We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

6,900
Open access books available

185,000
International authors and editors

200M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

Chapter 8

Integrating Modularity and Reconfigurability for Perfect Implementation of Neural Networks

Hazem M. El-Bakry

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/53021

1. Introduction

In this chapter, we introduce a powerful solution for complex problems that are required to be solved by using neural nets. This is done by using modular neural nets (MNNs) that divide the input space into several homogenous regions. Such approach is applied to implement XOR functions, 16 logic function on one bit level, and 2-bit digital multiplier. Compared to previous non- modular designs, a salient reduction in the order of computations and hardware requirements is obtained.

Modular Neural Nets (MNNs) present a new trend in neural network architecture design. Motivated by the highly-modular biological network, artificial neural net designers aim to build architectures which are more scalable and less subjected to interference than the traditional non-modular neural nets [1]. There are now a wide variety of MNN designs for classification. Non-modular classifiers tend to introduce high internal interference because of the strong coupling among their hidden layer weights [2]. As a result of this, slow learning or over fitting can be done during the learning process. Sometime, the network could not be learned for complex tasks. Such tasks tend to introduce a wide range of overlap which, in turn, causes a wide range of deviations from efficient learning in the different regions of input space [3]. Usually there are regions in the class feature space which show high overlap due to the resemblance of two or more input patterns (classes). At the same time, there are other regions which show little or even no overlap, due to the uniqueness of the classes therein. High coupling among hidden nodes will then, result in over and under learning at different regions [8]. Enlarging the network, increasing the number and quality of training samples, and techniques for avoiding local minina, will not stretch the learning capabilities of the NN classifier beyond a certain limit as long as hidden nodes are tightly coupled, and hence cross talking

INTECH
open science | open minds

© 2013 M. El-Bakry; licensee InTech. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

during learning [2]. A MNN classifier attempts to reduce the effect of these problems via a divide and conquer approach. It, generally, decomposes the large size / high complexity task into several sub-tasks, each one is handled by a simple, fast, and efficient module. Then, sub-solutions are integrated via a multi-module decision-making strategy. Hence, MNN classifiers, generally, proved to be more efficient than non-modular alternatives [6]. However, MNNs can not offer a real alternative to non-modular networks unless the MNNs designer balances the simplicity of subtasks and the efficiency of the multi module decision-making strategy. In other words, the task decomposition algorithm should produce sub tasks as they can be, but meanwhile modules have to be able to give the multi module decision making strategy enough information to take accurate global decision [4].

In a previous paper [9], we have shown that this model can be applied to realize non-binary data. In this chapter, we prove that MNNs can solve some problems with a little amount of requirements than non-MNNs. In section 2, XOR function, and 16 logic functions on one bit level are simply implemented using MNN. Comparisons with conventional MNN are given. In section 3, another strategy for the design of MNNS is presented and applied to realize, and 2-bit digital multiplier.

2. Complexity reduction using modular neural networks

In the following subsections, we investigate the usage of MNNs in some binary problems. Here, all MNNs are feedforward type, and learned by using backpropagation algorithm. In comparison with non-MNNs, we take into account the number of neurons and weights in both models as well as the number of computations during the test phase.

2.1. A simple implementation for XOR problem

There are two topologies to realize XOR function whose truth Table is shown in Table 1 using neural nets. The first uses fully connected neural nets with three neurons, two of which are in the hidden layer, and the other is in the output layer. There is no direct connections between the input and output layer as shown in Fig.1. In this case, the neural net is trained to classify all of these four patterns at the same time.

x	y	O/P
0	0	0
0	1	1
1	0	1
1	1	0

Table 1. Truth table of XOR function.

http://dx.doi.org/10.5772/53021

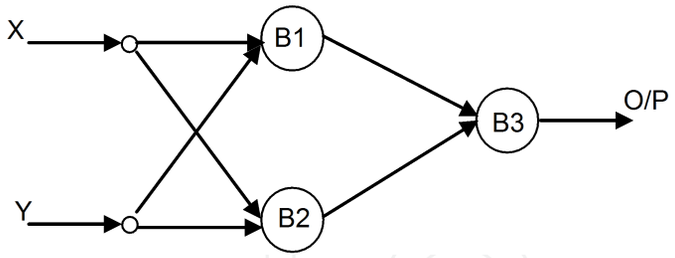

Figure 1. Realization of XOR function using three neurons.

The second approach was presented by Minsky and Papert which was realized using two neurons as shown in Fig. 2. The first representing logic AND and the other logic OR. The value of +1.5 for the threshold of the hidden neuron insures that it will be turned on only when both input units are on. The value of +0.5 for the output neuron insures that it will turn on only when it receives a net positive input greater than +0.5. The weight of -2 from the hidden neuron to the output one insures that the output neuron will not come on when both input neurons are on [7].

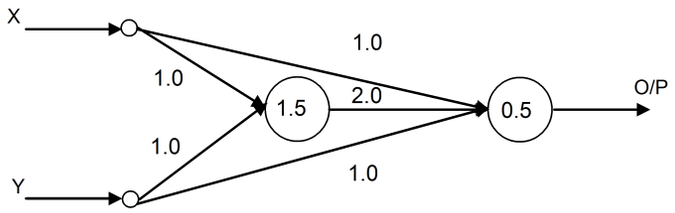

Figure 2. Realization of XOR function using two neurons.

Using MNNs, we may consider the problem of classifying these four patterns as two individual problems. This can be done at two steps:

1. We deal with each bit alone.
2. Consider the second bit Y, Divide the four patterns into two groups.

The first group consists of the first two patterns which realize a buffer, while the second group which contains the other two patterns represents an inverter as shown in Table 2. The first bit (X) may be used to select the function.

X	Y	O/P	New Function
0	0	0	Buffer (Y)
0	1	1	
1	0	1	Inverter ($\bar{Y}$)
1	1	0	

Table 2. Results of dividing XOR Patterns.

