A New Method for Hardware Design of Multi-Layer Perceptron Neural Networks with Online Training

R. Rezvani[*], M. Katiraee[**], A. H. Jamalian[*], SH. Mehrabi[*] and A. Vezvaei[***]
[*] Sama Technical and Vocational Training College, Islamic Azad University, Andisheh Branch, Andisheh, IRAN
[**] Department of Computer Engineering, Islamic Azad University, South Tehran Branch, Tehran, IRAN
[***] Department of Computer Engineering, Amirkabir University of Technology, Tehran, IRAN
{rezvani, jamalian, mehrabi}@andisheh-samacollege.ir, m_katiraee@azad.ac.ir, vezvayee@aut.ac.ir

***Abstract*—In this paper, a Multi-Layer Perceptron (MLP) has been simulated using synthesizable VHDL code. This is a well-known artificial neural network tool which is widely used for classification and function approximation problems. Our proposed model has special flexibilities and user can determine his/her proper parameters such as number of layers and number of neurons in each layer. The learning phase in this network model is online and after this phase, the network starts the operational phase immediately. Unlike some other similar models, in this hardware model there is no restriction on weights of the network. Weights can define as floating point type and synthesize easily. We have implemented the simulation of network described above, in two, three and four layer structure for a problem of numeric patterns recognition. The simulation results show that the network has been properly trained and can differentiate input patterns from each other with a negligible error.**

Keywords—Artificial Neural Network; Multi-Layer Perceptron (MLP); Hardware Implementation; VHDL.

I. INTRODUCTION

An Artificial Neural Network (ANN) is a tool with parallel architecture in large scale which is able to solve complicated problems in many fields such as Modeling, optimization, classification and etc. Some of most important applications of ANNs are pattern recognition, detection and identification of human speech, converting black and white images to color images [1].

Since far past time, most of simulations of various artificial neural networks were done using software simulators and high level programming languages; but due to parallel structure of artificial neural networks, their hardware implementation maybe faster and more efficient than software simulations. Hence to achieve an optimal performance in real-world applications (especially in cases where continuous and real-time training is essential) hardware implementation is strongly recommended.

Since past two decades, many researchers work on hardware implementation of ANNs on various hardware modules such as FPGA and CPLD [2-10]. For instance, many researchers have focused on hardware implementation of a simple neuron [11, 12]. On the other hand, some hardware implementations are so complicated [13, 14] and need complex processor structures that SIMD processor structure is one of them [15].

In many hardware implementations of neural networks, training of the network is offline. It means that first the network will be trained using a software training algorithm, then using the tuned weights obtained from training phase, the network will be implemented on hardware for operational phase. On the other hand, in some other researches, training phase of the network is implemented on hardware [13, 15]. Of course, in such implementations, usually more than one FPGA is used which is not efficient from point of view of hardware resources, power consumption and cost. Although some researches have done in field of hardware implementation of MLP, they were costly and didn't have generality and were mostly used for special purpose applications.

Besides, in many applications, using floating point numbers for presentation of neuron weights and inputs of the network is inevitable. However up to this moment, no successful implementation of floating point numbers in neural networks is reported and usually some constraints for the weights and inputs are considered.

In this paper, a new method for hardware implementation of general purpose multi-layer perceptron neural networks is presented. This system is able to receive the number of network layers and neurons in each layer from the user. Also, network data, especially neuron weights and inputs can be in floating point form and all the arithmetic operations inside the network, including summation, multiplication, calculation of excitation function and its first order derivative are implemented using floating point algorithms.

Rest of the paper is organized as follows: in section II, preliminary concepts of artificial neural networks including multi-layer perceptron networks, training of network and Back-Propagation (BP) training algorithm are described. Section III is devoted to description of proposed structure and performance of the proposed method. In section IV, the simulation results are presented and finally conclusions are covered in section V.

II. Preliminary Concepts

In this section, main concepts and basic definitions of the proposed structure will be reviewed. This section contains three subsections: A. Introducing ANN, B. Multi-layer Perceptron networks and finally C. Back-Propagation training algorithm.

