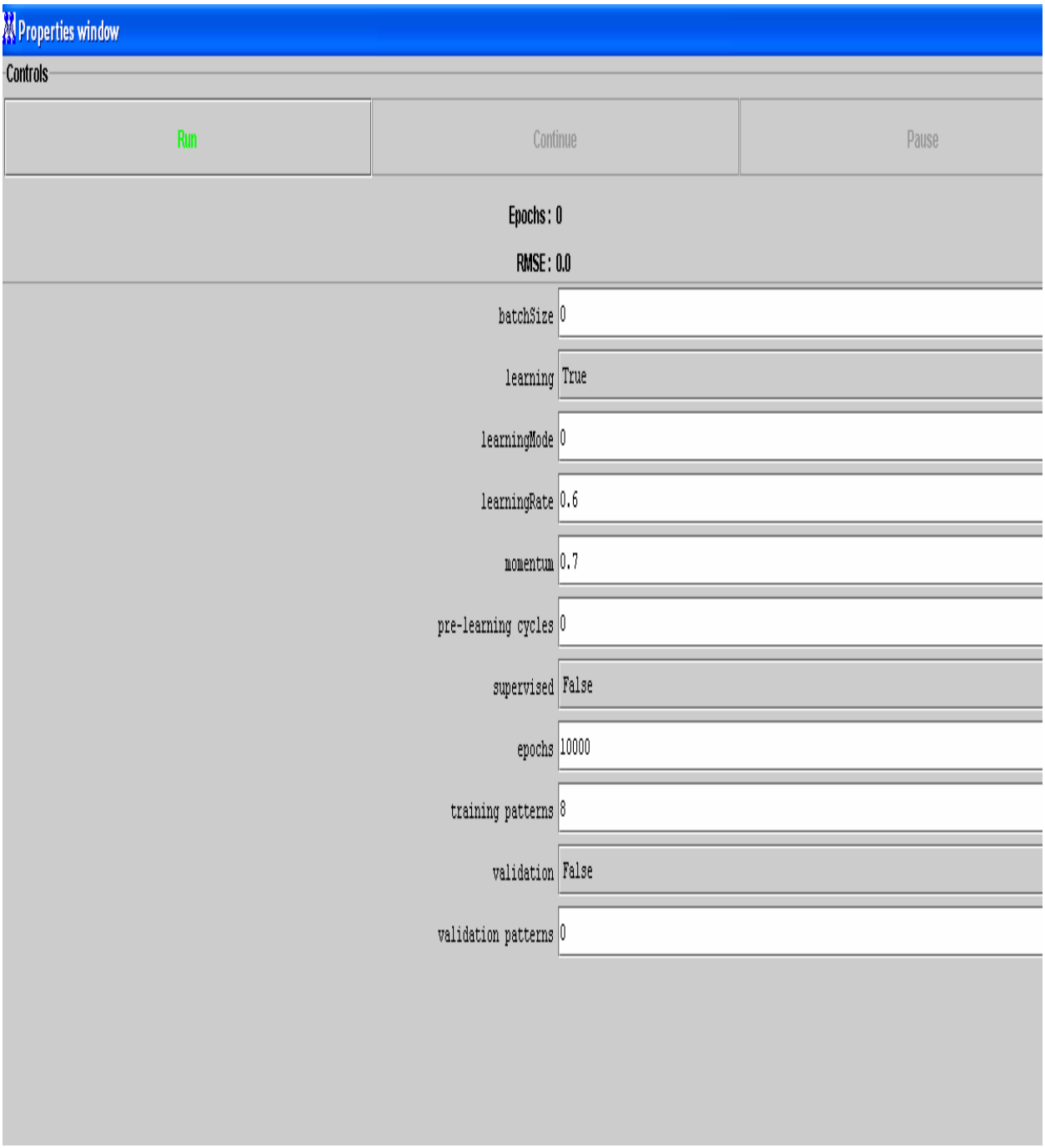

Figure 5: Parameter entry for full adder training.

The neural network is trained until the RMSE is close to zero. Once the RMS error becomes approximately zero, the network is ready for testing. The full adder testing is as shown in Figure 6.