So, we may use two neural nets, one to realize the buffer, and the other to represent the inverter. Each one of them may be implemented by using only one neuron. When realizing these two neurons, we implement the weights, and perform only one summing operation. The first input X acts as a detector to select the proper weights as shown in Fig.3. In a special case, for XOR function, there is no need to the buffer and the neural net may be represented by using only one weight corresponding to the inverter as shown in Fig.4. As a result of using cooperative modular neural nets, XOR function is realized by using only one neuron. A comparison between the new model and the two previous approaches is given in Table 3. It is clear that the number of computations and the hardware requirements for the new model is less than that of the other models.

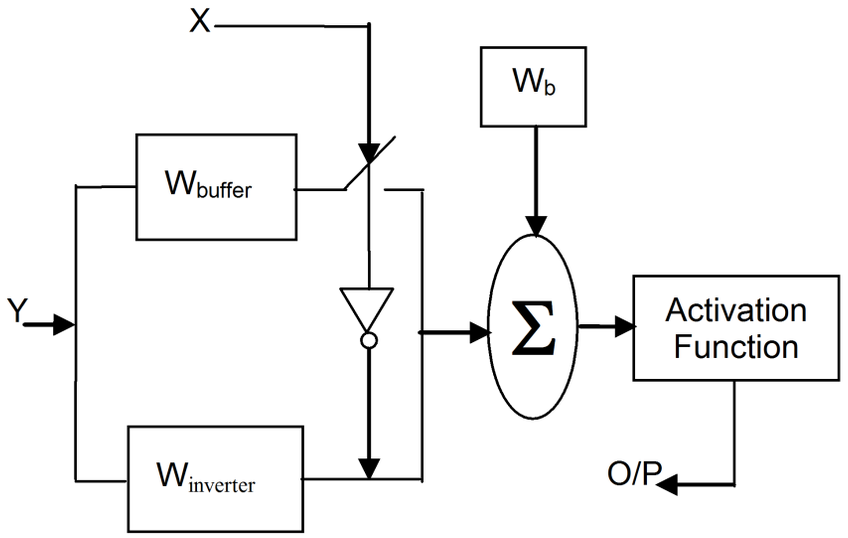

Figure 3. Realization of XOR function using modular neural nets.

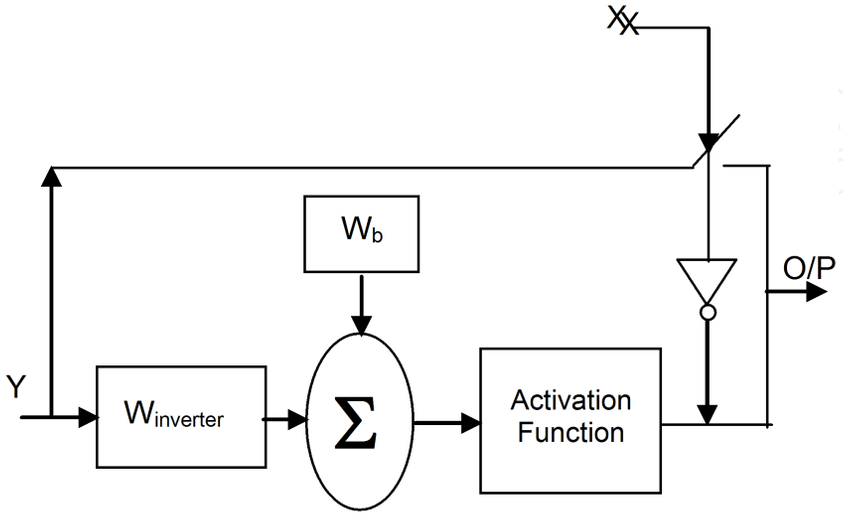

Figure 4. Implementation of XOR function using only one neuron.

http://dx.doi.org/10.5772/53021

Type of Comparison	First model (three neurons)	Second model (two neurons)	New model (one neuron)
No. of computations	O(15)	O(12)	O(3)
Hardware requirements	3 neurons, 9 weights	2 neurons, 7 weights	1 neuron, 2 weights, 2 switches, 1 inverter

Table 3. A comparison between different models used to implement XOR function.

2.2. Implementation of logic functions by using MNNs

Realization of logic functions in one bit level (X,Y) generates 16 functions which are (AND, OR, NAND, NOR, XOR, XNOR, $\overline{X}$,$\overline{Y}$, X, Y, 0, 1, $\overline{X}$Y, X$\overline{Y}$, $\overline{X}$+Y, X+$\overline{Y}$). So, in order to control the selection for each one of these functions, we must have another 4 bits at the input, thereby the total input is 6 bits as shown in Table 4.

Function	C1	C2	C3	C4	X	Y	O/p
AND	0	0	0	0	0	0	0
	0	0	0	0	0	1	0
	0	0	0	0	1	0	0
	0	0	0	0	1	1	1
.....	....	....	....	....	....	....	....
X+$\bar{Y}$	1	1	1	1	0	0	1
	1	1	1	1	0	1	0
	1	1	1	1	1	0	1
	1	1	1	1	1	1	1

Table 4. Truth table of Logic function (one bit level) with their control selection.

Non-MNNs can classify these 64 patterns using a network of three layers. The hidden layer contains 8 neurons, while the output needs only one neuron and a total number of 65 weights are required. These patterns can be divided into two groups. Each group has an input of 5 bits, while the MSB is 0 with the first group and 1 with the second. The first group requires 4 neurons and 29 weights in the hidden layer, while the second needs 3 neurons and 22 weights. As a result of this, we may implement only 4 summing operations in the hidden layer (in spite of 8 neurons in case of non-MNNs) where as the MSB is used to select which group of weights must be connected to the neurons in the hidden layer. A similar procedure is done between hidden and output layer. Fig. 5 shows the structure of the first neuron in the hidden layer. A comparison between MNN and non-MNNs used to implement logic functions is shown in Table 5.