A. *Artificial Neural Networks (ANN)*

Neural networks are common tools for knowledge of learning. In practice, a neural network is a dynamic non-linear system with high complexity. Artificial neural networks consist of simple elements named neurons that work parallel together and are inspired from biological nervous systems. Although structure and functionality of a single neuron is so simple, but the behavior of a neural network system composed of to many neurons, could be so complicated.

Like nature, the operation of the network is largely determined by connections between its elements. By adjusting the weights of connections between neurons, a neural network could be trained to perform a particular operation. Generally, artificial neural networks are large scale parallel structures that are able to solve complicated problems in many fields such as modeling, optimization, classification and etc.

Nowadays, artificial neural networks are used in many applications due to their learning ability from input patterns that give them the necessary knowledge for generating appropriate response to the new input patterns.

Some of main applications of artificial neural networks are signal processing, pattern recognition, pattern classification and robot control systems.

Neural networks have different types. The most important types of artificial neural networks are as follow:

- Multi-layer perceptron neural networks (MLP)
- Spiking Neural Networks (SNN)
- Hopfield Neural Networks (HNN)
- Radial Basis Function Neural Networks (RBF)
- Probabilistic Neural Networks (PNN)
- General Regression Neural Networks (GRNN)

Since the Multi-layer perceptron neural networks are more common than the other types of neural networks, and most of the hardware implementations are based on this type of neural networks, so in this research, Multi-layer perceptron neural networks are used for simulations and finally hardware implementation. In next subsection, MLP is discussed more.

B. *Multi-Layer Perceptron Networks (MLP)*

These types of networks that are known as MLP are most widely used neural networks. The structure of a MLP is layered and consists of an input layer, an output layer and one or more hidden layers between these two layers; however, a MLP could have no hidden layer. The structure of a 3 layer MLP is shown in Figure 1. In MLP, the input layer neurons act like buffers and their main task is feeding the network inputs to next layer neurons in training phase and also operational phase.

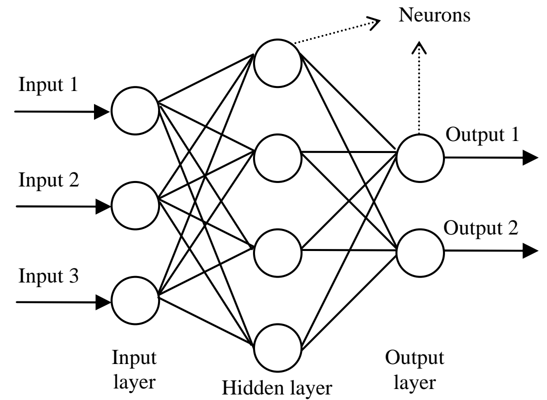

Figure 1. The structure of a 3 layer MLP

But for neurons in hidden or output layers, the output value of each neuron is calculated using the following formula:

$$\mathrm{net}_i = \sum_{j \in A} O_j W_{ji} \qquad \forall i : i \in B \tag{1}$$

Where A means set of neurons in a layer, B means set of neurons in the next layer, O_j is the output of neuron j, and W_{ji} is the connection weight between neurons j and i. Indeed, Eq. 1 calculates the sum of products of a neuron's inputs in corresponding weight of that input neuron. Finally using Eq. 2 neuron's output will be calculated:

$$O_i = f(\mathrm{net}_i) = \frac{1}{1 + e^{-\mathrm{net}_i}} \tag{2}$$

The excitation function used in formula 2 is known as sigmoid function. This function could be in another form. But normally in simulations and implementations, sigmoid function or Hyperbolic Tangent function are used. In MLP, output of a neuron goes to all the neurons in next layer, until the final outputs of the network would be generated.

C. *Back-Propagation (BP) Training Algorithm*

In most cases, initial weights of MLP, aren't accurate and the network should be able to correct them in order to generate appropriate output values. So, each neural network needs an algorithm for correction of initial weights. This algorithm is known as Learning Algorithm of Neural networks.