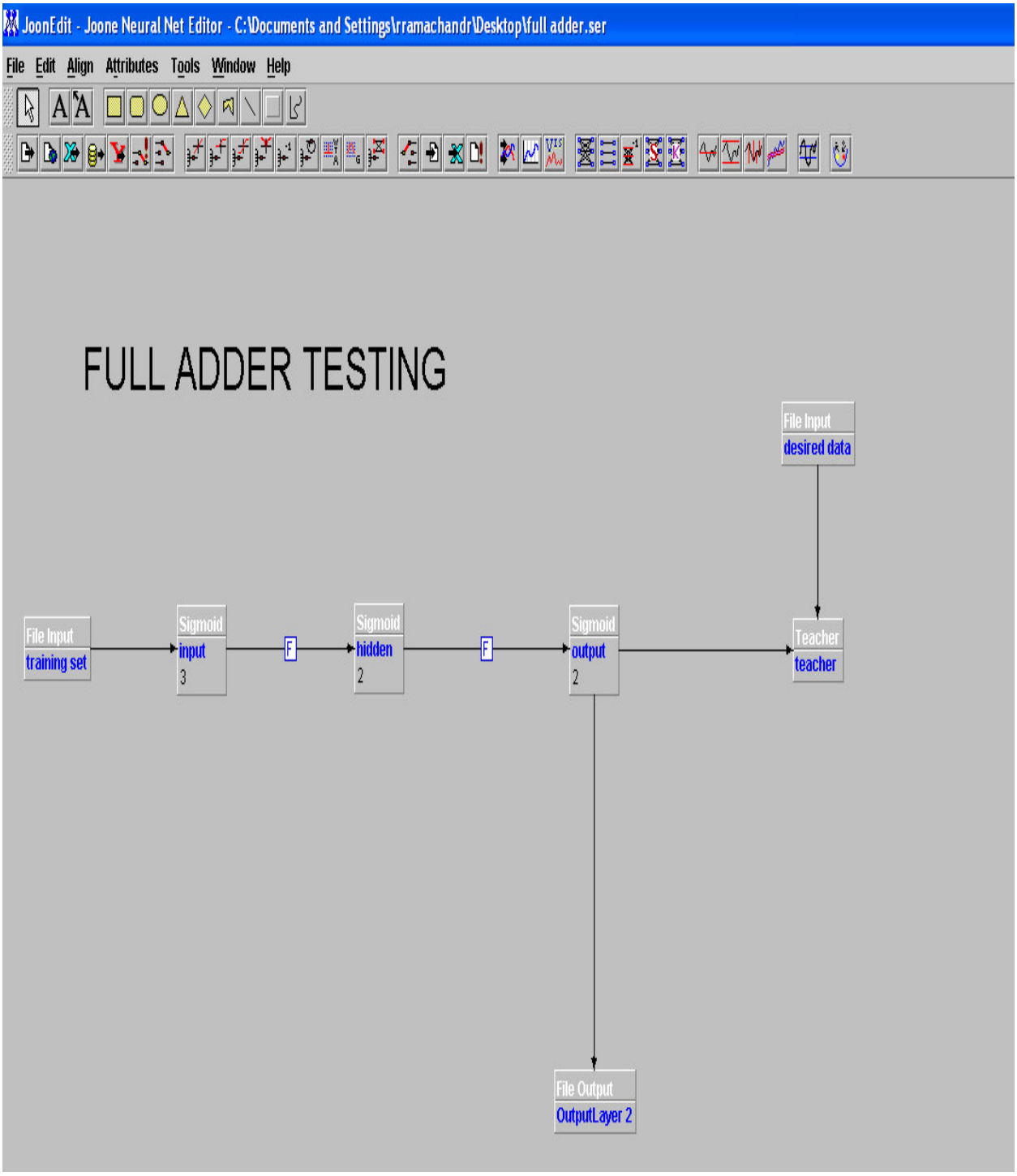

Figure 6: Full Adder testing.

The testing is carried out using a file output layer that helps in storing the expected values. The parameters are again entered with epoch of 1 and learning is set to false. This is shown in Figure 7.