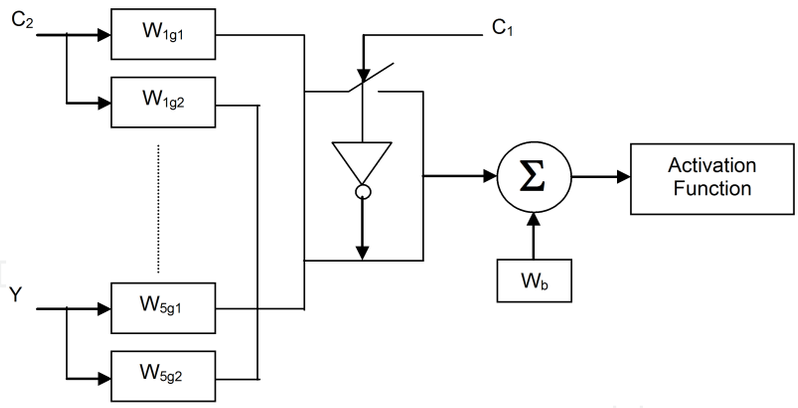

Figure 5. Realization of logic functions using MNNs (the first neuron in the hidden layer).

Type of Comparison	Realization using non MNNs	Realization using MNNs
No. of computations	O(121)	O(54)
Hardware requirements	9 neurons, 65 weights	5 neurons, 51 weights, 10 switches, 1 inverter

Table 5. A comparison between MNN and non MNNs used to implement 16 logic functions.

3. Implementation of 2-bits digital multiplier by using MNNs

In the previous section, to simplify the problem, we make division in input, here is an example for division in output. According to the truth table shown in Table 6, instead of treating the problem as mapping 4 bits in input to 4 bits in output, we may deal with each bit in output alone. Non MNNs can realize the 2-bits multiplier with a network of three layers with total number of 31 weights. The hidden layer contains 3 neurons, while the output one has 4 neurons. Using MNN we may simplify the problem as:

$$W = CA \tag{1}$$

$$\begin{aligned} X &= AD \otimes BC = AD(\bar{B} + \bar{C}) + BC(\bar{A} + \bar{D}) \\ &= (AD + BC)(\bar{A} + \bar{B} + \bar{C} + \bar{D}) \end{aligned} \tag{2}$$

$$Y = BD(\bar{A} + \bar{C}) = BD(\bar{A} + \bar{B} + \bar{C} + \bar{D}) \tag{3}$$

$$Z = ABCD \tag{4}$$

http://dx.doi.org/10.5772/53021

Equations 1, 2, 3 can be implemented using only one neuron. The third term in Equation 3 can be implemented using the output from Bit Z with a negative (inhibitory) weight. This eliminates the need to use two neurons to represent $\overline{A}$ and$\overline{D}$. Equation 2 resembles an XOR, but we must first obtain AD and BC. AD can be implemented using only one neuron. Another neuron is used to realize BC and at the same time oring (AD, BC) as well as anding the result with ($\overline{ABCD}$) as shown in Fig.6. A comparison between MNN and non-MNNs used to implement 2bits digital multiplier is listed in Table 7.

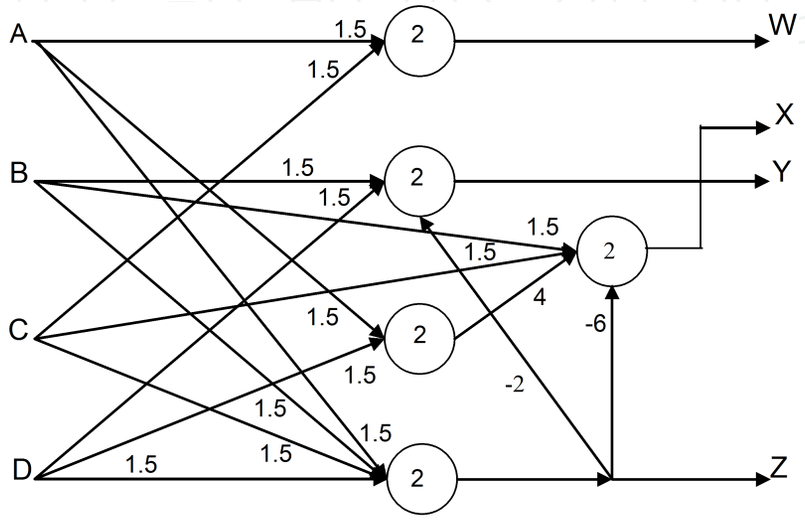

Figure 6. Realization of 2-bits digital multiplier using MNNs.

Input Patterns				Output Patterns			
D	C	B	A	Z	Y	X	W
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	1	1	0
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Table 6. Truth table of 2-bit digital multiplier.

Type of Comparison	Realization using non MNNs	Realization using MNNs
No. of computations	O(55)	O(35)
Hardware requirements	7 neurons, 31 weights	5 neurons, 20 weights

Table 7. A comparison between MNN and non-MNNs used to implement 2-bits digital multiplier.

4. Hardware implementation of MNNs by using reconfigurable circuits

Advances in MOS VLSI have made it possible to integrate neural networks of large sizes on a single-chip [10,12]. Hardware realizations make it possible to execute the forward pass operation of neural networks at high speeds, thus making neural networks possible candidates for real-time applications. Other advantages of hardware realizations as compared to software implementations are the lower per unit cost and small size system.

Analog circuit techniques provide area-efficient implementations of the functions required in a neural network, namely, multiplication, summation, and the sigmoid transfer characteristic [13]. In this paper, we describe the design of a reconfigurable neural network in analog hardware and demonstrate experimentally how a reconfigurable artificial neural network approach is used in implementation of arithmetic unit that including full-adder, full-subtractor, 2-bit digital multiplier, and 2-bit digital divider.

One of the main reasons for using analog electronics to realize network hardware is that simple analog circuits (for example adders, sigmoid, and multipliers) can realize several of the operations in neural networks. Nowadays, there is a growing demand for large as well as fast neural processors to provide solutions for difficult problems. Designers may use either analog or digital technologies to implement neural network models. The analog approach boasts compactness and high speed. On the other hand, digital implementations offer flexibility and adaptability, but only at the expense of speed and silicon area consumption.