Many algorithms have been proposed for learning of a MLP neural network. One of the most widely used of these algorithms is Back-Propagation (BP) learning algorithm.

BP algorithm is a good solution for training a MLP feed-forward neural network with several layers. This algorithm begins with feeding input data through the network from input layer to hidden and output layers using Eq. 1 and 2. Eq. 1 takes the output values of a layer and feeds them to the next layer. This action will be done for each neuron in next layer and a weighted sum of all the neuron's output values in previous

layer will be generated. Then, this weighted sum will be feed to Eq. 2 named sigmoid function of neuron in order to calculate each neuron's output value.

After calculating output values for all neurons for an input pattern (except input neurons), the algorithm continues with finding error values for each non-input neuron; this process begins with calculation of error values for all neurons in output layer using the following equation:

$$\delta_i = f'(net_i).(T_i - O_i) \qquad \forall i : i \in C \tag{3}$$

In this formula, *C* is set of output neurons, T_i is the expected output value for neuron *i*, $f'(\ )$ is the first order derivative of Eq. 2 and δ_i is the error value of neuron *i*. Then, the error values calculated for output neurons should be propagated to previous layer in order to calculate error values for hidden neurons using Eq. 4.

$$\delta_i = f'(net_i).\sum_{j \in E} \delta_j W_{ij} \qquad \forall i : i \in D \tag{4}$$

In this formula, *D* is set of neurons of a non-input layer and E is set of neurons of the next layer. This formula will be calculated for the entire hidden layers inside the network. After calculating the output and error values for all the non-input neurons, the weights of neurons could be corrected. The amount that each weight should vary will be calculated using formula 5;

$$\Delta W_{ij} = \eta.O_i.\delta_j \qquad \forall i, j : i \in A, j \in B \tag{5}$$

In above formula, η is the learning parameter and controls the amount of weight variations in each step of training process. Finally, new values for weights in step t+1 will be calculated using formula 6;

$$W_{ij_{t+1}} = W_{ij_t} + \Delta W_{ij} \tag{6}$$

III. THE PROPOSED METHOD

In this part, first we will describe the proposed structure, in subsection *A* and then the operation and performance of our structure will be studied in details in subsections *B* and *C*.

A. Structure

Figure 2 shows the structural view of the proposed method in this paper for implementation of a multi-layer perceptron neural network. The modular structure shown in Figure 2 could be implemented on FPGA simply. This block diagram shown in Figure 2 consists of two input ports and one output port and the user can communicate with the network using these ports.

Most of problems solvable using multi-layer perceptron neural networks need maximum 2 hidden layers for generating appropriate outputs, therefore in this research; we have considered maximum 2 hidden layers for our network. In this modular structure, the control unit receives the initial data (including number of layers needed and number of neurons in each layer, the train dataset and initial weights for all neurons in hidden and output layers) via an input port and feed them to existing neurons in all layers. Also, coordinating between different layers is another task of control unit; when a layer prepares some data for the other layer in both training and operational phases, sends a signal to control unit. Then this unit sends signals to all the neurons in destination layer and these neurons will start their operations simultaneously.

In the proposed structure, the coordination between all the layers is controlled by control unit using the above method. Also during the run of Back-Propagation algorithm, when a layer prepares its error values for the previous layer, informs neurons of the destination layer via signals generated by control unit.

B. Functionality and Performance Evaluation

The proposed network in this research consists of two phases: training phase and operational phase. Since the training phase is online, the user should be able to deliver the basic information (such as number of layers, number of neurons in each layer and initial weights of neurons) with the train dataset to the network. Also, after completion of the training phase and at the start of operational phase, the user should be able to feed the operational input data (that sometimes are impregnated with perturbation) to the network so that the network could generate the appropriate responses for each input dataset.