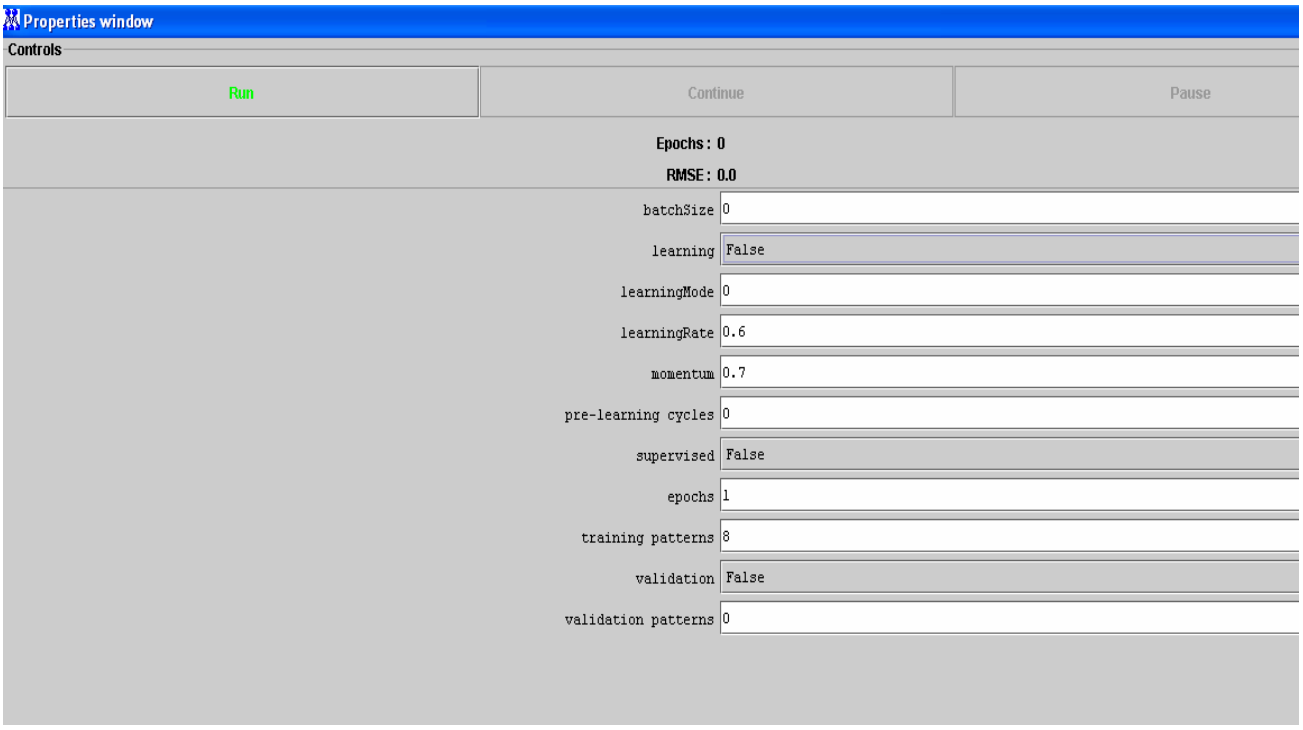

Figure 7: Parameter entry for testing.

The network is tested for best results. The results are displayed on the notepad file stored in the file output layer as shown in Figure 8.

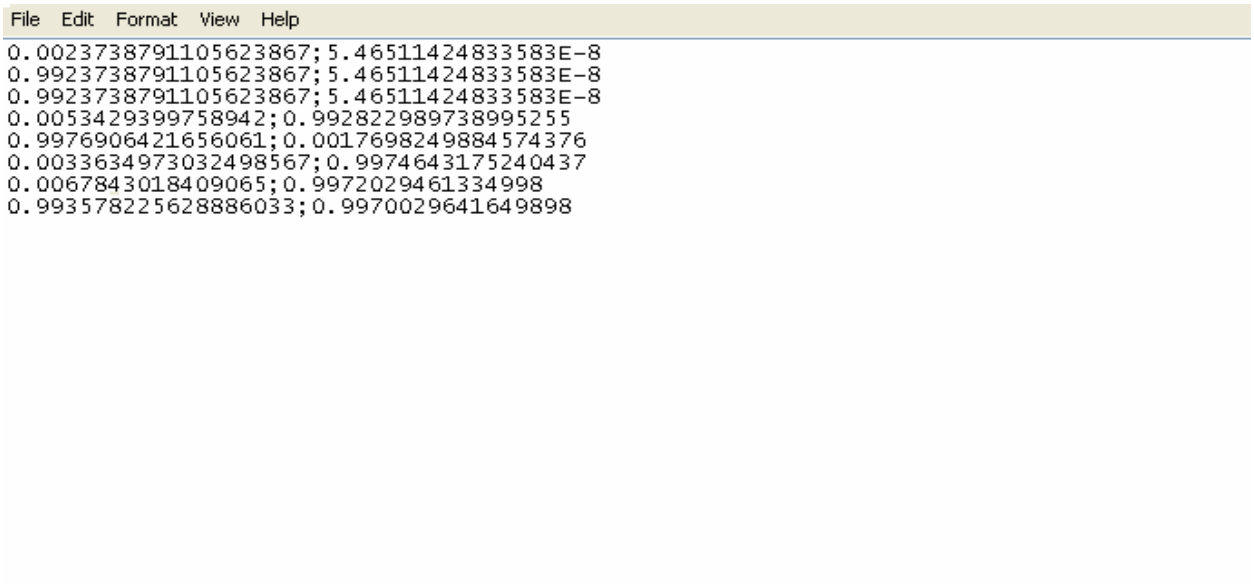

Figure 8: Results of testing are verified using a notepad file.