4.1. Implementation of artificial neuron

Implementation of analog neural networks means that using only analog computation [14,16,18]. Artificial neural network as the name indicates, is the interconnection of artificial neurons that tend to simulate the nervous system of human brain [15]. Neural networks are modeled as simple processors (neurons) that are connected together via weights. The weights can be positive (excitatory) or negative (inhibitory). Such weights can be realized by resistors as shown in Fig. 7.

http://dx.doi.org/10.5772/53021

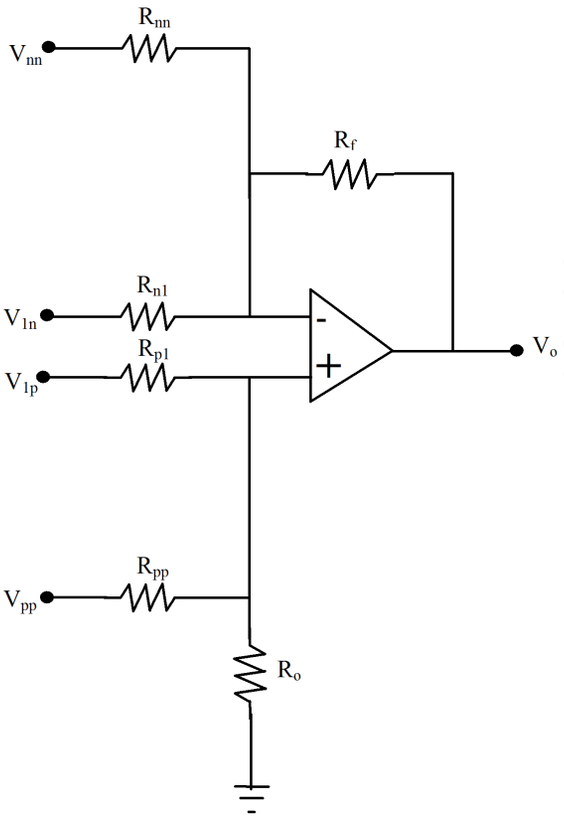

Figure 7. Implementation of positive and negative weights using only one opamp.

The computed weights may have positive or negative values. The corresponding resistors that represent these weights can be determined as follow [16]:

$$w_{in} = -R_f / R_{in} \qquad i = 1, 2, \ldots\ldots, n \tag{5}$$

$$Wpp = \frac{\left(1 + \sum_{i1}^{n} Win\right)\frac{Ro}{Rpp}}{\left(1 + \frac{Ro}{R1p} + \frac{Ro}{R2p} + \ldots\ldots\ldots\ldots\ldots + \frac{Ro}{Rpp}\right)} \tag{6}$$

The exact values of these resistors can be calculated as presented in [14,18]. The summing circuit accumulates all the input-weighted signals and then passes to the output through the transfer function [13]. The main problem with the electronic neural networks is the realization of resistors which are fixed and have many problems in hardware implementation [17]. Such resistors are not easily adjustable or controllable. As a consequence, they can be used neither for learning, nor can they be used for recall when another task needs to be solved. So the calculated resistors corresponding to the obtainable weights can be implemented by using CMOS transistors operating in continuous mode (triode region) as shown in Fig. 8. The equivalent resistance between terminal 1 and 2 is given by [19]:

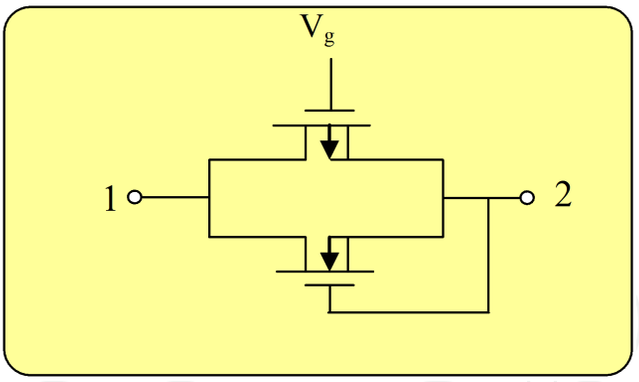

Figure 8. Two MOS transistor as a linear resistor.

$$R_{eq} = 1/\left[K\left(V_g - 2V_{th}\right)\right] \tag{7}$$

4.2. Reconfigurability

The interconnection of synapses and neurons determines the topology of a neural network. Reconfigurability is defined as the ability to alter the topology of the neural network [19]. Using switches in the interconnections between synapses and neurons permits one to change the network topology as shown in Fig. 9. These switches are called "reconfiguration switches".

The concept of reconfigurability should not be confused with *weight programmability*. Weight programmability is defined as the ability to alter the values of the weights in each synapse. In Fig. 9, weight programmability involves setting the values of the weights w_1, w_2, w_3,...., w_n. Although reconfigurability can be achieved by setting weights of some synapses to zero value, this would be very inefficient in hardware.

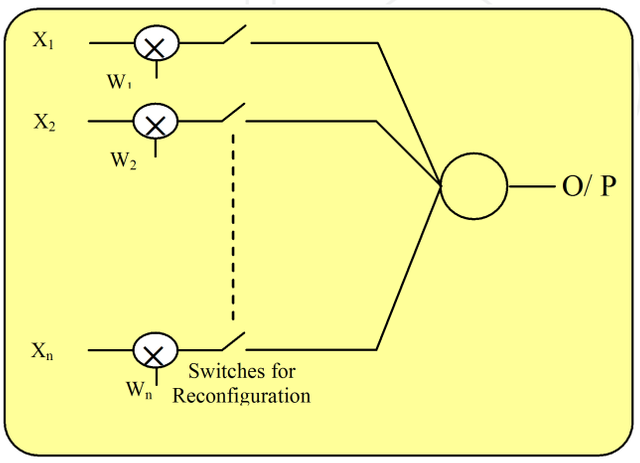

Figure 9. Neuron with reconfigurable switches.

http://dx.doi.org/10.5772/53021

Reconfigurability is desirable for several reasons [20]:

1. Providing a general problem-solving environment.
2. Correcting offsets.
3. Ease of testing.
4. Reconfiguration for isolating defects.