Since the user should communicate with the network module, we have considered two input ports for the network that are visible in Figure 2. The user stores the basic information and necessary data for train phase in an EEPROM which this memory is connected to the network via the first port. Also after completion of the training phase, the user should deliver the operational data to the network; for this purpose the operational data will be stored on another

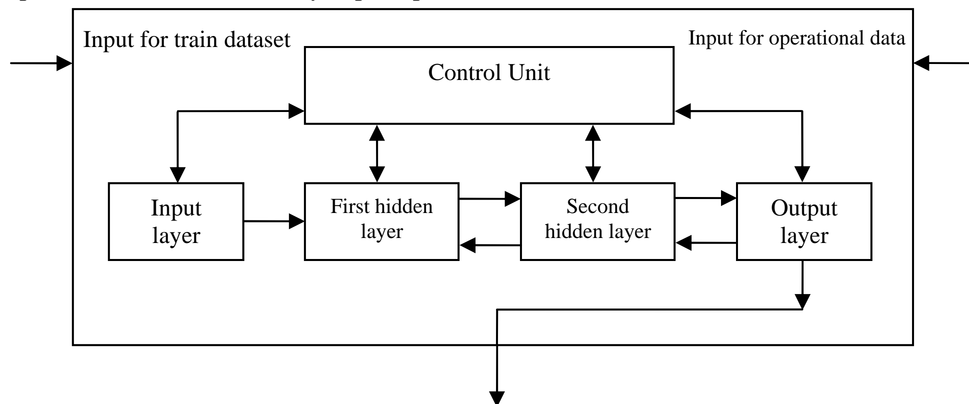

Figure 2: Block diagram of proposed structure for a 4 layer multi-layer perceptron neural network

EEPROM memory that is connected to the network via second port.

In operational phase, the network receives its input data from the second EEPROM and produces appropriate responses for each operational dataset. The final outputs could be sent to a display screen or any suitable output device via the output port. The block diagram of these modules is shown in Figure 3.

Because choosing the number of layers is one of user's authorities and the maximum of hidden layers allowed in this network is 2, the user could choose 0, 1 or 2 hidden layers and therefore, could have a MLP network with 2, 3 or 4 layers.

In addition the network user is able to decide number of neurons for each layer. Obviously the maximum number of neurons for whole the network in final implementation is constrained to the FPGA device used and couldn't be infinite. So the user should select a number between 1 and the maximum neurons considered for each layer.

The precision of output values depends on number of hidden layers, number of neurons in each layer, learning parameter and number of iterations in training phase.

According to what was said, the operation of the proposed system in this paper could be summarized in steps below:

1. Deciding number of layers and also number of neurons for each layer.
2. Determining initial weights of the neurons and number of iterations for learning phase by the user.
3. Defining train set data by the user.
4. Storing prepared data in steps 1 to 3 on first EEPROM by the user.
5. Running training phase by the network.
6. Defining Operational data set and also noisy data and storing them on second EEPROM by the user.
7. Running operational phase for each operational data by the network and showing the final results on display screen.

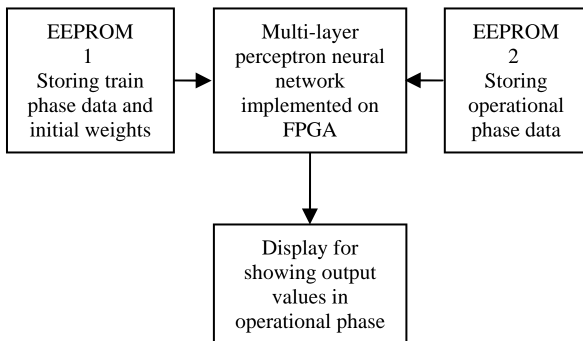

Figure 3: Connection method for connecting MLP neural network to EEPROMs and I/Os for receiving data from user and displaying the results

C. *Performing Floating Point Data Type*

In many applications using floating point numbers is something common and inevitable. Also, neural networks usually use floating point weights and inputs. Yet, fewer efforts have been done for implementing artificial neural networks using floating point numbers on hardware platforms ,specially FPGA, and based on this study, no Successful results have been observed up to this moment.