4. RESULTS

The table 1 shows the output of a sample run of the JOONE program for a full adder. The inputs and the desired outputs are entered and the neural network is tested. The root mean-square error is then noted for different momentum and learning parameters. In this case, the network consists of 3 hidden neurons and epoch is set to 1000. The momentum M specified by the user increases from 0.2 to 0.6 vertically and beta represents the learning rate. Choosing right values of momentum and learning rate improved the performance of the neural network.

Table 1: RMSE after 1000 epochs.

M/Beta	0.2	0.3	0.4	0.5	0.6
0.2	0.006439	0.006429	0.006182	0.005902	0.005612
0.3	0.006435	0.006395	0.006125	0.005840	0.005548
0.4	0.006398	0.006374	0.006327	0.005770	0.005476
0.5	0.006354	0.006309	0.006049	0.006226	0.005392
0.6	0.006301	0.006230	0.005956	0.005669	0.006080

5. FUTURE RESEARCH

With the rapid development of VLSI, reconfigurable computing and FPGA make fast massively parallel computations more affordable and neural networks with any architecture can be easily designed. Since most of the signals in real world are analog in nature, many researchers have developed analog neural networks (Valle, et al., 1995; Valle, et al., 1996; Brea, et al; 1998). But problems like noise, power supply instability, cross talk and temperature variations become inevitable. Due to these reasons, digital logic is used as an alternate solution. With the substantial advances in FPGA technology, programmable logic provides a new way of designing and developing systems. The research would help in understanding hardware aspects of neural networks for modeling and verification of digital logic circuits.

6. DISCUSSION AND CONCLUSION

This paper proposed a neural network for designing digital logic circuits using Joone software. The software testing provided expected results. This project not only helped the students to grasp the basics of neural network and digital logic gates but also to solve wide variety of digital logic design problems using neural models. Motivated by these results, future investigations will address the design and testing of neural networks using hardware. Today, hardware testing is

gaining popularity. Neural networks are inherently massively parallel in nature (Nordstrom, et al., 1992) which means that they lend themselves well to hardware implementations, such as Field programmable arrays. With the use of reconfigurable hardware, the performance of the neural networks can be improved. This advantage will be exploited in the test generation of combinational and sequential circuits using neural networks.

REFERENCES

Brea, V.M., Vilariño, D.L., and Cabello, D. (1998). *Active Contours with Cellular Neural Networks: An Analog CMOS Realization*, *Signal Processing and Communications*. (J. Ruiz-Alzola ed.), pp.419-422, IASTED/Acta Press.

DACS Store, Artificial Neural Networks Technology, http://www.dacs.dtic.mil/techs/neural/neural2.html.

Marrone. P, Java Object Oriented Neural Engine, GUI Editor, http://www.jooneworld.com/docs/guiEditor.html.

Nordstrom, T., Davis, E.W., and Svensson, B. (1992). *Issues and applications driving research in non-conforming massively parallel processors*. (In I. D. Scherson, editor, Proceedings of the New Frontiers, a Workshop of Future Direction of Massively Parallel Processing), pp. 68-78. McLean, Virginia.

Principe, C.Jose., Euliano, R.Neil., and Lefebvre, W.Curt. (2000). *Neural and Adaptive Systems: Fundamentals Through Simulations,* John Wiley and Sons.

Valle, M., Chiblé, H., Baratta, D., and Caviglia, D.D. (October, 1995). *Evaluation of synaptic multipliers behavior in the back propagation analog hardware implementation*, **vol. 2**, pp 357-362. Proceedings of ICANN'95 Fifth International Conference on Artificial Neural Networks, Paris.

Valle, M., Caviglia, D.D., and Bisio, G.M. (1996). *An experimental analog VLSI neural network with on-chip back-propagation learning, Analog Integrated Circuits & Signals Processing,* **vol. 9**, pp. 25-40. Kluwer Academic Publishers, Boston.