5. Design arithmetic and logic unit by using reconfigurable neural networks

In previous paper [20], a neural design for logic functions by using modular neural networks was presented. Here, a simple design for the arithmetic unit using reconfigurable neural networks is presented. The aim is to have a complete design for ALU by using the benefits of both modular and reconfigurable neural networks.

5.1. Implementation of full adder/full subtractor by using neural networks

Full-adder/full-subtractor problem is solved practically and a neural network is simulated and implemented using the back-propagation algorithm for the purpose of learning this network [10]. The network is learned to map the functions of full-adder and full-subtractor. The problem is to classify the patterns shown in Table 8 correctly.

I/P			Full-Adder		Full-Subtractor	
x	*y*	*z*				
	S	*C*	*D*	*B*		
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	1	1	1
0	1	1	1	0	1	0
1	0	0	0	1	0	1
1	0	1	1	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Table 8. Truth table of full-adder/full-subtractor

The computed values of weights and their corresponding values of resistors are described in Table 9. After completing the design of the network, simulations are carried out to test both the design and performance of this network by using H-spice. Experimental results confirm the proposed theoretical considerations. Fig. 10 shows the construction of full-adder/full-subtractor neural network. The network consists of three neurons and 12-connection weights.

I / P	Neuron (1)		Neuron (2)		Neuron (3)	
Weight	Resistance	Weight	Resistance	Weight	Resistance	
1	7.5	11.8 Ro	15	6.06 Ro	15	6.06 Ro
2	7.5	11.8 Ro	15	6.06 Ro	15	6.06 Ro
3	7.5	11.8 Ro	-10	0.1 Rf	-10	0.1 Rf
Bias	-10.0	0.1 Rf	-10	0.1 Rf	-10	0.1 Rf

Table 9. Computed weights and their corresponding resistances of full-adder/full-subtractor

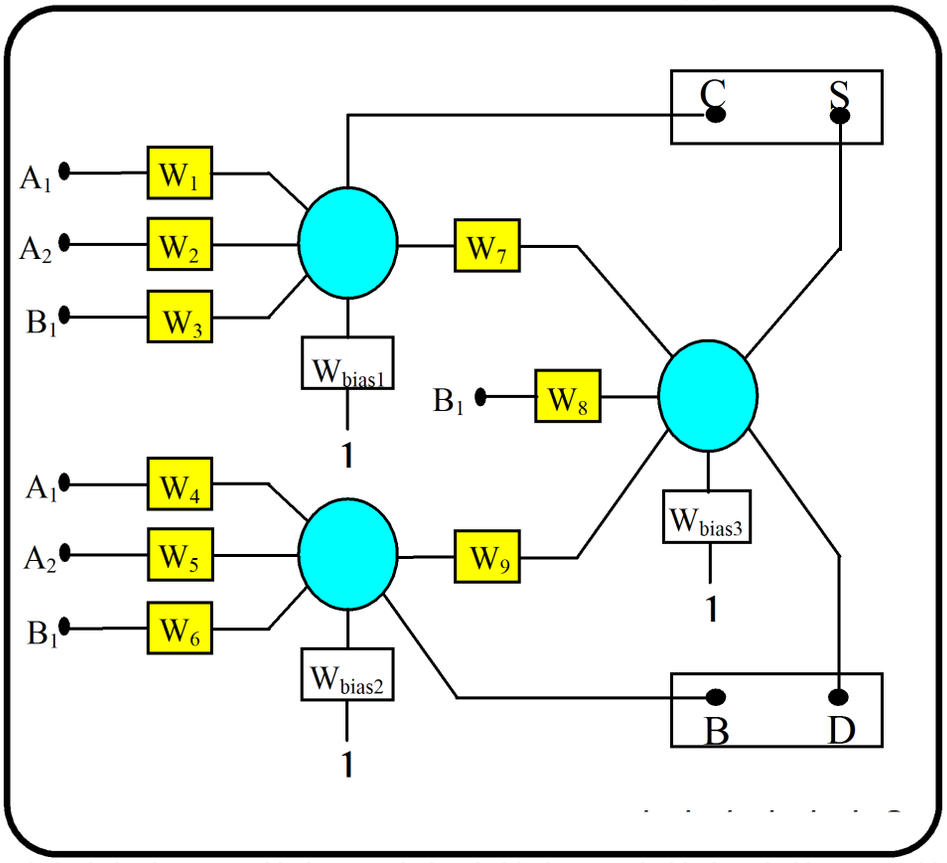

Figure 10. Full-adder/full-subtractor implementation.

5.2. Hardware implementation of 2-bit digital multiplier

2-bit digital multiplier can be realized easily using the traditional feed-forward artificial neural network [21]. As shown in Fig. 11, the implementation of 2-bit digital multiplier using the traditional architecture of a feed-forward artificial neural network requires 4-neurons, 20-synaptic weights in the input-hidden layer, and 4-neurons, 20-synaptic weights in the hidden-output layer. Hence, the total number of neurons is 8-neurons with 40-synaptic weights.