Of course, in some researches some constraints have been applied on weights and data of the network; For instance, some researches claim that the learning of artificial neural networks could be performed using integer weights and a common solution for simplifying the design is converting real numbers to integer numbers that is used in some researches.

Obviously, limiting network data and weights would reduce the generality of neural networks in solving different kinds of problems.

In this study, floating point algorithms have been designed and used for implementing arithmetic operations needed in multi-layer perceptron neural networks in order to solve the problem of working with floating point data.

These operations are: summation, subtraction, multiplication, calculating excitation function of neurons and its first order derivative.

It should be noted that for expression of weights and network inputs, single precision floating numbers based on IEEE754 standard have been used.

Implementing Excitation Function of Neurons: Implementing summation, subtraction and multiplication operations using floating point algorithms is simple, but implementing excitation functions that are mainly nonlinear, would be very costly. Therefore, in most implementations, piece linear approximation methods or lookup tables have been used instead.

As mentioned in the first, in this study we have used sigmoid function for generating neural outputs. Due to the wide range of floating point numbers, implementing floating point numbers using lookup tables was nearly impossible; so we interpolated the sigmoid function using polynomials and approximated the sigmoid function using polynomials of degree 3 with relatively good precision. The function interpolation is shown in Eq. 7:

$$S(n)=\begin{cases} 0 & n<-5 \\ 0/0466\,n^3+0/0643\,n^2+0/3064\,n+0/5131 & -5\le n<-1/5 \\ -0/0166\,n^3+0/249\,n+0/5 & -1/5\le n<1/5 \\ 0/0466\,n^3-0/0643\,n^2+0/3064\,n+0/5131 & 1/5\le n<5 \\ 1 & n\ge 5 \end{cases} \quad (7)$$

Implementing First Order Derivative of Excitation Function: If we directly used polynomials obtained for interpolating sigmoid function to calculate the first order derivative of the function, the calculation errors could be high. So we performed some arithmetic operations on sigmoid function and Succeeded to express the first derivative based on the function, itself. The steps of these operations are shown in formula 8. In this formula, F(x) is the sigmoid function.

Using formulas 7 and 8 and also floating point algorithms, the sigmoid function and its first order derivative are easily synthesizable and ready for hardware implementations of multi-layer perceptron neural networks.

$$\frac{dF(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{e^{-x}-1+1}{(1+e^{-x})^2} = \frac{1}{(1+e^{-x})} - \frac{1}{(1+e^{-x})^2} =$$
$$-(\frac{1}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})} + \frac{1}{4}) + \frac{1}{4} = -(\frac{1}{(1+e^{-x})} - \frac{1}{2})^2 + \frac{1}{4} \quad (8)$$
$$= -(F(x) - \frac{1}{2})^2 + \frac{1}{4}$$

IV. SIMULATION RESULTS

The Program used for simulating the proposed network structure in previous subsection has been written in VHDL language and the simulator software is ModelSim SE. Simulation stages in this study, has been done for multi-layer perceptron neural networks with 2, 3 and 4 layers. In this section the simulation results will be surveyed.

In order to check the accuracy and efficiency of our model, we have tried to solve the pattern recognition problem for digits 0 to 9 by our MLP network. Figure 4 shows how each digit is demonstrated. As you see, each digit is designed and displayed by 35 dots. Therefore, it's obvious that for receiving inputs proportional to each of these numeric patterns requires a MLP network with 35 neurons in its input layer. Also, separating 10 distinctive numeric patterns in network output requires 10 neurons in output layer. In other words, each output neuron determines one of digits and whenever the input pattern conforms to one of digits 0 to 9, the appropriate output value will be created.

We have performed the simulation of mentioned network for MLP with 0, 1 or 2 hidden layers and each time, we have changed number of neurons in hidden layers. On the other hand, initial network weights of hidden and output neurons have been chosen completely random in range -1 to 1 and training phase has started with these values. It is necessary to say that in all cases number of neurons in output layer is 10 and each neuron corresponds to one of digits 0 to 9.

A. *Network Learning Parameters*

0011000 0101000 0001000 0001000 0001000	0011100 1000001 0000100 0010000 1111111	0011100 1000001 0000110 1000001 0011100	0000010 0000110 0010010 1111111 0000010	1111111 1000000 1111110 0000001 0011100
0011100 0100000 1001110 1000001 0011100	1111110 0000100 0001000 0010000 1000000	0011100 1000001 0011100 1000001 0011100	0011100 1000001 0111001 0000010 0011100	0111110 1000001 1000001 1000001 0111110

Figure 4: Numeric patterns, top row represent digits 1, 2, 3, 4, 5 from left to right and bottom row represent digits 6, 7, 8, 9, 0 from left to right

The proposed MLP neural network has been simulated with different layers between 2 and 4. In each step of simulation, number of iterations in train phase was different and as the results will be seen in the table, by increasing number of iterations in train phase, the generated errors in output neurons have been reduced.

On the other hand, the network structure is able to be trained with both fixed and variable learning parameter. It should be noted that the learning parameter varies with the number of iterations passed, and its variation in each repetition of train phase is calculated using Eq. 9:

$$New_\eta = \frac{Old_\eta}{(1+\alpha * iteration)} \quad (9)$$

In above equation, η is train coefficient of network, iteration is iteration counter and α is variation coefficient of learning parameter.

B. *Analyzing the results*

In this subsection, first some instances of output neuron waveforms in our simulations will be shown. Then a summary of simulation results for different layers, neurons and iterations in train phase will be listed in Table I. Note that in all the figures below, index 0 in data_out array relates to digit 1 in output layer, and index 9 in data_out array relates to digit 0 in output layer and so on.

TABLE I. SIMULATION RESULTS

Network Type	Number of Input Neurons	Number of Neurons in First Hidden Layer	Number of Neurons in Second Hidden Layer	Number of Neurons in Output Layer	Number of Iterations	Output Error Values
2 layer	35	-	-	10	200	5 %
2 layer	35	-	-	10	500	3 %
3 layer	35	5	-	10	500	8 %
3 layer	35	5	-	10	1000	5 %
3 layer	35	10	-	10	500	5 %
3 layer	35	10	-	10	1000	3 %
4 layer	35	10	10	10	500	5 %
4 layer	35	10	10	10	1000	3 %
4 layer	35	5	5	10	1000	7 %
4 layer	35	5	5	10	2000	4 %

As a first example, Figures 5 to 14 show the output values generated in a 3 layer MLP for input patterns of digits 0 to 9

with 1000 iterations in train phase. As you see, the error rate

Figure 5: A 3 layer MLP network output for input pattern 0

Figure 6: A 3 layer MLP network output for input pattern 1

Figure 7: A 3 layer MLP network output for input pattern 2

Figure 8: A 3 layer MLP network output for input pattern 3

of output values in this case is about 3%.

As the second example, Figures 15 to 19 show the output values generated in a 2 layer MLP for input patterns of digits 1 to 5 with 100 iterations in train phase. As you see, due to relatively small number of iterations in train phase, the error rate of output values is variable and between 5% and 8%.

Figure 9: A 3 layer MLP network output for input pattern 4

Figure 10: A 3 layer MLP network output for input pattern 5

Figure 11: A 3 layer MLP network output for input pattern 6

Figure 12: A 3 layer MLP network output for input pattern 7

V. CONCLUSION

Figure 13: A 3 layer MLP network output for input pattern 8

Figure 14: A 3 layer MLP network output for input pattern 9

Figure 15: A 2 layer MLP network output for input pattern 1

Figure 16: A 2 layer MLP network output for input pattern 2

In this paper a multi-layer perceptron neural network was implemented using VHDL hardware description language. The main advantages of this study are as below.

First, number of layers and also number of neurons in each layer are determined by the user; second, the train phase is online and is done on hardware; third, neural weights could be in floating point from.