http://dx.doi.org/10.5772/53021

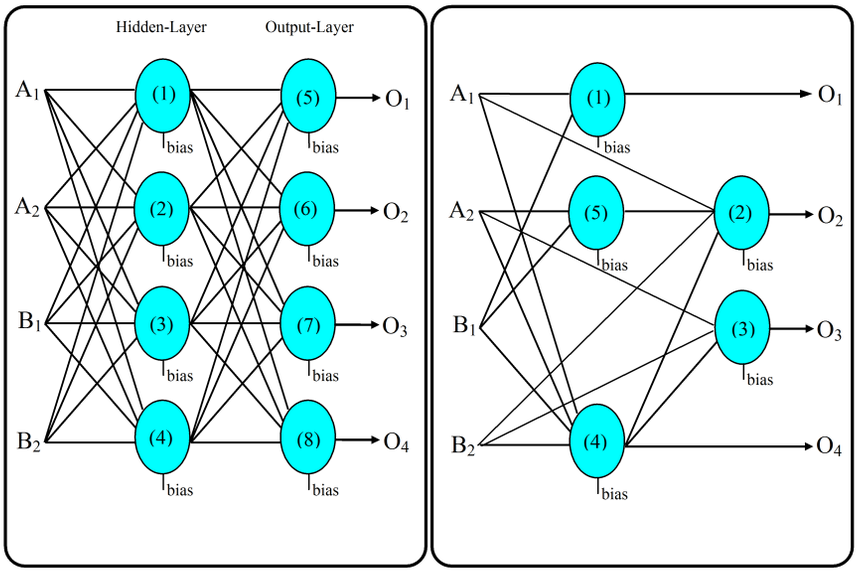

Figure 11. Bit digital multiplier using traditional feed-forward neural network

In the present work, a new design of 2-bit digital multiplier has been adopted. The new design requires only 5-neurons with 20-synaptic weights as shown in Fig. 12. The network receives two digital words, each word has 2-bit, and the output of the network gives the resulting multiplication. The network is trained by the training set shown in Table 10.

I/P				O/P			
B_2	B_1	A_2	A_1	O_4	O_3	O_2	O_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1

Table 10. 2-bit digital multiplier training set

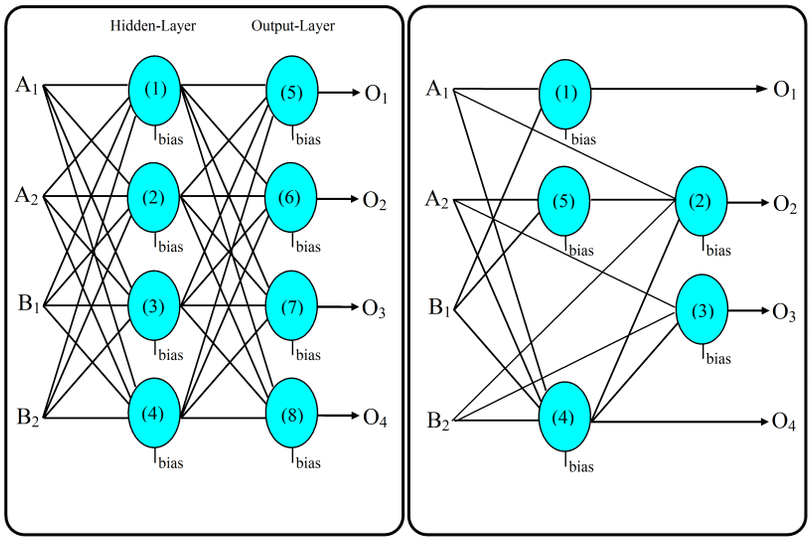

Figure 12. A novel design for2-Bit multiplier using neural network

During the training phase, these input/output pairs are fed to the network and in each iteration; the weights are modified until reached to the optimal values. The optimal value of the weights and their corresponding resistance values are shown in Table 11. The proposed circuit has been realized by hardware means and the results have been tested using H-spice computer program. Both the actual and computer results are found to be very close to the correct results.

Neuron	I/P	W. Value	Resistor
(1)	A_1	7.5	1200
	B_1	7.5	1200
	Bias	-10.0	100
(2)	A1	7.5	1450
	B2	7.5	1450
	Bias	-10.0	100
	N4	-30.0	33
	N5	20.0	618
(3)	A2	7.5	1200
	B2	7.5	1200
	bias	-10.0	100
	N4	-10.0	100
(4)	A1	3.0	1200
	A2	3.0	1200
	B1	3.0	1200
	B2	3.0	1200
	bias	-10.0	100
(5)	A2	7.5	1200
	B1	7.5	1200
	Bias	-10.0	100

Table 11. Weight values and their corresponding resistance values for digital multiplier.

http://dx.doi.org/10.5772/53021

5.3. Hardware implementation of 2-bit digital divider

2-bit digital divider can be realized easily using the artificial neural network. As shown in Fig. 13, the implementation of 2-bit digital divider using neural network requires 4-neurons, 20-synaptic weights in the input-hidden layer, and 4-neurons, 15-synaptic weights in the hidden-output layer. Hence, the total number of neurons is 8-neurons with 35-synaptic weights. The network receives two digital words, each word has 2-bit, and the output of the network gives two digital words one for the resulting division and the other for the resulting remainder. The network is trained by the training set shown in Table 12

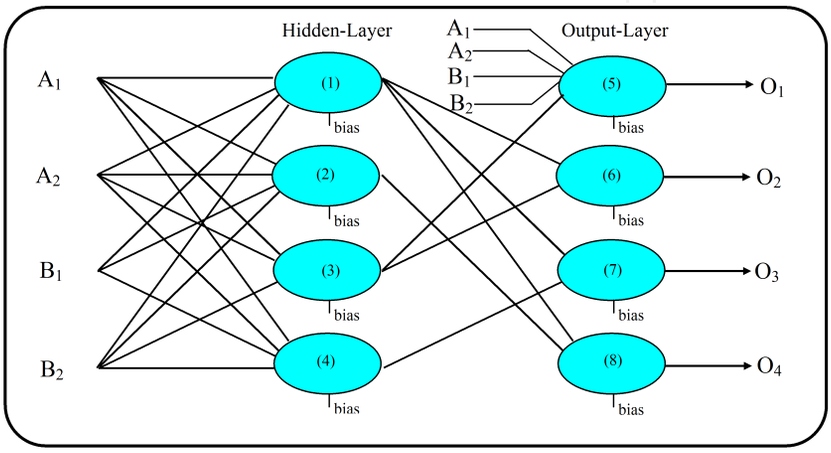

Figure 13. Bit digital divider using neural network.