Figure 17: A 2 layer MLP network output for input pattern 3

Figure 18: A 2 layer MLP network output for input pattern 4

Figure 19: A 2 layer MLP network output for input pattern 5

Running both train and operational phases on hardware is much faster than software implementations; because, in hardware implementations the neurons are interacting together in parallel.

And finally as future work, other types of artificial neural networks could be implemented using hardware description languages and their speedup and advantages could be surveyed and analyzed.

REFERENCES

[1] S. Haykin, “*Neural Networks: A Comprehensive Foundation*”, New York: Macmillan Collage Publishing Company, 1994.

[2] do A , Ferreira A.P , da S Barros E.N , "A high performance full pipelined arquitecture of MLP Neural Networks in FPGA ", 17th IEEE International Conference on Electronics, Circuits, and Systems (ICECS), PP : 742 - 745 , 2010.

[3] Andres A , Carlos A.P , Eduardo S , "An FPGA platform for on-line topology exploration of spiking neural networks ", Conference on Microprocessors and Microsystems, Vol 29 , PP: 211 - 223 , June 2007.

[4] Pearson M.J , Melhuish.C , et al. "Design and FPGA implementation of an embedded real-time biologically plausible spiking neural network

processor", Conference on Field Programmable Logic and Applications , PP: 582 - 585 , Aug 2006.

[5] B. Glackin , et al. "A Novel Approach for the Implementation of Large Scale Spiking Neural Networks on FPGA Hardware", LNCS: Computational Intelligence and Bioinspired Systems, Vol 3512 , PP: 552 - 563 , 2005.

[6] Pedro F , Pedro R , Ana A , Fernando M.D , "Artificial Neural Networks Processor – A Hardware Implementation Using a FPGA", LNCS: Field Programmable Logic and Applications, Vol 3203 , PP: 1084 - 1086 , 2004.

[7] Bellis S , et al. "FPGA implementation of spiking neural networks - an initial step towards building tangible collaborative autonomous agents", Proceedings of IEEE International Conference on Field-Programmable Technology, PP: 449 - 452 , 2004.

[8] Wang Q , Yi B , Xie Y , Liu B , "The hardware structure design of perceptron with FPGA implementation", IEEE International Conference on Man and Cybernetics Systems, Vol 1 , PP: 762 - 767 , Oct 2003.

[9] Jihan Z , Peter S , "FPGA Implementations of Neural Networks - A Survey of a Decade of Progress", LNCS: Field Programmable Logic and Applications, Vol 2778 , PP: 1062 - 1066 , 2003.

[10] Eva M.O , Antonio C , Eduardo R , Richard R.C , "FPGA Implementation of a Perceptron-Like Neural Network for Embedded Applications", LNCS: Artificial Neural Nets Problem Solving Methods, Vol 2687 , PP: 1 - 8 , 2003.

[11] Noory B , Groza V , "A reconfigurable approach to hardware implementation of neural networks ", Conference on Electrical and Computer Engineering , Vol 3 , PP: 1861 - 1864 , May 2003.

[12] Chujo N., Kuroyanagi S., Doki S., Okuma S., *"An iterative calculation method of the neuron model for hardware implementation"*, IEEE IECON, Vol. 1, pp. 664-671, 2000.

[13] Eldredge J. G., Hutchings B. L., *"RRANN: a hardware implementation of the backpropagation algorithm using reconfigurable FPGAs"*, IEEE International conference on neural networks, Vol. 4, pp. 2097-2102, 1994.

[14] Girau, B.; Tisserand, A.; "On-line arithmetic-based reprogrammable hardware implementation of multilayer perceptron back-propagation", Microelectronics for Neural Networks, 1996., Proceedings of Fifth International Conference on 12-14 Feb. 1996 Page(s):168 – 175.

[15] Seok B.Y , Young J.K , Sung S.D , Chong H.L , "Hardware implementation of neural network with expansible and reconfigurable architecture", Proceedings of the 9th International Conference on Neural Information, Vol 2 , PP: 970 - 975 , Nov 2002.