I/P				O/P			
B_2	B_1	A_2	A_1	O_4	O_3	O_2	O_1
0	0	0	0	1	1	1	1
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	1	1	1	1
0	1	0	1	0	0	0	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	1	1	1
1	0	0	1	0	0	1	0
1	0	1	0	0	0	0	1
1	0	1	1	1	0	0	0
1	1	0	0	1	1	1	1
1	1	0	1	0	0	1	1
1	1	1	0	0	1	0	1
1	1	1	1	0	0	0	1

Table 12. 2-bit digital dividier training set

The values of the weights and their corresponding resistance values are shown in Table 13.

Neuron	I/P	W. Val.	Resistor
(1)	A1	-17.5	56
	A2	-17.5	56
	B1	5	2700
	B2	5	2700
	Bias	5	2700
(2)	A1	7.5	1200
	A2	7.5	1200
	B1	-10	100
	B2	7.5	1200
	Bias	-17.5	56
(3)	A1	7.5	1200
	A2	-10	100
	B2	7.5	1200
	Bias	-10	100
(4)	A1	-4.5	220
	A2	7.5	1200
	B1	7.5	1200
	B2	-4.5	220
	Bias	-10	100
(5)	A1	-20	50
	A2	-30	33
	B1	10	1200
	B2	25	500
	N3	-25	40
	Bias	17.5	700
(6)	N1	10	1000
	N3	10	1000
	Bias	-5	220
(7)	N1	10	1000
	N4	10	1000
	Bias	-5	220
(8)	N1	10	1000
	N2	10	1000
	Bias	-5	220

Table 13. Weight values and their corresponding resistance values for digital divider.

http://dx.doi.org/10.5772/53021

The results have been tested using H-spice computer program. Computer results are found to be very close to the correct results.

Arithmetic operations namely, addition, subtraction, multiplication, and division can be realized easily using a reconfigurable artificial neural network. The proposed network consists of only 8-neurons, 67-connection weights, and 32-reconfiguration switches. Fig. 14 shows the block diagram of the arithmetic operation using reconfigurable neural network. The network includes full-adder, full-subtractor, 2-bit digital multiplier, and 2-bit digital divider. The proposed circuit is realized by hardware means and the results are tested using H-spice computer program. Both the actual and computer results are found to be very close to the correct results.

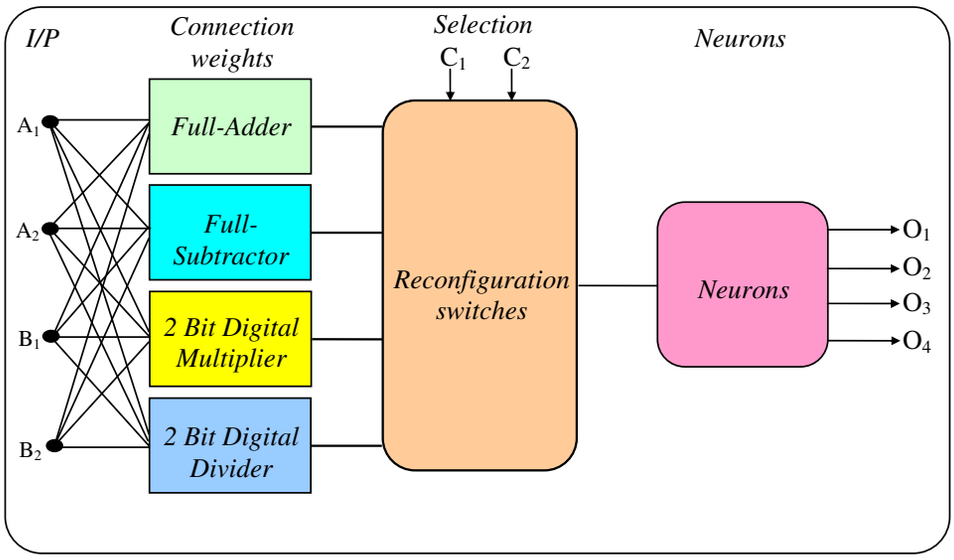

Figure 14. Block diagram of arithmetic unit using reconfigurable neural network.

The computed values of weights and their corresponding values of resistors are described in Tables 9,10,116. After completing the design of the network, simulations are carried out to test both the design and performance of this network by using H-spice. Experimental results confirm the proposed theoretical considerations as shown in Tables 14,15.

6. Conclusion

We have presented a new model of neural nets for classifying patterns that appeared expensive to be solved using conventional models of neural nets. This approach has been introduced to realize different types of logic problems. Also, it can be applied to manipulate non-binary data. We have shown that, compared to non MNNs, realization of problems using MNNs resulted in reduction of the number of computations, neurons and weights.

I/p			Neuron(1)		Neuron(2)		Neuron(3)	
X	*Y*	*Z*	Practical	Simulated	Practical	Simulated	Practical	Simulated
0	0	0	-2.79	-3.4157	-2.79	-3.4135	-2.79	-3.4135
0	0	1	-2.73	-2.5968	3.46	3.3741	3.46	3.3741
0	1	0	-2.73	-2.5968	3.46	3.2741	3.46	3.3741
0	1	1	3.46	3.3761	3.46	3.4366	-2.75	-3.3081
1	0	0	-2.73	-2.5968	-2.79	-3.4372	3.46	3.3741
1	0	1	3.46	3.3761	-2.75	-3.3081	-2.75	-3.3081
1	1	0	3.46	3.3761	-2.75	-3.3081	-2.75	-3.3081
1	1	1	3.46	3.4231	3.48	3.4120	3.48	3.4120

Table 14. Practical and Simulation results after the summing circuit of full-adder/full-subtractor

Neuron (1)		Neuron (2)		Neuron (3)		Neuron (4)		Neuron (5)	
Pract.	Sim.	Pract.	Sim.	Pract.	Sim.	Pract.	Sim.	Pract.	Sim.
-2.79	-3.415	-2.79	-3.409	-2.79	-3.413	-2.79	-3.447	-2.79	-3.415
-2.34	-2.068	-2.72	-2.498	-2.79	-3.314	-2.78	-3.438	-2.79	-3.415
-2.79	-3.415	-2.79	-3.409	-1.63	-1.355	-2.78	-3.438	-2.34	-2.068
-2.34	-2.068	-2.72	-2.498	-1.63	-1.355	-2.78	-3.423	-2.34	-2.068
-2.34	-2.068	-2.79	-3.409	-2.79	-3.413	-2.78	-3.438	-2.34	-2.068
3.46	3.390	-2.72	-2.498	-2.79	-3.413	-2.78	-3.423	-2.34	-2.068
-2.34	-2.068	3.45	3.397	-1.63	-1.355	-2.78	-3.423	3.46	3.390
3.46	3.390	3.45	3.424	-1.63	-1.355	-2.74	-3.384	3.46	3.390
-2.79	-3.415	-2.72	-2.498	-1.63	-1.355	-2.78	-3.438	-2.79	-3.415
-2.34	-2.068	3.45	3.373	-1.63	-1.355	-2.78	-3.423	-2.79	-3.415
-2.79	-3.415	-2.72	-2.498	3.45	3.399	-2.78	-3.423	-2.34	-2.068
-2.34	-2.068	3.45	3.373	3.45	3.399	-2.74	-3.384	-2.34	-2.068
-2.34	-2.068	-2.72	-2.498	-1.63	-1.355	-2.78	-3.423	-2.34	-2.068
3.46	3.390	3.45	3.373	-1.63	-1.355	-2.74	-3.384	-2.34	-2.068
-2.34	-2.068	3.45	3.373	3.45	3.399	-2.74	-3.384	3.46	3.390
3.46	3.390	-2.73	-3.398	-2.70	-2.710	1.86	2.519	3.46	3.390

Table 15. Practical and Simulation results after the summing circuit of 2-bit digital multiplier

Author details

Hazem M. El-Bakry

Faculty of Computer Science & Information Systems, Mansoura University, Egypt

http://dx.doi.org/10.5772/53021

References

[1] J, Murre, Learning and Categorization in Modular Neural Networks, Harvester Wheatcheaf. 1992.

[2] R. Jacobs, M. Jordan, A. Barto, Task Decomposition Through Competition in a Modular Connectionist Architecture: The what and where vision tasks, Neural Computation 3, pp. 79-87, 1991.

[3] G. Auda, M. Kamel, H. Raafat, Voting Schemes for cooperative neural network classifiers, IEEE Trans. on Neural Networks, ICNN95, Vol. 3, Perth, Australia, pp. 1240-1243, November, 1995.

[4] G. Auda, and M. Kamel, CMNN: Cooperative Modular Neural Networks for Pattern Recognition, Pattern Recognition Letters, Vol. 18, pp. 1391-1398, 1997.

[5] E. Alpaydin, , Multiple Networks for Function Learning, Int. Conf. on Neural Networks, Vol.1 CA, USA, pp. 9-14, 1993.

[6] A. Waibel, Modular Construction of Time Delay Neural Networks for Speach Recognition, Neural Computing 1, pp.39-46.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representation by error backpropagation, Parallel distributed Processing: Explorations in the Microstructues of Cognition, Vol. 1, Cambridge, MA:MIT Press, pp. 318-362, 1986.

[8] K. Joe, Y. Mori, S. Miyake, Construction of a large scale neural network: Simulation of handwritten Japanese Character Recognition, on NCUBE Concurrency 2 (2), pp. 79-107.

[9] H. M. El-bakry, and M. A. Abo-elsoud, Automatic Personal Identification Using Neural Nets, The 24th international Conf. on Statistics computer Science, and its applications, Cairo, Egypt, 1999.

[10] Srinagesh Satyanarayna, Yannis P. Tsividis, and Hans Peter graf, "A Reconfigurable VLSI Neural Network," IEEE Journal of Solid State Circuits, vol. 27, no. 1, January 1992.

[11] E. R. Vittos, "Analog VLSI Implementation of Neural Networks," in proc. Int. Symp. Circuits Syst. (new Orleans, LA), 1990, pp. 2524-2527.

[12] H. P. graf and L. D. Jackel, "Analog Electronic Neural Network Circuits," IEEE Circuits Devices Mag., vol. 5, pp. 44-49, July 1989.

[13] H. M. EL-Bakry, M. A. Abo-Elsoud, and H. H. Soliman and H. A. El-Mikati " Design and Implementation of 2-bit Logic functions Using Artificial Neural Networks ," Proc. of the 6th International Conference on Microelectronics (ICM'96), Cairo, Egypt, 16-18 Dec. , 1996.

[14] Simon Haykin, "Neural Network : A comprehensive foundation", Macmillan college publishing company, 1994.

[15] Jack M. Zurada, "Introduction to Artificial Neural Systems," West Publishing Company, 1992.

[16] C. Mead, and M. Ismail, "Analog VLSI Implementation of Neural Systems," Kluwer Academic Publishers, USA, 1989

[17] H. M. EL-Bakry, M. A. Abo-Elsoud, and H. H. Soliman and H. A. El-Mikati " Implementation of 2-bit Logic functions Using Artificial Neural Networks ," Proc. of the 6th International Conference on Computer Theory and Applications, Alex., Egypt, 3-5 Sept. , 1996, pp. 283-288.

[18] I. S. Han and S. B. Park, "Voltage-Controlled Linear Resistor by Using two MOS Transistors and its Applications to RC Active Filter MOS Integration," Proceedings of the IEEE, Vol.72, No.11, Nov. 1984, pp. 1655-1657.

[19] Laurene Fausett, "Fundamentals of Neural Network : Architectures, Algorithms, and Applications," Prentice Hall International.

[20] H. M. El-bakry, "Complexity Reduction Using Modular Neural Networks," Proc. of IEEE IJCNN'03, Portland, Oregon, pp. 2202-2207, July, 20-24, 2003.

[21] H. M. El-Bakry, and N. Mastorakis "A Simple Design and Implementation of Reconfigurable Neural Networks Detection," Proc. of IEEE IJCNN'09, Atlanta, USA, June 14-19, 2009, pp. 744-750.

IntechOpen